Synchronization and Adaptation of Systems of Nonlinear Oscillators

by Jennifer E. Smith

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAY 2005 [June 2005]

Signature of Author: [illegible]
Department of Mechanical Engineering May 6, 2005

Certified by: [illegible]
Jean-Jacques E Slotine
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by: [illegible]
Ernest G Cravalho
Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

MASSACHUSETTS INSTITUTE OF [illegible]
JUN [illegible]
LIBRARIES

1

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 0 8 2005
LIBRARIES

ARCHIVES

Synchronization and Adaptation of Systems of Nonlinear Oscillators

Jennifer E. Smith

May 6, 2005

Abstract

This paper discusses the synchronization and adaptation of systems of nonlinear oscillators. The paper presents several simulations, which attempt to illustrate these systems as clearly as possible, in particular in the presence of leaders.

Thesis Supervisor: Jean-Jacques E Slotine
Title: Professor of Mechanical Engineering

1 Introduction

A flock of birds, a school of fish, brain synapses, or a crowd of people: what do groups have in common? In each of these situations, one bird, fish, neuron, or person moves based on its neighbors without having knowledge of the greater surroundings. These situations, however, are difficult to predict or model with the grace and smoothness with which they occur in nature. For example, a school of fish swims around obstacles, and changes directions to avoid predators, with great speed. The direction of motion of each fish is similar and depends on the one in front of it. Because of this behavior, the school moves fluidly, almost as if it were one individual, instead of being composed of multiple beings. Modeling this smoothness can be complicated.

In the above cases, each individual adjusts its parameters, for example direction, based on only the movements of the others within a certain radius. This radius is analogous to the field of vision in cases such as a flock of birds or a school of fish. These various situations can be modeled mathematically using systems of nonlinear oscillators. In such systems, each oscillator represents one individual of the group, such as a bird, fish, neuron, or person. In the oscillator equations, different parameters of the oscillator are adjusted based on the parameters of other oscillators. Different parameters linking the individual to its neighbors are used in each simulation. These variations model different physical situations.

This paper discusses simulations that explore the reactions of systems of nonlinear oscillators to adaptation and synchronization. For example, if one individual does not adapt or synchronize with the rest of the group, then there is an impact on the final state of the system. This individual is called a "leader." The situations discussed in this paper deal with one or more leaders. Also, this paper presents simulations that represent the reactions of different systems of nonlinear oscillators to various initial conditions and couplings. The goal of these simulations is to make the complex behaviors of the mathematical models easier to visualize.

Section 2 discusses the origin of the FitzHugh–Nagumo model, and the equations that represent it. Next, Section 2 talks about how these oscillators can be coupled, and the effects of both coupled adaptation and synchronization. Section 3 begins by describing the visual representation of the values of variables in these equations through the simulations. Then Section 3 goes on to describe the third equation required for adaptation.

Sections 4 through 8 give the results of specific simulations that were run. Section 4 gives an example of simple synchronization of a system of oscillators. Section 5 talks about a simulation where the system of oscillators adapts to a common phase without the synchronization term present in the equations. Section 6 talks about a simulation where the system synchronizes and adapts at the same time. Section 7 talks about a simulation where the system starts out with the same parameters, but different phases. The system is then coupled for synchronization. After synchronization, the frequency of one oscillator is changed, and the system adapts to this new frequency. Lastly, Section 8 presents a simulation where the system has two different leaders. One leader had fixed parameters, and the other was not, at first, coupled for synchronization.

2 Background

The simulations presented in this paper use the equations that represent the model known as the "FitzHugh–Nagumo model." This model was developed to represent the spiking impulses in neurons. It is classified as a "cubic Bonhoeffer–van der Pol model." Cubic Bonhoeffer–van der Pol models are used to find the voltage across membranes. This voltage is given by the variable V in the following equations. As shown in [2, pp. 20-21], the FitzHugh–Nagumo model is a simplification of the Hodgkin–Huxley model, which uses the system of the following form:

$$\begin{aligned} \dot{V} &= V + V^3/3 - W + I, \\ \dot{W} &= \phi(V + a - bW). \end{aligned}$$

where $\dot{V}$ and $\dot{W}$ are the derivatives with respect to time. In this system of equations, I is the external current applied to the system, and W is the recovery variable. All of these variables are dimensionless. The parameters ϕ, a, and b are positive constants.

Similarly, the FitzHugh–Nagumo model uses the system below:

$$\begin{aligned} \dot{v} &= v(\alpha - v)(v - 1) - w + I, \\ \dot{w} &= \beta v - \gamma w. \end{aligned}$$

In these equations, the parameters are α, I, β, and γ. As in the Hodgkin–Huxley equations, in the FitzHugh–Nagumo model, I stands for the applied current. Again, v is the potential across the membrane, and w is a recovery variable. In this case, α, β, I, and γ are positive constants.

The system of equations shown above can be coupled, modeling a situation in which numerous oscillators interact with each other. The ith oscillator is described by a system of equations for v_i and w_i. In the case of the simulations in Section 4, and Sections 6 through 8, the effect of coupling two oscillators v_i, w_i and v_j, w_j is that each oscillator influences the other's change in value. The system becomes the following:

$$\begin{aligned} \dot{v}_i &= v_i(a_i - v_i)(v_i - 1) - w_i + I_i + \sum_{j=1}^{N} k_{ij}(v_j - v_i), \\ \dot{w}_i &= \beta_i v_i - \gamma_i w_i. \end{aligned} \tag{1}$$

where N is the number of oscillators.

The parameters k_{ij} in (1) are changed in order to achieve synchronization. The parameter I_i governs the frequency of oscillation, and so is changed in order for adaptation to take place, as described in Section 3. The value of k_{ij} controls the way in which the oscillators synchronize in phase. If the value of k_{ij} is nonzero for two oscillators with indices i and j, then these two oscillators are coupled, and eventually synchronize to a common phase. The value of k_{ij} may depend on many different factors depending on the situation being modeled. These factors might be whether the oscillators are within a certain radius of each other or whether they are joined in network. In the simulations below, k_{ij} is dependent on the distance between the ith and jth oscillators. Each oscillator is coupled with every oscillator that is within a certain radius of it.

3 Methods

Systems of FitzHugh–Nagumo oscillators were used in several simulations. These simulations show the effects of both synchronization and adaptation on each system. The function ODE45 in MATLAB and the graphics program OpenGL were used to create a visual output based on the mathematical equations.

These visual representations are based around a gas-like movement of dots. Each dot represents one oscillator, and the values of the parameters are computed using the system of equations presented in Section 2. These dots move inside a 2-dimensional region, where they bounce off of the edges with elastic collisions. These dots are given random initial positions and directions, all dots are given the same velocity, and the velocity remains constant throughout the simulations. Each system of oscillator equations is then given various initial conditions for its parameters. The values of the initial parameters are different for the various situations. For example, in one simulation, the oscillators are given random phases and frequencies as initial conditions. In this simulations, a first frame resembles Figure 1 from Section 5.

In the example shown in Figure 1, the different sizes of the dots represent the different initial phases given to the oscillators. As the value of w_i for each oscillator varies from positive to negative values, the oscillators blink "on" and "off." A large positive value of an oscillator corresponds to a large radius for the respective dot. As v_i moves towards a negative value, the radius fades to 0 and the dot is turned "off." In this way, Figure 1 represents a variety of phases for the various oscillators. Similarly, the frequency of each oscillator is represented by the shade of grey of the dot. This frequency is proportional to the value of I_i in the system (1). In Figure 1, the variety of shades of grey of the dots show the variety of frequencies in the initial conditions.

The simulations use these basic representations to show many varieties of synchronization and adaptation. In the system (1), the value of k_{ij} is based on the distance between the positions of the oscillators with indices i and j. These simulations use a circular radius of synchronization and adaptation. If the distance between the oscillators is within this radius, then the value of k_{ij} is a positive constant. However, if the distance is greater, then the value of k_{ij} is 0, and the oscillators do not contribute to each others' change in value.

The adaptation of the oscillators works in a similar way, but deals with the parameters α_i, β_i, γ_i, and I_i. In the case of adaptation, α_i, β_i, γ_i, and I_i are

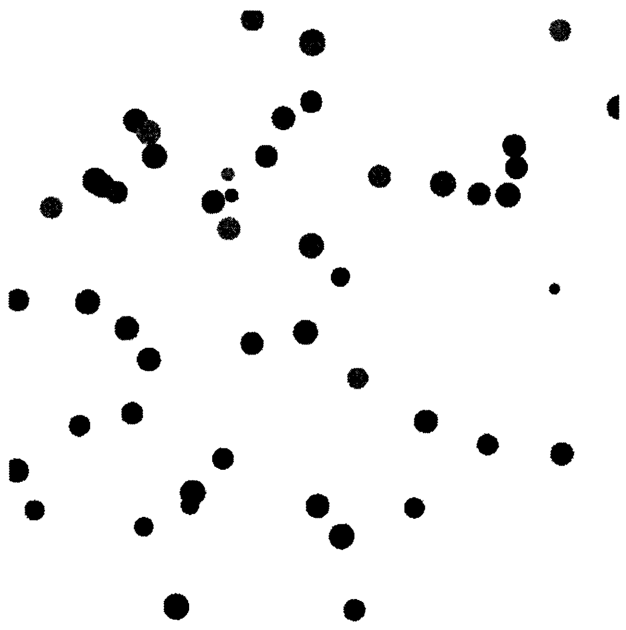
Figure 1: Random Initial Conditions.

no longer constants, but instead change based on the adaptation equation (2) below. Again, two different oscillators with indices i and j only have an effect on each others parameters α_i, β_i, γ_i, and I_i if they are within the set radius. The equation for adaptation in (2) below is added to those of v_i and w_i. The vector A_i is composed of the parameters α_i, β_i, γ_i, and I_i. The equation for A_i controls the changing of the parameters, α_i, β_i, γ_i, and I_i. Therefore, as A_i adapts and takes on a common value for all oscillators, each parameter adapts to one value for all oscillators. For the FitzHugh–Nagumo spiking neuron model, Equation (2) is the third equation that controls adaptation:

$$\dot{A}_i = A_i + QW_i^T D_i \tag{2}$$

where

$$Q = \begin{pmatrix} .01 & 0 & 0 & 0 \\ 0 & .01 & 0 & 0 \\ 0 & 0 & .01 & 0 \\ 0 & 0 & 0 & .05 \end{pmatrix},$$

$$W_i = \begin{pmatrix} v_i(v_i - 1) & 1 & 0 & 0 \\ 0 & 0 & -w_i & v_i \end{pmatrix},$$

$$D_i = \begin{pmatrix} \sum_{j=1}^N k_{ij}(v_j - v_i) \\ \sum_{j=1}^N k_{ij}(w_j - w_i) \end{pmatrix}.$$

This system converges to a constant value of A_i. In [3, p. 8], convergence of parameters is proved for a generalized system of nonlinear oscillator equations. Other systems of nonlinear oscillators, such as the system of equations representing the FitzHugh–Nagumo model, converge in the same way as this generalized system. This system is of the following form:

$$\dot{x}_i = f(x_i, a_i, t) + \sum_{j=1}^{N} k_{ji}(x_j - x_i),$$

$$\dot{a}_i = P_i W^T(x_i, t) + \sum_{j=1}^{N} k_{jk}(x_j - x_i).$$

In this generalized case, A_i converges to a common value for all oscillators. This convergence of A_i is proved using a lyapunov-like function. Also, as shown in [3, p. 8], a similar form of this function can be used to improve the convergence rate of the system. This improved convergence equation for $\dot{a}$ is as follows:

$$\dot{a}_i = a_i + Q_i W^T(x_i, t) \sum_{j=1}^{N} K_{ji}(x_j - xi)$$

The adaptation equations used for these simulations are based on this improved adaptation model. In the our specific case, $\dot{A}_i$ is composed of the parameter vector for the FitzHugh–Nagumo model. The paper [3] proves the synchronization and adaptation of the model used in the following simulations. The simulations should give further evidence of synchronization and adaptation of the system in various situations, such as in the presence of a leader.

For each iteration of the simulation, the vector A_i is updated; in other words, each of the parameters α_i, β_i, γ_i, and I_i is updated. The result is a system moving towards having one common frequency. Similarly, synchronization moves along in a frame-by-frame process. The frequency of the leader, however, does not change with time. Therefore, the simulations should show a clear step-by-step progression towards this common phase and frequency. In this way, the simulations should show multiple parameters and their evolution over time in a clear and precise manner. The simulations in Sections 4 through 8 represent a variety of possible situations, and show adaptation and synchronization with leaders; for more information see [3, pp. 3–5].

4 Simulation 1

Simulation 1 is an example of synchronization. In this simulation, the oscillators were allowed to oscillate uncoupled for the first 30 frames. This time period shows their initial conditions. There were 81 oscillators present in this simulation, and initially they had the same frequency, and random phase. Three of these first 30 frames are shown in Figure 2.

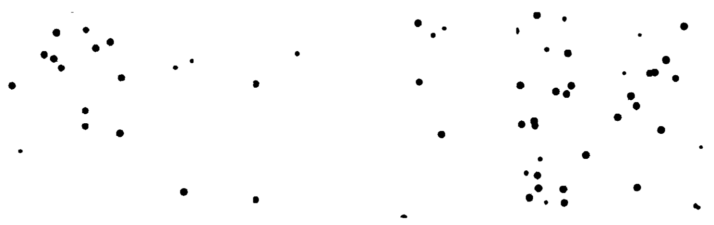
Figure 2: Out of sync oscillators of the same frequency.

This series of frames shows every third frame from the beginning of the simulation. As can be seen in Figure 2, there was never a frame where no oscillators were present. Consequently, the oscillators were completely out of phase during this initial time period. Also, the shade of the oscillators is constant, representing their common frequency.

After this period of initial conditions ended , the oscillators were coupled, using the equations in (1). As can be seen in Figure 3 and Figure 4, the oscillators slowly took on a common phase. First, as shown in Figure 3, blank frames appeared as the the oscillators became closer and closer in phase. Eventually, as shown in Figure 4, the oscillators blinked "on" and "off" mostly at the same time. The common phase of the oscillators can also be seen from the fact that the oscillators share a common size. The few last points that are slightly out of phase synchronized to the common phase in the end of the simulation.

Figure 3: Oscillators beginning to synchronize.

Figure 4: Oscillators fully synchronized.

This simulation gives insight into how the system of oscillators synchronized to a common frequency. A simple plot alone can not give the viewer all of this information. As can be seen in Figure 5, the plot of 81 synchronizing oscillators is very complicated. It is hard to determine one oscillator from another on this plot. Also it is hard to see which oscillators are close to each other in location.

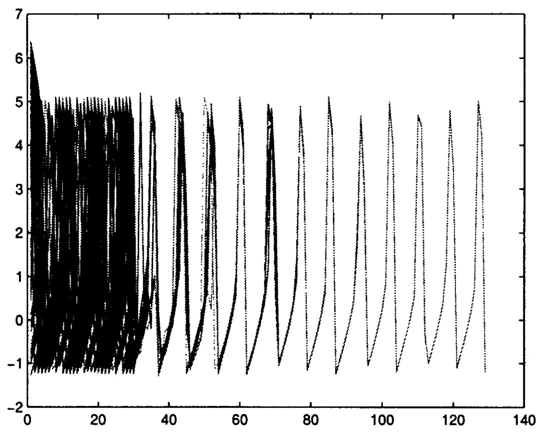

Figure 5: Plot of v_i vs time for 81 synchronizing oscillators

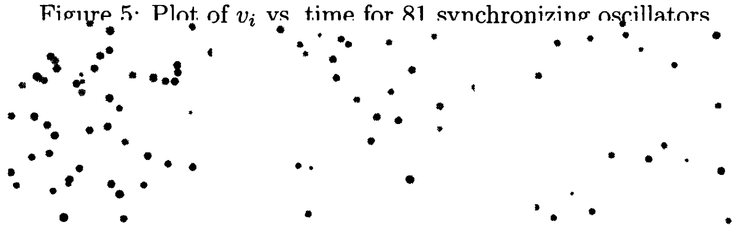

Figure 6: Oscillators of various frequencies out of synchronization.

As can be seen from the simulation, regions of oscillators blinked "on" and "off" together. Several oscillators in one area of the screen, close together, tended to take on common phase more quickly. This information can not be seen from simple plots of the data. In this way, the simulation is helpful in learning how the oscillators synchronized.

5 Simulation 2

Simulation 2 is an example of adaptation, based on the equations given in (2). Similar to the period of initial conditions from Simulation 1, the initial conditions for this simulation were uncoupled. In Simulation 2, however, the initial frequency for each oscillator was random as well as the phase. As can be seen in Figure 6, there was never a time during the 30 frames of initial conditions when no oscillators were present in a frame. Also, during this period, each oscillator had a different frequency. This behavior can be seen by the different shades of the oscillators.

After the initial 30 frames, the oscillators were coupled for adaptation, but not synchronization. The third equation in (2) was added to each oscillator's system of equations. The value of k for each oscillator in the system (1) was zero, so they were not coupled for synchronization. The results are shown in Figure 7 and Figure 8.

When adaptation began, the oscillators began to take on the same phase.

Figure 7: Oscillators beginning to adapt.

Figure 8: Oscillators continuing to adapt.

As shown in Figure 7, frames began to appear without any oscillators present. Also, oscillators in a section of the frame started to take on the same shade of grey. This represents their common frequency.

As the system continued to adapt, the oscillators all took on the same phase. The majority of the oscillators became one shade of gray. Fewer and fewer had different frequencies. As can be seen in Figure 8, eventually the oscillators took on one common grey scale. Also, they blinked "on" and "off" at the same time. Along with their common size, this behavior shows their common phase. At this point the system had adapted, taking on a common frequency and phase.

Again, this simulation is a much clearer representation of the system than a simple plot. As can be seen in Figure 9, a simple plot of v versus time does not show the frequencies of the oscillators very well. From Figure 9, it is only possible to see that the oscillators had in fact adapted to a common phase. The different initial frequencies of the oscillators are even difficult to distinguish from each other.

A plot of I_i versus time does not give a full explanation of the system either. As can be seen in Figure 10, it is hard to determine what the frequency of each oscillator is at each point in its oscillation. These details of the system can be easily seen through the simulation.

6 Simulation 3

Simulation 3 is a logical follow up to the first two simulations, where each synchronization and adaptation took place independently. Simulation 3 shows the effects of adaptation and synchronization at the same time. As in Simulation 2, all of the oscillators start out with random frequencies and phases. The 30 frames of initial conditions are similar to Figure 6.

After the initial conditions were run for 30 frames, both synchronization and

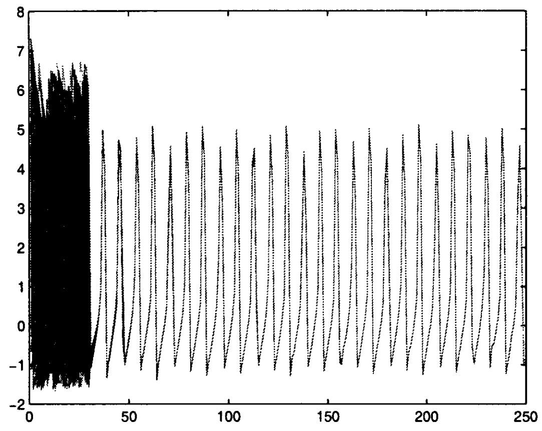

Figure 9: Plot of v_i vs. time.

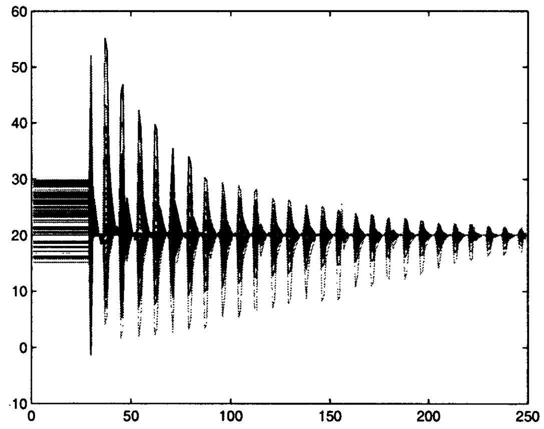

Figure 10: Plot of I_i vs. time.

adaptation were turned on at the same time. Synchronization took place much more quickly than adaptation; the result is as shown in Figure 11.

As can be seen in Figure 11, the oscillators begin to reach a common phase before their frequencies have completely adapted. In this case, common periods develop where all the oscillators are turned "on" or "off." Not all of the shades have become similar, but there is still a large period where none of the oscillators are in the frame, because all of the oscillators have a negative value. As the simulation progressed, the oscillators all adapted to a common frequency. This result can be seen in Figure 12.

As can be seen, the system has both synchronized and adapted to a final common state. All the oscillators share one phase and one frequency. The leader's frequency was set to different values, leaving the other frequencies as random. The result shows the effects of the leader. The final state of the system

Figure 11: Synchronization and adaptation, Phase 1.

Figure 12: Synchronization and adaptation, Phase 2.

always takes on the frequency stated in the initial conditions of the leader.

This simulation appears similar to Simulation 2. At the end of both simulations, the oscillators are in the same state. The oscillators, which initially have random phase and frequency, all take on one common phase and frequency. The presence of the synchronization term in the system greatly increased the rate at which this process occurred. In Simulation 2, it took the oscillators 250 frames to reach a point of common frequency and phase. In Simulation 3, however, this process took only approximately 100 frames. This difference can be seen from comparison of Figure 13 and Figure 9. The improved rate of adaptation can also be seen in Figure 14 and Figure 10. This rate can also be seen through the simulations.

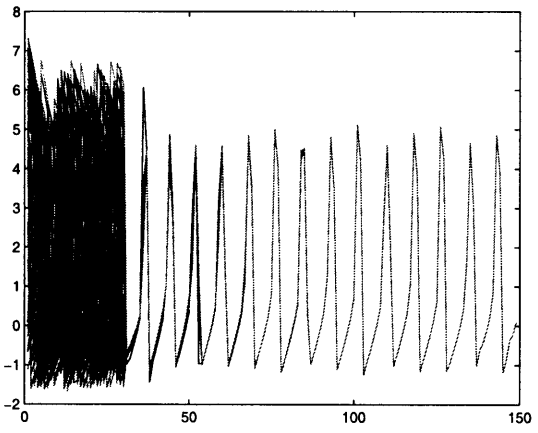

Figure 13: Plot of v_i vs. time for 81 synchronizing and adapting oscillators.

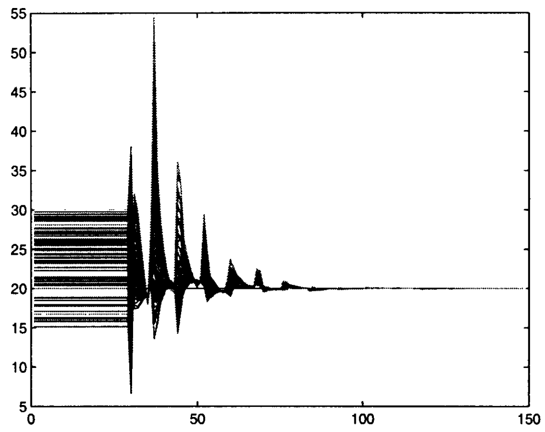

Figure 14: Plot of I_i vs. time for 81 synchronizing and adapting oscillators.

Figure 15: Synchronization.

7 Simulation 4

This simulation was also a combination of Simulation 1 and Simulation 2. The system, in this case, was made to synchronize and then to adapt to a common frequency. The initial conditions were similar to Simulation 1. The oscillators started out with a common frequency, but out of phase. This behavior can be seen in Figure 2. In this case, the oscillators were all of the same shade, representing the common frequency, but were of varied sizes, showing their dissimilar phase.

After the first 30 frames of initial conditions, the oscillators were coupled and underwent synchronization. Given the same radius as in Simulation 2 and Simulation 3, the oscillators took on a common phase. This process can be seen in Figure 15.

Again, these oscillators took on similar "on" and "off" periods, and blinked together, sharing a common size in all frames. Compared to Simulation 1, synchronization occurred almost immediately because of the larger radius in this simulation.

Once synchronization had taken place, the frequency of one oscillator was changed to be different from the rest. The original frequency of the system is represented by a dark shade of gray. The higher the frequency, the lighter the shade. The shade is dark if the frequency is around 20. This shade becomes

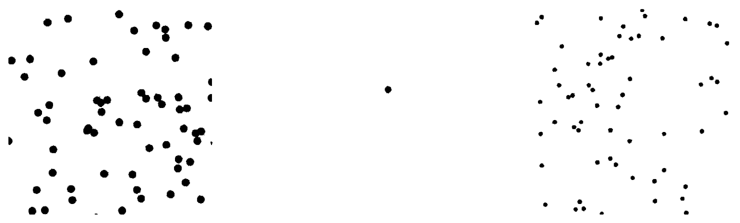
Figure 16: Uncoupled oscillators, with one's frequency increased.

Figure 17: Synchronization.

somewhat lighter if the frequency is increased to 35. The shade becomes slightly lighter again as the frequency approaches 50. For another 30 frames, the initial adaptation conditions were held, without the oscillators being coupled, to show the presence of the one lighter oscillator of different frequency. The result is shown in Figure 16.

As can be seen in these initial conditions, the one lighter oscillator turns "on" and "off" at a higher frequency than the rest, but otherwise the system is in sync and phase together. This initial condition series was kept uncoupled. After these initial conditions had been kept for 30 frames, adaptation was turned on, with the one lighter oscillator representing the leader of the system. Again, every third frame was recorded, and is shown in Figure 17.

The result of this last part of the simulation is that all of the oscillators took on the frequency of the one lighter oscillator. As can be seen in Figure 17, the shade of the oscillators moved between very light and a darkness between that of the oscillators of frequency 20 and 50. The blinking rate of all of these oscillators increased from the previous period, during which they had a frequency of 20. The final result was a system of oscillators, all of a uniform, lighter shade and in phase. The frequency of all of these oscillators had become 50. In this simulation, adaptation took place much more slowly than synchronization at the same radius.

8 Simulation 5

In Simulation 5, there were two different leaders present in the system. One leader was kept at a constant frequency, and the other was not coupled with its neighbors for synchronization. As in the simulations of Sections 4 to 7, the oscillators were not coupled for the first 30 frames. Each oscillator was

Figure 18: Synchronization and Adaptation in the Presence of Two leaders

given a random initial phase and frequency as in Simulation 2 and Simulation 3. After this period, the oscillators were coupled for both synchronization and adaptation.

However, this time two different leaders were present in the system. The first leader did not adapt to the other oscillators in frequency. The second leader did not synchronize to its neighbors in phase. As Figure 18 shows, the oscillators began to synchronize and adapt to a common frequency. One oscillator, though, is very slow to take on the frequency of the others. Because of this slower adaptation, this oscillator had more influence on the others' frequency.

Also, as can be seen in Figure 19, in the final state, the oscillators had a common phase and frequency. Thus the system does, in fact, adapt to the final state of the frequency leader.

Figure 19: Synchronization and Adaptation in the Presence of Two leaders

As can be seen in Figure 20, the the plot of v verses time is similar to that for Simulation 3. Simulation 5 makes it possible to see the frequency of each oscillator and to see how they adapt to the leader's frequency.

Also, as can be seen in Figure 21, the plot of I_i versus time shows how one oscillator is slower in adapting to the rest.

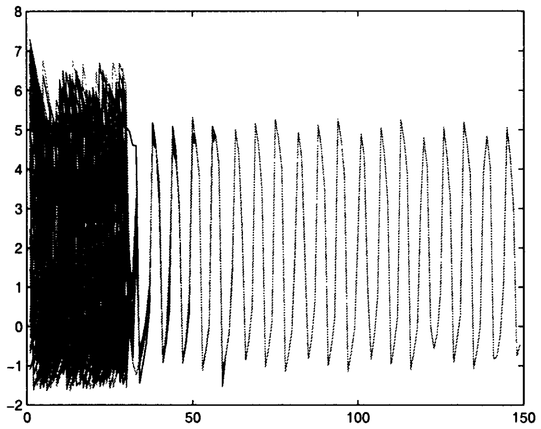
Figure 20: Plot of v_i vs. time for 81 oscillators and 2 leaders.

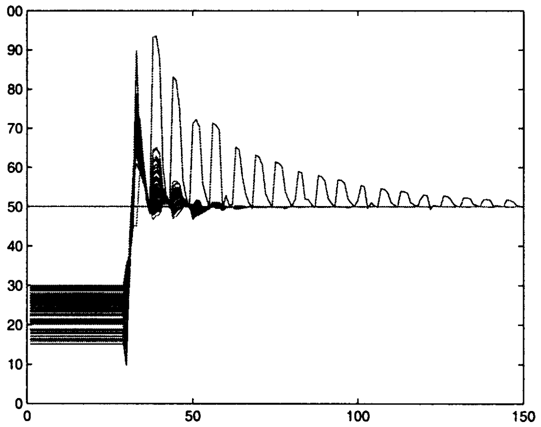
Figure 21: Plot of I_i vs. time for 81 oscillators and 2 leaders.

9 Appendex

Simulation One Code 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global vairables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global row; global column; global init;; global a; global b;
global c; global I; global k; global A_hat_dot;
global P; global xveloc; global yveloc;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global radius; global S; global S2; global I_0; global countsc;
global countlist; global init2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for number of oscillators etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init=130;
%t = [0:1:init];
t = [0, init];
velocmov = .6;
radius = 1.5;
row =9;
column = 9;
init2= 30;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for F-N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 5.23;
b = 3;
c = .1;
k = 15;
I = 20;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpos0 = (10*rand(row,column))-5*ones(row,column);
ypos0 = (10*rand(row,column))-5*ones(row,column);
xveloc =ones(row, column);
xveloc(:,floor(row/2+1):row)=-1;
yveloc = ones(row,column);
yveloc(floor(column/2+1):column,:)=-1;
xveloc = velocmov*xveloc;
yveloc = velocmov*yveloc;
xposlist = zeros((init+S), row*column);
yposlist = zeros((init+S), row*column);
for m=1:1:(init)
for i=1:1:row
for j=1:1:column
        xpos0(i,j)= xpos0(i,j)+ xveloc(i,j);
        ypos0(i,j)= ypos0(i,j)+ yveloc(i,j);
       if xpos0(i,j)>5
            xveloc(i,j) = -xveloc(i,j);
       end
       if xpos0(i,j)<-5
            xveloc(i,j) = -xveloc(i,j);
        end
        if ypos0(i,j)>5
            yveloc(i,j) = -yveloc(i,j);
       end
        if ypos0(i,j)<-5
            yveloc(i,j) = -yveloc(i,j);
        end

       xposlist(m,((i-1)*column+j)) = xpos0(i, j);
      yposlist(m,((i-1)*column+j)) = ypos0(i, j);

    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%ODE and plot
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v = zeros(row, column);
w = zeros(row, column);
v_dot = zeros(row, column);
w_dot = zeros(row, column);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose initial conditions randomly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v_0 = random('Uniform',-2,9, 1,row*column );
%w_0 = random('Uniform',-2,9, 1,row*column );
v_0 = random('Uniform',-40,40, 1,row*column );
w_0 = random('Uniform',-40,40, 1,row*column );

x_dot = zeros(2*row*column,1);
[T, X]=ode23t('Adapt12',t, [v_0 w_0]);

plot(T(:,:),X(:,1:81))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Save Matrix R
%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
n=1;
for m=1:1:(length(X))

     if T(m)>n


for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))= I;


end
end
n=n+1;

end
end

save('rad12.mat', 'R')
```

Simulation One Code 2

```
function dx=Adapt12(t,x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%
global a; global b; global c; global I; global k;
global P; global init2; global x_dot; global A_hat_dot
global row; global column; global init;
global v; global w; global v_dot; global w_dot; ;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global init2;
global radius; global I_0; global countsc; global countlist;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract the initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
        v(i,j) = x( (i-1)*column+j );
        w(i,j) = x( row*column + (i-1)*column+j );
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%synchronization code
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if t>(init2)

for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))-...
     ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
          ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
               ...yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;
```

```
if (rad <= radius)

                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
       end


v_dot(i,j) = v(i,j)*(a-v(i,j))*(v(i,j)-1) - w(i,j) + I+k*vdiff;
w_dot(i,j) = b*v(i,j) - c*w(i,j);

end
    end
end




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial conditions-not coupled
%%%%%%%%%%%%%%%%%%%%%%%%


if t<init2

for i=1:1:row
    for j=1:1:column

v_dot(i,j) = v(i,j)*(a-v(i,j))*(v(i,j)-1) - w(i,j) + I;
w_dot(i,j) = b*v(i,j) - c*w(i,j);
 end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%orgize the returned values
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
         x_dot( (i-1)*column+j ) = v_dot(i,j);
         x_dot( row*column + (i-1)*column+j ) = w_dot(i,j);
     end
 end
```

```
dx=[x_dot];
end
```

Simulation Two Code 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global vairables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global row; global column; global init; global a; global b;
global c; global I; global k; global A_hat_dot; global xveloc;
global yveloc; global countlist; global init2;
global P; global A_hat_0; global init3; global init4;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global radius; global S; global S2; global I_0; global countsc;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for number of oscillators etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init=250;
%t = [0:1:init];
t = [0, init];
velocmov = .6;
radius = 2.7;
row =9;
column = 9;
init2= 30;
init3= 100;
init4=130;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for F-N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 5.23;
a1= 4+3*rand(row*column,1);
b = 3;
b1= 2+2*rand(row*column,1);
c = .1;
c1=.1+.05*rand(row*column,1);
k = 15;
I = 25;
I1=15+20*rand(row*column,1);
P = [0.6 0 0 0 ; 0 30 0 0 ; 0 0 0.002 0 ; 0 0 0 0.4]/10;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpos0 = (10*rand(row,column))-5*ones(row,column);
ypos0 = (10*rand(row,column))-5*ones(row,column);
xveloc =ones(row, column);
xveloc(:,floor(row/2+1):row)=-1;
yveloc = ones(row,column);
yveloc(floor(column/2+1):column,:)=-1;
xveloc = velocmov*xveloc;
yveloc = velocmov*yveloc;
xposlist = zeros((init+S), row*column);
yposlist = zeros((init+S), row*column);
for m=1:1:(init)
for i=1:1:row
for j=1:1:column
        xpos0(i,j)= xpos0(i,j)+ xveloc(i,j);
        ypos0(i,j)= ypos0(i,j)+ yveloc(i,j);
       if xpos0(i,j)>5
            xveloc(i,j) = -xveloc(i,j);
       end
       if xpos0(i,j)<-5
            xveloc(i,j) = -xveloc(i,j);
        end
        if ypos0(i,j)>5
            yveloc(i,j) = -yveloc(i,j);
       end
        if ypos0(i,j)<-5
            yveloc(i,j) = -yveloc(i,j);
        end

       xposlist(m,((i-1)*column+j)) = xpos0(i, j);
      yposlist(m,((i-1)*column+j)) = ypos0(i, j);

    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v = zeros(row, column);
w = zeros(row, column);
```

```
A_hat=zeros(row*column,4);
v_dot = zeros(row, column);
w_dot = zeros(row, column);
A_hat_dot=zeros(4*row*column,1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose initial conditions randomly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v_0 = random('Uniform',-2,9, 1,row*column );
%w_0 = random('Uniform',-2,9, 1,row*column );
v_0 = random('Uniform',-40,40, 1,row*column );
w_0 = random('Uniform',-40,40, 1,row*column );
A_hat_0=zeros(1,4*row*column);

for i=2:1:row*column
A_hat_0(1,(i-1)*4+1)= a1(i,1);
A_hat_0(1,(i-1)*4+2)= I1(i,1);
A_hat_0(1,(i-1)*4+3)= c1(i,1);
A_hat_0(1,(i-1)*4+4)= b1(i,1);
end

A_hat_0(1,1)= a;
A_hat_0(1,2)= I;
A_hat_0(1,3)= c;
A_hat_0(1,4)= b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%run and plot ODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_dot = zeros(6*row*column,1);
[T, X]=ode23tb('Adapt14',t, [v_0 w_0 A_hat_0]);

plot(T(:,:),X(:,1:4))


%%%%%%%%%%%%%%%%%%%%
%Save Matrix R
%%%%%%%%%%%%%%%%%%%
     step=floor(length(X)/init);

n=1;
for m =1:1:length(X)

     if T(m)>n
```

```
for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))=X(m, 2*row*column+4*((i-1)*column+j)-2);

end
end
n=n+1;
end
end

save('rad14.mat', 'R')
```

Simulation Two Code 2

```
function dx=Adapt14(t,x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%
global a; global b; global c; global I; global k; global P; global init2
global row; global column; global init;global A_hat_0;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc;  global A_hat_dot;
global radius; global I_0; global countsc; global countlist; global init2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract the initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
        v(i,j) = x( (i-1)*column+j );
        w(i,j) = x( row*column + (i-1)*column+j );
    end
end

        A_hat = x(2*row*column+1:6*row*column);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%adaptation code
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if t>(init2)

for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))-...
       ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
             ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
                   .../yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;


if (rad <= radius)
```

```
                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
       end



v_dot(1,1) = v(1,1)*(A_hat_0(1,1)-v(1,1))*(v(1,1)-1) - ...
      ...w(1,1) + A_hat_0(1,2);
w_dot(1,1) = A_hat_0(1,4)*v(1,1) - A_hat_0(1,3)*w(1,1);


%end
%    end


if i*j>1


W =  [(v(i,j)*(v(i,j)-1)), 1, 0, 0; 0, 0, -w(i,j), v(i,j)];
A_hat_dot(((i-1)*column+j-1)*4+1:((i-1)*column+j-1)*4+4) = ...
      ...k *P*transpose(W)* [vdiff ; wdiff];
%A_hat_dot= .1*ones(row*column*4,1);
Q = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 5] * 0.01;
A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4)= ...
      ...A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4) + ...
            ...Q * transpose(W) * [vdiff ; wdiff];

end


v_dot(i,j) = v(i,j)*(A_hat(((i-1)*column+j)*4-3,1)-v(i,j))*(v(i,j)-1) ...
      ...- w(i,j) + A_hat(((i-1)*column+j)*4-2,1);
w_dot(i,j) = A_hat(((i-1)*column+j)*4,1)*v(i,j) - ...
      ...A_hat(((i-1)*column+j)*4-1,1)*w(i,j);

end
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial conditions-not coupled
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if t<init2

for i=1:1:row
    for j=1:1:column

v_dot(i,j) = v(i,j)*(A_hat_0(1,((i-1)*column+j)*4-3)-v(i,j))*(v(i,j)-1) ...
      ...- w(i,j) + A_hat_0(1,((i-1)*column+j)*4-2);
w_dot(i,j) = A_hat_0(1,((i-1)*column+j)*4)*v(i,j) - ...
      ...A_hat_0(1,((i-1)*column+j)*4-1)*w(i,j);
 end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%orgize the returned values
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
         x_dot( (i-1)*column+j ) = v_dot(i,j);
         x_dot( row*column + (i-1)*column+j ) = w_dot(i,j);
     end
 end


 x_dot(2*row*column+1: 6*row*column) = A_hat_dot(:,1);

dx=[x_dot];
end
```

Simulation Three Code 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global vairables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global row; global column; global init; global a; global b;
global c; global I; global k; global x_dot;global A_hat_dot;
global P; global A_hat_0; global init3; global init4;
global v; global w; global v_dot; global w_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global xveloc; global yveloc;
global radius; global S; global S2; global I_0; global countsc;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for number of oscillators etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init=150;
%t = [0:1:init];
t = [0, init];
velocmov = .6;
radius = 2.7;
row =9;
column = 9;
init2= 30;
init3= 100;
init4=130;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for F-N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 5.23;
a1= 4+3*rand(row*column,1);
b = 3;
b1= 2+2*rand(row*column,1);
c = .1;
c1=.1+.05*rand(row*column,1);
k = 15;
I = 25;
I1=15+20*rand(row*column,1);
P = [0.6 0 0 0 ; 0 30 0 0 ; 0 0 0.002 0 ; 0 0 0 0.4]/10;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpos0 = (10*rand(row,column))-5*ones(row,column);
ypos0 = (10*rand(row,column))-5*ones(row,column);
xveloc =ones(row, column);
xveloc(:,floor(row/2+1):row)=-1;
yveloc = ones(row,column);
yveloc(floor(column/2+1):column,:)=-1;
xveloc = velocmov*xveloc;
yveloc = velocmov*yveloc;
xposlist = zeros((init+S), row*column);
yposlist = zeros((init+S), row*column);
for m=1:1:(init)
for i=1:1:row
for j=1:1:column
        xpos0(i,j)= xpos0(i,j)+ xveloc(i,j);
        ypos0(i,j)= ypos0(i,j)+ yveloc(i,j);
       if xpos0(i,j)>5
            xveloc(i,j) = -xveloc(i,j);
       end
       if xpos0(i,j)<-5
            xveloc(i,j) = -xveloc(i,j);
        end
        if ypos0(i,j)>5
            yveloc(i,j) = -yveloc(i,j);
       end
        if ypos0(i,j)<-5
            yveloc(i,j) = -yveloc(i,j);
        end

       xposlist(m,((i-1)*column+j)) = xpos0(i, j);
      yposlist(m,((i-1)*column+j)) = ypos0(i, j);

    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v = zeros(row, column);
w = zeros(row, column);
```

```
A_hat=zeros(row*column,4);
v_dot = zeros(row, column);
w_dot = zeros(row, column);
A_hat_dot=zeros(4*row*column,1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose initial conditions randomly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v_0 = random('Uniform',-2,9, 1,row*column );
%w_0 = random('Uniform',-2,9, 1,row*column );
v_0 = random('Uniform',-40,40, 1,row*column );
w_0 = random('Uniform',-40,40, 1,row*column );
A_hat_0=zeros(1,4*row*column);

for i=2:1:row*column
A_hat_0(1,(i-1)*4+1)= a1(i,1);
A_hat_0(1,(i-1)*4+2)= I1(i,1);
A_hat_0(1,(i-1)*4+3)= c1(i,1);
A_hat_0(1,(i-1)*4+4)= b1(i,1);
end

A_hat_0(1,1)= a;
A_hat_0(1,2)= I;
A_hat_0(1,3)= c;
A_hat_0(1,4)= b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%run and plot ODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_dot = zeros(6*row*column,1);
[T, X]=ode23tb('Adapt16',t, [v_0 w_0 A_hat_0]);

plot(T(:,:),X(:,1:4))


%%%%%%%%%%%%%%%%%%%%
%Save Matrix R
%%%%%%%%%%%%%%%%%%%
     step=floor(length(X)/init);

n=1;
for m =1:1:length(X)

     if T(m)>n
```

```
for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))=X(m, 2*row*column+4*((i-1)*column+j)-2);

end
end
n=n+1;
end
end

save('rad16.mat', 'R')
```

Simulation Three Code 2

```
function dx=Adapt16(t,x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%
global a; global b; global c; global I; global k; global P;
global row; global column; global init;global A_hat_0;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global A_hat_dot; global init2
global radius; global I_0; global countsc; global countlist;
global init2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract the initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
        v(i,j) = x( (i-1)*column+j );
        w(i,j) = x( row*column + (i-1)*column+j );
    end
end

        A_hat = x(2*row*column+1:6*row*column);
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%adaptation code
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if t>(init2)

for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))-...
      ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
            ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
                  ...yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;
```

```
if (rad <= radius)

                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
       end



v_dot(1,1) = v(1,1)*(A_hat_0(1,1)-v(1,1))*(v(1,1)-1) - w(1,1) + ...
      ...A_hat_0(1,2) +k*vdiff;
w_dot(1,1) = A_hat_0(1,4)*v(1,1) - A_hat_0(1,3)*w(1,1);


%end
%    end


if i*j>1


W =  [(v(i,j)*(v(i,j)-1)), 1, 0, 0; 0, 0, -w(i,j), v(i,j)];
A_hat_dot(((i-1)*column+j-1)*4+1:((i-1)*column+j-1)*4+4) = ...
      ...k *P*transpose(W)* [vdiff ; wdiff];
%A_hat_dot= .1*ones(row*column*4,1);
Q = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 5] * 0.01;
A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4)= ...
      ...A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4) + ...
            ...Q * transpose(W) * [vdiff ; wdiff];

end


v_dot(i,j) = v(i,j)*(A_hat(((i-1)*column+j)*4-3,1)-v(i,j))*(v(i,j)-1) ...
      ...- w(i,j) + A_hat(((i-1)*column+j)*4-2,1)+k*vdiff;
w_dot(i,j) = A_hat(((i-1)*column+j)*4,1)*v(i,j) - ...
      ...A_hat(((i-1)*column+j)*4-1,1)*w(i,j);

end
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial conditions-not coupled
```

```
%%%%%%%%%%%%%%%%%%%%%%%%


if t<init2

for i=1:1:row
    for j=1:1:column

v_dot(i,j) = v(i,j)*(A_hat_0(1,((i-1)*column+j)*4-3)-v(i,j))*(v(i,j)-1) ...
      ...- w(i,j) + A_hat_0(1,((i-1)*column+j)*4-2);
w_dot(i,j) = A_hat_0(1,((i-1)*column+j)*4)*v(i,j) - ...
      ...A_hat_0(1,((i-1)*column+j)*4-1)*w(i,j);
 end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%orgize the returned values
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
         x_dot( (i-1)*column+j ) = v_dot(i,j);
         x_dot( row*column + (i-1)*column+j ) = w_dot(i,j);
     end
 end


 x_dot(2*row*column+1: 6*row*column) = A_hat_dot(:,1);

dx=[x_dot];
end
```

Simulation Four Code 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global vairables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global row; global column; global init; global a; global b;
global c; global I; global k; global xveloc; global yveloc;
global P; global A_hat_0; global init3; global init4; global I2;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global radius; global S; global S2; global I_0; global countsc;
global countlist; global init2; global jump; global A_hat_dot;
global A_hat; global X;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for number of oscillators etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init=250;
%t = [0:1:init];
t = [0, init];
velocmov = .6;
radius = 2.4;
row =9;
column = 9;
init2= 30;
init3= 90;
init4=120;
jump=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for F-N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 5.23;
a1= 4+3*rand(row*column,1);
b = 3;
b1= 2+2*rand(row*column,1);
c = .1;
c1=.1+.05*rand(row*column,1);
k = 15;
I = 20;
I1=15+15*rand(row*column,1);
I2 = 50;
P = [0.6 0 0 0 ; 0 30 0 0 ; 0 0 0.002 0 ; 0 0 0 0.4]/10;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpos0 = (10*rand(row,column))-5*ones(row,column);
ypos0 = (10*rand(row,column))-5*ones(row,column);
xveloc =ones(row, column);
xveloc(:,floor(row/2+1):row)=-1;
yveloc = ones(row,column);
yveloc(floor(column/2+1):column,:)=-1;
xveloc = velocmov*xveloc;
yveloc = velocmov*yveloc;
xposlist = zeros((init+S), row*column);
yposlist = zeros((init+S), row*column);
for m=1:1:(init)
for i=1:1:row
for j=1:1:column
        xpos0(i,j)= xpos0(i,j)+ xveloc(i,j);
        ypos0(i,j)= ypos0(i,j)+ yveloc(i,j);
       if xpos0(i,j)>5
            xveloc(i,j) = -xveloc(i,j);
       end
       if xpos0(i,j)<-5
            xveloc(i,j) = -xveloc(i,j);
        end
        if ypos0(i,j)>5
            yveloc(i,j) = -yveloc(i,j);
       end
        if ypos0(i,j)<-5
            yveloc(i,j) = -yveloc(i,j);
        end

       xposlist(m,((i-1)*column+j)) = xpos0(i, j);
      yposlist(m,((i-1)*column+j)) = ypos0(i, j);

    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
v = zeros(row, column);
w = zeros(row, column);
A_hat=zeros(row*column*4,1);
v_dot = zeros(row, column);
w_dot = zeros(row, column);
A_hat_dot=zeros(4*row*column,1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose initial conditions randomly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v_0 = random('Uniform',-2,9, 1,row*column );
%w_0 = random('Uniform',-2,9, 1,row*column );
v_0 = random('Uniform',-40,40, 1,row*column );
w_0 = random('Uniform',-40,40, 1,row*column );
A_hat_0=zeros(1,4*row*column);

for i=2:1:row*column
%A_hat_0(1,(i-1)*4+1)= a1(i,1);
A_hat_0(1,(i-1)*4+1)=a;
%A_hat_0(1,(i-1)*4+2)= I1(i,1);
A_hat_0(1,(i-1)*4+2)= I;
%A_hat_0(1,(i-1)*4+3)= c1(i,1);
A_hat_0(1,(i-1)*4+3)= c;
%A_hat_0(1,(i-1)*4+4)= b1(i,1);
A_hat_0(1,(i-1)*4+4)= b;
end

A_hat_0(1,1)= a;
A_hat_0(1,2)= I2;
A_hat_0(1,3)= c;
A_hat_0(1,4)= b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%run and plot ODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_dot = zeros(6*row*column,1);
[T, X]=ode23tb('Adapt15',t, [v_0 w_0 A_hat_0]);

plot(T(:,:),X(:,10:4:24))


     step=floor(length(X)/init);
     n=1;
     for m=1:1:length(X)


     if T(m)>n

     if n<init3
for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))= I;

end
end
n=n+1;
end


if T(m)>init3

for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))= X(m, 2*row*column+4*((i-1)*column+j)-2);

end
end
n=n+1;
end
end
end


save('rad15.mat', 'R')
```

Simulation Four Code 2

```
function dx=Adapt14(t,x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%
global a; global b; global c; global I; global k; global P; global init2
global row; global column; global init;global A_hat_0;
global init3; global init4; global A_hat_dot;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global init2; global X;
global radius; global I_0; global countsc; global countlist;
global I2; global jump; global A_hat;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract the initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
        v(i,j) = x( (i-1)*column+j );
        w(i,j) = x( row*column + (i-1)*column+j );
    end
end

        A_hat = x(2*row*column+1:6*row*column);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%adaptation code
%%%%%%%%%%%%%%%%%%%%%%%%%%
    if t>(init4)

     %A_hat(1,2)=I2;
for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))-...
      ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
            ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
                  ...yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;
```

```
if (rad <= radius)

                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
       end



v_dot(1,1) = v(1,1)*(A_hat_0(1,1)-v(1,1))*(v(1,1)-1) - w(1,1) + A_hat_0(1,2);
w_dot(1,1) = A_hat_0(1,4)*v(1,1) - A_hat_0(1,3)*w(1,1);


if i*j>1

     A_hat(2,1)=I2;
W =  [(v(i,j)*(v(i,j)-1)), 1, 0, 0; 0, 0, -w(i,j), v(i,j)];
A_hat_dot(((i-1)*column+j-1)*4+1:((i-1)*column+j-1)*4+4) = ...
      ...k *P*transpose(W)* [vdiff ; wdiff];
%A_hat_dot= .1*ones(row*column*4,1);
Q = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 5] * 0.01;
A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4)=
      ...A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4) +
            ...Q * transpose(W) * [vdiff ; wdiff];

end

v_dot(i,j) = v(i,j)*(A_hat(((i-1)*column+j)*4-3,1)-v(i,j))*(v(i,j)-1) - ...
      ...w(i,j) + A_hat(((i-1)*column+j)*4-2,1);
w_dot(i,j) = A_hat(((i-1)*column+j)*4,1)*v(i,j) - ...
      ...A_hat(((i-1)*column+j)*4-1,1)*w(i,j);

end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%
%IC's 2
%%%%%%%%%%%%%%%%%%%%%%%%%%
if t>init3
if t<init4

A_hat(2,1)=I2;
X(init3:end,row*column*2+2)=I2;
```

```
for i=1:1:row
    for j=1:1:column

if i*j>1
v_dot(i,j) = v(i,j)*(A_hat_0(1,((i-1)*column+j)*4-3)-v(i,j))*(v(i,j)-1)...
        ... - w(i,j) + A_hat_0(1,((i-1)*column+j)*4-2);
w_dot(i,j) = A_hat_0(1,((i-1)*column+j)*4)*v(i,j) - ...
        ...A_hat_0(1,((i-1)*column+j)*4-1)*w(i,j);
 end
end
end

v_dot(1,1) = v(1,1)*(A_hat_0(1,1)-v(1,1))*(v(1,1)-1) - w(1,1) + I2;
w_dot(1,1) = A_hat_0(1,4)*v(1,1) - A_hat_0(1,3)*w(1,1);

end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Synchronization Code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if t>(init2)
     if t<(init3)

for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))- ...
      ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
            ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
                  ...yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;


if (rad <= radius)

                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
```

```
       end


v_dot(i,j) = v(i,j)*(a-v(i,j))*(v(i,j)-1) - w(i,j) + I+k*vdiff;
w_dot(i,j) = b*v(i,j) - c*w(i,j);

end
    end
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial conditions-not coupled
%%%%%%%%%%%%%%%%%%%%%%%%


if t<init2


for i=1:1:row
    for j=1:1:column

v_dot(i,j) = v(i,j)*(a-v(i,j))*(v(i,j)-1) - w(i,j) + I;
w_dot(i,j) = b*v(i,j) - c*w(i,j);
 end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%orgize the returned values
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
         x_dot( (i-1)*column+j ) = v_dot(i,j);
         x_dot( row*column + (i-1)*column+j ) = w_dot(i,j);
     end
 end


 x_dot(2*row*column+1: 6*row*column) = A_hat_dot(:,1);

dx=[x_dot];
end
```

Simulation Five Code 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global vairables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global row; global column; global init; global a; global b; global c;
global P; global A_hat_0; global init3; global init4;
global v; global w; global v_dot; global w_dot; global x_dot;global A_hat_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global I; global k
global radius; global S; global S2; global I_0; global countsc;
global countlist; global init2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for number of oscillators etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
init=150;
%t = [0:1:init];
t = [0, init];
velocmov = .6;
radius = 2.7;
row =9;
column = 9;
init2= 30;
init3= 100;
init4=130;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%setting variables for F-N
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = 5.23;
a1= 4+3*rand(row*column,1);
b = 3;
b1= 2+2*rand(row*column,1);
c = .1;
c1=.1+.05*rand(row*column,1);
k = 15;
I = 50;
I1=15+15*rand(row*column,1);
P = [0.6 0 0 0 ; 0 30 0 0 ; 0 0 0.002 0 ; 0 0 0 0.4]/10;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%random motion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
xpos0 = (10*rand(row,column))-5*ones(row,column);
ypos0 = (10*rand(row,column))-5*ones(row,column);
xveloc =ones(row, column);
xveloc(:,floor(row/2+1):row)=-1;
yveloc = ones(row,column);
yveloc(floor(column/2+1):column,:)=-1;
xveloc = velocmov*xveloc;
yveloc = velocmov*yveloc;
xposlist = zeros((init+S), row*column);
yposlist = zeros((init+S), row*column);
for m=1:1:(init)
for i=1:1:row
for j=1:1:column
        xpos0(i,j)= xpos0(i,j)+ xveloc(i,j);
        ypos0(i,j)= ypos0(i,j)+ yveloc(i,j);
       if xpos0(i,j)>5
            xveloc(i,j) = -xveloc(i,j);
       end
       if xpos0(i,j)<-5
            xveloc(i,j) = -xveloc(i,j);
        end
        if ypos0(i,j)>5
            yveloc(i,j) = -yveloc(i,j);
       end
        if ypos0(i,j)<-5
            yveloc(i,j) = -yveloc(i,j);
        end

       xposlist(m,((i-1)*column+j)) = xpos0(i, j);
      yposlist(m,((i-1)*column+j)) = ypos0(i, j);

    end
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%set variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v = zeros(row, column);
w = zeros(row, column);
```

```
A_hat=zeros(row*column,4);
v_dot = zeros(row, column);
w_dot = zeros(row, column);
A_hat_dot=zeros(4*row*column,1);


%%%%%%%%%%%%%%%%%%%%%%%%%%%
%choose initial conditions randomly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%v_0 = random('Uniform',-2,9, 1,row*column );
%w_0 = random('Uniform',-2,9, 1,row*column );
v_0 = random('Uniform',-40,40, 1,row*column );
w_0 = random('Uniform',-40,40, 1,row*column );
A_hat_0=zeros(1,4*row*column);

for i=2:1:row*column
A_hat_0(1,(i-1)*4+1)= a1(i,1);
A_hat_0(1,(i-1)*4+2)= I1(i,1);
A_hat_0(1,(i-1)*4+3)= c1(i,1);
A_hat_0(1,(i-1)*4+4)= b1(i,1);
end

A_hat_0(1,1)= a;
A_hat_0(1,2)= I;
A_hat_0(1,3)= c;
A_hat_0(1,4)= b;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%run and plot ODE
%%%%%%%%%%%%%%%%%%%%%%%%%%%

x_dot = zeros(6*row*column,1);
[T, X]=ode23tb('Adapt18',t, [v_0 w_0 A_hat_0]);

plot(T(:,:),X(:,1:4))


%%%%%%%%%%%%%%%%%%%%
%Save Matrix R
%%%%%%%%%%%%%%%%%%%
     step=floor(length(X)/init);

n=1;
for m =1:1:length(X)

     if T(m)>n
```

```
for i=1:1:row
    for j=1:1:column

R(n, (((i-1)*column+j)*4)-3)=X(m,((i-1)*column+j));

R(n, (((i-1)*column+j)*4-2)) = xposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4-1)) = yposlist(n, ((i-1)*column+j));

R(n, (((i-1)*column+j)*4))=X(m, 2*row*column+4*((i-1)*column+j)-2);

end
end
n=n+1;
end
end

save('rad18.mat', 'R')

\begin{verbatim}

\newpage
\begin{flushleft}
{\bf Simulation Five Code 2}
\end{flushleft}
\begin{verbatim}
function dx=Adapt16(t,x)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%global variables
%%%%%%%%%%%%%%%%%%%%%%%%%%
global a; global b; global c; global I; global k; global P; global init2
global row; global column; global init;global A_hat_0;
global v; global w; global v_dot; global w_dot; global x_dot;
global xpos0; global ypos0; global xposlist; global yposlist;
global xveloc; global yveloc; global A_hat_dot;
global radius; global I_0; global countsc; global countlist; global init2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%extract the initial conditions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
```

```
        v(i,j) = x( (i-1)*column+j );
        w(i,j) = x( row*column + (i-1)*column+j );
    end
end

        A_hat = x(2*row*column+1:6*row*column);
%%%%%%%%%%%%%%%%%%%%%%%%%%
%adaptation code
%%%%%%%%%%%%%%%%%%%%%%%%%
    if t>(init2)

for i=1:1:row
    for j=1:1:column

        vdiff=0;
        wdiff=0;
        for l=1:1:row
            for m = 1:1:column
rad = ((xposlist((ceil(t)),((i-1)*column+j))-
      ...xposlist((ceil(t)),((l-1)*column+m)))^2 + ...
            ...(yposlist((ceil(t)),((i-1)*column+j)) - ...
                  ...yposlist((ceil(t)),((l-1)*column+m)))^2)^.5;


if (rad <= radius)

                    vdiff = vdiff + v(l,m)- v(i,j);
                    wdiff = wdiff + w(l,m)- w(i,j);
              end
            end
       end



v_dot(1,1) = v(1,1)*(A_hat_0(1,1)-v(1,1))*(v(1,1)-1) - ...
      ...w(1,1) + A_hat_0(1,2)+ k*vdiff;
w_dot(1,1) = A_hat_0(1,4)*v(1,1) - A_hat_0(1,3)*w(1,1);



%end
%    end


if i*j>1
```

```
W =  [(v(i,j)*(v(i,j)-1)), 1, 0, 0; 0, 0, -w(i,j), v(i,j)];
A_hat_dot(((i-1)*column+j-1)*4+1:((i-1)*column+j-1)*4+4) = ...
      ...k *P*transpose(W)* [vdiff ; wdiff];
%A_hat_dot= .1*ones(row*column*4,1);
Q = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 5] * 0.01;
A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4)= ...
      ...A_hat(((i-1)*column+j-1)*4+1 : ((i-1)*column+j-1)*4+4) + ...
            ...Q * transpose(W) * [vdiff ; wdiff];

end

v_dot(i,j) = v(i,j)*(A_hat(((i-1)*column+j)*4-3,1)-v(i,j))*(v(i,j)-1) - ...
      ...w(i,j) + A_hat(((i-1)*column+j)*4-2,1)+k*vdiff;
w_dot(i,j) = A_hat(((i-1)*column+j)*4,1)*v(i,j) - ...
      ...A_hat(((i-1)*column+j)*4-1,1)*w(i,j);



end
end
end

v_dot(1,2) = v(1,2)*(A_hat(5,1)-v(1,2))*(v(1,2)-1) - w(1,2) + A_hat(6,1);
w_dot(1,2) = A_hat(8,1)*v(1,2) - A_hat(7,1)*w(1,2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%initial conditions-not coupled
%%%%%%%%%%%%%%%%%%%%%%%%


if t<init2

for i=1:1:row
    for j=1:1:column

v_dot(i,j) = v(i,j)*(A_hat_0(1,((i-1)*column+j)*4-3)-v(i,j))*(v(i,j)-1) ...
      ... - w(i,j) + A_hat_0(1,((i-1)*column+j)*4-2);
w_dot(i,j) = A_hat_0(1,((i-1)*column+j)*4)*v(i,j) - ...
      ... A_hat_0(1,((i-1)*column+j)*4-1)*w(i,j);
 end
end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%orgize the returned values
%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:1:row
    for j=1:1:column
         x_dot( (i-1)*column+j ) = v_dot(i,j);
         x_dot( row*column + (i-1)*column+j ) = w_dot(i,j);
     end
 end


 x_dot(2*row*column+1: 6*row*column) = A_hat_dot(:,1);

dx=[x_dot];
end
```

Graphics Code

```
#include "mex.h"
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <unistd.h>
#include "dumpImages.h"

GLenum rgb, doubleBuffer, windType;
GLint windW, windH;

/*
#include "tkmap.c"
*/

GLenum mode;
GLint size;
float point[3] = {1.0, 1.0, 0.0};
#define PI 3.1415926535898
#define NUM_PARAMS (0)
#define IMG_NUMROWS 600
#define IMG_NUMCOLS 600
#define BYTES_PER_PIXEL 3
#define ASCII_0 48
#define numparam   4
#define numrow 9
#define numcol 9
#define numscenes 249


int mrows=0;
int ncols=0;
int count=0;
int count2=0;
double xone[numrow*numcol*numparam*numscenes]={0};
int xonecount = 0;
int loop1=0;
int loop2=0;
int rtri=0;
int scene=0;
float cosine, sine;

/*
static void Init(void)
```

```
{
    GLint i;

    glClearColor(0.0, 0.0, 0.0, 0.0);

    glBlendFunc(GL_SRC_ALPHA, GL_ZERO);

    if (!rgb) {
for (i = 0; i < 16; i++) {
  //glutSetColor(i+CI_ANTI_ALIAS_RED, i/15.0, 0.0, 0.0);
    //   glutSetColor(i+CI_ANTI_ALIAS_YELLOW, i/15.0, i/15.0, 0.0);
    //  glutSetColor(i+CI_ANTI_ALIAS_GREEN, 0.0, i/15.0, 0.0);
     }
   }

    // mode = GL_FALSXE;
       size = 1;
    }
*/

void InitGL(int Width, int Height)
// We call this right after our OpenGL window is created.

{
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
  // This Will Clear The Background Color To Black
  glClearDepth(1.0);
  // Enables Clearing Of The Depth Buffer
  glDepthFunc(GL_LESS);
  // The Type Of Depth Test To Do
  glEnable(GL_DEPTH_TEST);
  // Enables Depth Testing
  glShadeModel(GL_SMOOTH);
  // Enables Smooth Color Shading

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  // Reset The Projection Matrix

  gluPerspective(90.0f,(GLfloat)Width/(GLfloat)Height,0.1f,120.0f);
  // Calculate The Aspect Ratio Of The Window

  glMatrixMode(GL_MODELVIEW);
}
static void Reshape(int width, int height)
{
    windW = (GLint)width;
    windH = (GLint)height;

    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    /*gluOrtho2D(-windW/2, windW/2, -windH/2, windH/2);*/
 gluPerspective(90.0f,(GLfloat)width/(GLfloat)height,0.1f,120.0f);
    glMatrixMode(GL_MODELVIEW);
}

static void Key2(int key, int x, int y)
{

    switch (key) {

}

}


static void Key(unsigned char key, int x, int y)
{
usleep(100);

    switch (key) {
      case 27:
//{glutDestroyWindow(scene);

exit(1);
case '1':
mode = !mode;
break;

     //break;
      default:
return;

    }

    glutPostRedisplay();
}

static void Draw()
```

```
 {
   double yone1[1][1]={0};
   double yone2[1][1]={0};
   double yone3[1][1]={0};
   double yone[numrow*numcol*numparam][numscenes]= {0};
   double colorg[1][1] = {0};
   double colorr[1][1] = {1};
   double colorb[1][1] = {0};
   double sub1[1][1] = {0};
   double sub2[1][1] = {0};
double sub3[1][1] = {0};
   int tot= numparam*numrow*numcol;
   int ycountone=0;
   int test;
   int tests;
   int numpt = 0;

     printf(" %f \n", xone[scene+numscenes*9]);
     printf(" %f \n", xone[scene+numscenes*5]);




  GLint i=0;
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       for (loop1=0; loop1<numrow; loop1++)
 {
     for (loop2=0; loop2<numcol; loop2++)
    {

 //To get into black and white:
      //    glBegin(GL_POLYGON);
      //     glColor3f(1.0f, 1.0f, 1.0f);
      // glVertex3f(12, -12, -6.5f);
      // glVertex3f(-12, -12, -6.5f);
      // glVertex3f(-12, 12, -6.5f);
      // glVertex3f(12, 12, -6.5f);

      //      glEnd();

      glLoadIdentity();
      glTranslatef(xone[(scene+(numscenes)*(numpt+1))],   ...
            ...xone[(scene+(numscenes)*(numpt+2))], -5.0);


      sub3[0][0]= xone[scene+(numscenes)*(numpt)];
      if (xone[(scene + (numscenes)*(numpt+3))]>57) {
     sub1[0][0] = (xone[(scene+(numscenes)*(numpt+3))]-52)/10;
     sub2[0][0] = (1-sub1[0][0]);
     //if (xone[(scene+(numscenes+1)*(numpt))]>0) {
   glColor3f(sub1[0][0], sub2[0][0], 0.0f);
     }

      if (xone[(scene + (numscenes)*(numpt+3))]<57) {
      if (xone[(scene + (numscenes)*(numpt+3))]>43) {
     sub1[0][0] = (xone[(scene+(numscenes)*(numpt+3))]-17)/1;
     sub2[0][0] = (1-sub1[0][0]);
     //if (xone[(scene+(numscenes+1)*(numpt))]>0) {
   glColor3f(0.0f, sub1[0][0], sub2[0][0]);
      }
     }


             if (xone[scene+(numscenes)*(numpt+3)]<43) {
   sub1[0][0]= (xone[(scene+(numscenes)*(numpt+3))] -38)/15;
   sub2[0][0]= (1-sub1[0][0]);
            glColor3f(0.0f, sub1[0][0], sub2[0][0]);
            }

      if (xone[(scene+(numscenes)*(numpt+3))] != 0) {
      if (xone[(scene+(numscenes)*(numpt+2))] != 0)
{

      xonecount+=1;
      //ycountone+=(numscenes)*3;
      numpt+=4;
      if (sub3[0][0]>1) {
    glBegin(GL_POLYGON);
      for (i=0; i<100; i++) {
cosine=cos(i * 2 * PI / 100.0)*(sqrt(sub3[0][0])) /13;
sine=sin(i * 2 * PI / 100.0)*(sqrt(sub3[0][0])) /13;
glVertex3f(cosine, sine, 0.0f);
      }
      glEnd();
      }
     }


  }
  }
  }
unsigned char img[IMG_NUMROWS * IMG_NUMCOLS*BYTES_PER_PIXEL];
FILE *fptr;
       glReadPixels(0, 0, IMG_NUMCOLS, IMG_NUMROWS, GL_RGB, ...
            ... GL_UNSIGNED_BYTE, img);

   char fname[17] = "fig6/fig000.jpg";

   if (scene<10){
   fname[10]=(char)(scene+ASCII_0);
   }
   else {
     if (scene < 100){
       fname[9]=(char)((int)(scene/10)+ASCII_0);
       fname[10]=(char)((int)(scene%10)+ASCII_0);
     }
     else{
       fname[8]=(char)((int)(scene/100)+ASCII_0);
       fname[9]=(char)((int)(scene/10)%10+ASCII_0);
       fname[10]=(char)((int)(scene%10)+ASCII_0);
     }}
   dump_RGB_img(fname, IMG_NUMCOLS, IMG_NUMROWS, img);


       scene++;
glFlush();

if (doubleBuffer) {

   glutSwapBuffers();
   }
  usleep(200000);
  }

static GLenum Args(int argc, char **argv)
  {
   GLint i;

   rgb = GL_TRUE;
   doubleBuffer = GL_FALSE;

     for (i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-ci") == 0) {

          } else {
    printf("%s (Bad option).\n", argv[i]);
```

```
           return GL_FALSE;
    }
}
   return GL_TRUE;
   }

void timestwo(double * x)
{
  int j = 0;
  int i = 0;
  int count3 = 0;

  //printf("x is &f \n", x);
  for (i=0; i<(numrow*numcol*numparam*numscenes); i++)
{
   xone[i]=*((double *)(x+count3));
   count3+=1;
    }


   int myArgc = 1;
   char** myArgv;
   char* myString = {"hello3"};
   myArgv = &myString;

  glutInit(&myArgc, myArgv);

  if (Args(myArgc, myArgv) == GL_FALSE) {

  }
  windW = 640;
  windH = 640;
  glutInitWindowPosition(200, 0); glutInitWindowSize( windW, windH);

  windType = (rgb) ? GLUT_RGB : GLUT_INDEX;
  windType |= (doubleBuffer) ? GLUT_DOUBLE : GLUT_SINGLE;
  glutInitDisplayMode(windType);

    if (glutCreateWindow("hello3") == GL_FALSE) {

    }



    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Key);
    glutDisplayFunc(Draw);
  glutIdleFunc(Draw);
  InitGL(windW, windH);

  glutMainLoop();
}


void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[])
{
  double *x;
  int status, mrows, ncols;

  x = mxGetPr(prhs[0]);
  mrows = mxGetM(prhs[0]);
  ncols = mxGetN(prhs[0]);
    timestwo(x);
}
```

References

[1] Pikovsky, A., Rosenblum, M., and Kurths, J., "Synchronization: A Universal Concept in Nonlinear Sciences", Cambridge University Press, 2003

[2] Sherman, H. "Biological Engineering," McGraw-Hill, 1969.

[3] Wang, W., and Slotine, J. J. E., "Where To Go and How To Go: A Theoretical Study of Different Leader Roles in Networked Systems," 2004.