```
clear all;

%Part A- This will make a model simmilar to figure 1, it uses the same
%numbers with the following values:

%A  -a                                          vel (0.003 < v< 0.3)
    %L= 7, r= 1, dT=1, tI= 500, n=2, nP=300, vel= .03
    za(  7,    1,    1,   500,   2,    300,      .03)

   %-b lower noise, greater L
    %L= 25, r= 1, dT=1, tI= 500, n=.1, nP=300, vel= .03
    za(  25,    1,    1,     500,   .1,    300,      .03)

   %-c higher time
    %L= 7, r= 1, dT=1, tI= 500, n=2, nP=300, vel= .03
   za(  7,    1,    1,     750,  2,    300,      .03)

   %-d L=5 n=.1
    %L= 5, r= 1, dT=1, tI= 500, n=.1, nP=300, vel= .03
    za(  5,    1,    1,     500,  .1,    300,      .03)

%%


%Part B- This will make a model simmilar to figure 2-
clear all;

for repool= 1:2:5 %diff particleNum
     parN= (repool^2)*(100/repool);

count=0;
yAx=0;
for looper= 0:.1:5  %diff noise
    count=count+1
    yAx(count)= zb(  7,    1,    1,     500,  looper,    parN,      .03);

end
plot(0:.1:5,(yAx))
hold on;
end




%Vicsek Model-Function for Figure 1
function f= za(L,r,dT,tI,n,nP,v)
```

```
%Inital Factors
LinDom= L; % linear domain size L as described in paper (LxL)
IntRad=r; % interaction radius
deltaTime=dT;% Delta-time unit, established in paper
TimeIntervals=tI; %the steps of time for the simulation
noise= n; % the noise level
NumPart=nP; % the number of particles
vel= v; % the absolute velocity

%Establishing X,Y, & Theta
x_comp=LinDom*rand(1,NumPart); %establishes the x coords for N amount of particles
y_comp=LinDom*rand(1,NumPart) ;%establishes the y coords for N amount of particles
theta=2*pi*(rand(1,NumPart)-0.5); % randomly distributed angles for N particles


for time= 1:TimeIntervals %loop through the TimeInterval

    % Calculation of average angle in the interacting circle-
    % the function(pdist() measures the distances between all particles(LARGE NUM)
    % Then, it determines if distance< r
    % Essientally it enables the arrows to follow their nearby neighbors
    D = pdist([x_comp' y_comp']);

        %setting up X boundary- basically establishing that it will go to
        %the other side of screen if it goes past the edge, I find it
        %easy to think of it as a really small globe
        bound_x(x_comp<IntRad) = x_comp(x_comp<IntRad) + LinDom ;
        bound_x(x_comp>LinDom-IntRad) = x_comp(x_comp>LinDom-IntRad)-LinDom;
        bound_x(IntRad<=x_comp & x_comp<=LinDom-IntRad) = x_comp(IntRad<=x_comp &
x_comp<=LinDom-IntRad);

        %setting up Y boundary- same thing just in the y axis
        bound_y(y_comp<IntRad) = LinDom + y_comp(y_comp<IntRad);
        bound_y(y_comp>LinDom-IntRad) = y_comp(y_comp>LinDom-IntRad)-LinDom;
        bound_y(IntRad<=y_comp & y_comp<=LinDom-IntRad) = y_comp(IntRad<=y_comp &
y_comp<=LinDom-IntRad);


        %In order to establish the shortest distance, it needs to work in
        %such a way because a particle all the way on the right side is
        %also very close to a particle all the way to the left, like a
        %globe, making this distinction is clear to the motion of the
        %particles
        OneTime_D = pdist([bound_x' bound_y']);
        D = min([D; OneTime_D]);
        %Matrix M, representation for small distance between particles, making
        %sure that the pairs of particles are within the range (the radius)
        Mat_dist = squareform(D); %①Establish Matrix
        [l1,l2]=find(0<Mat_dist & Mat_dist<IntRad); %each pair of particles


    %In Paper, avg direction given by arcTan( sinTheta_r, cosTheta_r)
    % r is radius surrounding (behavior of neighbors)
    for i = 1:NumPart
        list = l1(l2==i);
        if ~isempty(list)
            ave_theta(i) = atan2(mean(sin(theta(list))),mean(cos(theta(list))));
        else
            ave_theta(i) = theta(i); %avg angle from nearby neighbors within the
```

```
radius, so that it can decide its own direction
        end
    end %ending small 4loop

    % Updating
    x_vel= vel*cos(theta); %establishes the averge x comp of Vel
    y_vel= vel*sin(theta); %establishes the averge y comp of Vel
    x_comp = x_comp + x_vel ; %adds the vel to x to get next x pos
    y_comp = y_comp + y_vel ; %adds the vel to y to get next y pos


    %Going back to the bounds idea, this keep the particles within the
    %viewable screen, if they go off the page in either direction the
    %LinDom will just be added/subtracted to reposition the particle
        x_comp(x_comp<0) = LinDom + x_comp(x_comp<0);
        x_comp(LinDom<x_comp) = x_comp(LinDom<x_comp) - LinDom;
        y_comp(y_comp<0) = LinDom + y_comp(y_comp<0);
        y_comp(LinDom<y_comp) = y_comp(LinDom<y_comp) - LinDom;

    theta = ave_theta + noise * (rand(1,NumPart) - 0.5); %Adds some element
    % of noise to the average theta obtained from its neighbors

    % Plotting the particles as quiver so that they are arrows with
    % position and direction-- for quiver (xpos,ypos,xdirec,ydirec,size)
    figure (1)
    quiver(x_comp,y_comp,x_vel,y_vel,.2) %plots arrow, position, direction, small
size
    xlim([0 LinDom]); % Keeping in Bounds
    ylim([0 LinDom]); % Keeping in Bounds
    axis square
    pause(0.15) %there must be a pause, the pause is so that the graph
    % appears to be changing over time depecting the motion, like frames
    % on a video, otherwise just the end result would be shown.
end %ending big 4loop



end %ending func







%Vicsek Model-Function for Figure 2
function f= zb(L,r,dT,tI,n,nP,v)


%Inital Factors
LinDom= L; % linear domain size L as described in paper (LxL)
IntRad=r; % interaction radius
deltaTime=dT;% Delta-time unit, established in paper
TimeIntervals=tI; %the steps of time for the simulation
noise= n; % the noise level
NumPart=nP; % the number of particles
vel= v; % the absolute velocity

%Establishing X,Y, & Theta
```

```
x_comp=LinDom*rand(1,NumPart); %establishes the x coords for N amount of particles
y_comp=LinDom*rand(1,NumPart) ;%establishes the y coords for N amount of particles
theta=2*pi*(rand(1,NumPart)-0.5); % randomly distributed angles for N particles

b=0; %counter
listOfTheta= 1:TimeIntervals;
distTheta= 1:TimeIntervals;
for time= 1:TimeIntervals %loop through the TimeInterval
    b=b+1; %counter +1
    % Calculation of average angle in the interacting circle-
    % the function(pdist() measures the distances between all particles(LARGE NUM)
    % Then, it determines if distance< r
    % Essientally it enables the arrows to follow their nearby neighbors
    D = pdist([x_comp' y_comp']);

        %setting up X boundary- basically establishing that it will go to
        %the other side of screen if it goes past the edge, I find it
        %easy to think of it as a really small globe
        bound_x(x_comp<IntRad) = x_comp(x_comp<IntRad) + LinDom ;
        bound_x(x_comp>LinDom-IntRad) = x_comp(x_comp>LinDom-IntRad)-LinDom;
        bound_x(IntRad<=x_comp & x_comp<=LinDom-IntRad) = x_comp(IntRad<=x_comp &
x_comp<=LinDom-IntRad);

        %setting up Y boundary- same thing just in the y axis
        bound_y(y_comp<IntRad) = LinDom + y_comp(y_comp<IntRad);
        bound_y(y_comp>LinDom-IntRad) = y_comp(y_comp>LinDom-IntRad)-LinDom;
        bound_y(IntRad<=y_comp & y_comp<=LinDom-IntRad) = y_comp(IntRad<=y_comp &
y_comp<=LinDom-IntRad);


        %In order to establish the shortest distance, it needs to work in
        %such a way because a particle all the way on the right side is
        %also very close to a particle all the way to the left, like a
        %globe, making this distinction is clear to the motion of the
        %particles
        OneTime_D = pdist([bound_x' bound_y']);
        D = min([D; OneTime_D]);
        %Matrix M, representation for small distance between particles, making
        %sure that the pairs of particles are within the range (the radius)
        Mat_dist = squareform(D); %①Establish Matrix
        [l1,l2]=find(0<Mat_dist & Mat_dist<IntRad); %each pair of particles


    %In Paper, avg direction given by arcTan( sinTheta_r, cosTheta_r)
    % r is radius surrounding (behavior of neighbors)
    for i = 1:NumPart
        list = l1(l2==i);
        if ~isempty(list)
            ave_theta(i) = atan2(mean(sin(theta(list))),mean(cos(theta(list))));
        else
            ave_theta(i) = theta(i); %avg angle from nearby neighbors within the
radius, so that it can decide its own direction
        end
    end %ending small 4loop

    % Updating
    x_vel= vel*cos(theta); %establishes the averge x comp of Vel
    y_vel= vel*sin(theta); %establishes the averge y comp of Vel
    x_comp = x_comp + x_vel ; %adds the vel to x to get next x pos
```

```
    y_comp = y_comp + y_vel ; %adds the vel to y to get next y pos


    %Going back to the bounds idea, this keep the particles within the
    %viewable screen, if they go off the page in either direction the
    %LinDom will just be added/subtracted to reposition the particle
        x_comp(x_comp<0) = LinDom + x_comp(x_comp<0);
        x_comp(LinDom<x_comp) = x_comp(LinDom<x_comp) - LinDom;
        y_comp(y_comp<0) = LinDom + y_comp(y_comp<0);
        y_comp(LinDom<y_comp) = y_comp(LinDom<y_comp) - LinDom;

     noisey=noise * (rand(1,NumPart) - 0.5);
     theta = ave_theta + noisey; %Adds some element






    listOfTheta(b)= abs(mean((mean(ave_theta)-mean(theta)-mean(noisey))));



end %ending big 4loop

OverallTheta= mean(listOfTheta);
distTheta = (((OverallTheta)/pi)*-1+1);
f= distTheta;
end %ending func
```