Homework 4.(b)

Modeling Complex Systems, Javier Lobato

Due date: Tuesday, April 17, 2018

1 Problem overview and motivation

After having designed a simple swarm model that implemented the boids with relations of proximity (circular zone of repulsion) and angle of vision (field of view), some questions were still unanswered. Reading the discussion paper about which were the different types of relationships between the birds and which ones were the more realistic, the next step is clearly to code some of those relations and see how the model behaves under a range parameter values. The objective is to see how the flock cohesion is maintained or lost when varying the different variables.

Modeling nature with accuracy is important in order to predict phenomena and to be able to study the processes that take place in nature. Obtaining models that are close representations of the reality is essential before asking more detailed questions to a system. Thus, testing a code under a wide range of parameters and knowing how will it behave is a good point to start.

2 Experimental design

Beginning with the MATLAB code of the previous assignment, some modifications were done to implement the different types of relations between birds. A vector called `visionMode` consisting on three elements was used. Each component turns on and off one type or relation between birds - being possible to combine some of them:

$$\texttt{visionMode} = [\texttt{zone of repulsion} \quad \texttt{field of view} \quad \texttt{k-nearest}]$$

There were some combinations that were not feasible, given that mixing a constant circle with the k closest neighbors is excluyent. Thus, the 8 possible combinations were:

- (`0 0 0`) : a case with no possible combinations makes no sense
- (`1 0 0`) : just the zone of repulsion
- (`0 1 0`) : field of view that extends to the end of the grid
- (`0 0 1`) : k-nearest neighbors
- (`1 1 0`) : mixing zone of repulsion with field of view, as the previous assignment
- (`1 0 1`) : this case was not analyzed given that mixing the radius of the zone of repulsion and the k-nearest individuals are restricting and the one that has the greatest value will dominate the other selection method
- (`0 1 1`) : this is the case more interesting and, at least for me, more realistic. Birds look in their field of view but instead trying to analyze all birds (`0 1 0`) or restricting themselves to certain metric distance (`1 1 0`), a topological distance is taken. Thus, the bird will analyze only the k closest birds in its field of view
- (`1 1 1`) : given that (`1 0 1`) was not implemented, this one was neither considered because it has the same restrictions

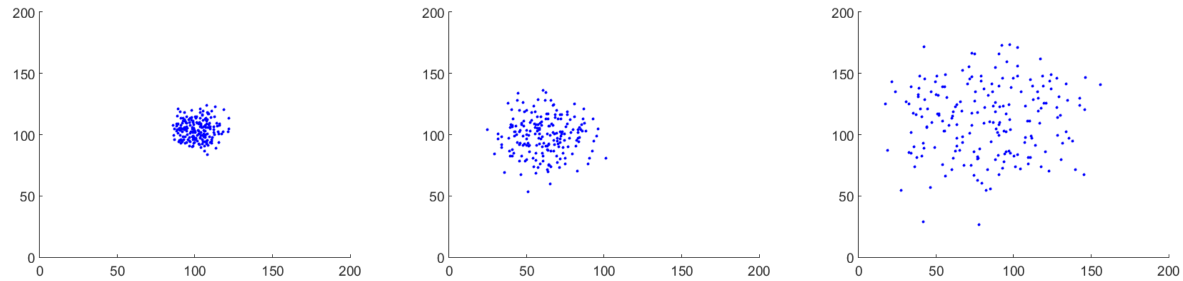
The model takes most part of the past homework, but some functions were modified. The modifications done to the next functions are a little explained below, whereas the functions without anything written next to haven't been modified (all code is listed at the end):

- `boundary.m`: the only modification is the increase of speed when a boid bounces on the limits
- `boundposition.m`
- `dataAnalysis.m`: a script to run the test performed and the experiment setup
- `HW4b_driver.m`: the whole driver was redesigned
- `initialization.m`
- `limitvelocity.m`
- `maxDist.m`: computes the maximum distance between a list of points in 2D
- `pltdistribution.m`: plot the current boid with the desired options
- `rule1.m`
- `rule2.m`: computes rule 2 with the different possible combinations of boid perception
- `rule3.m`
- `rule4.m`
- `swarmModel.m`: includes all the changes related to function calling and vector manipulation
- `updateboid.m`: changes of function calling and data manipulation
- `updatepred.m`

3 Quantification of the results

The metric used to know how disperse the flock was consisted of the maximum distance between boids. In order to increase the performance of c theode and reduce computation time, a more efficient approach was carried out. The naive idea of comparing each boid to the others in the flock, taking the maximum of those distances and repeating that process for all boids, having at the end the maximum of the maximums as the metric of the size is computationally slow. A less precise but quicker method was implemented. The idea (taken from `https://stackoverflow.com/a/8006849`) begins by computing the smaller and biggest possible $(x + y)$ (where x and y are the position of thse boid) and the smallest and biggest $(x-y)$. Those values are afterwards combined in cross: the distance between the lower left element (minimum $(x+y)$) and the upper right element (maximum $(x+y)$) is computed ($d1$) and the distance between the upper left element (minimum $(x-y)$) and the lower right element (maximum $(x-y)$) that gives $d2$. The maximum between $d1$ and $d2$ will be the value of the metric for a given flock at certain timestep. This metric will be computed and analyzed for each timestep, having its evolution in time.

In the images below it can be seen three example flocks with the value returned from this function. These are randomly chosen timesteps of a simulation with different flock densities:

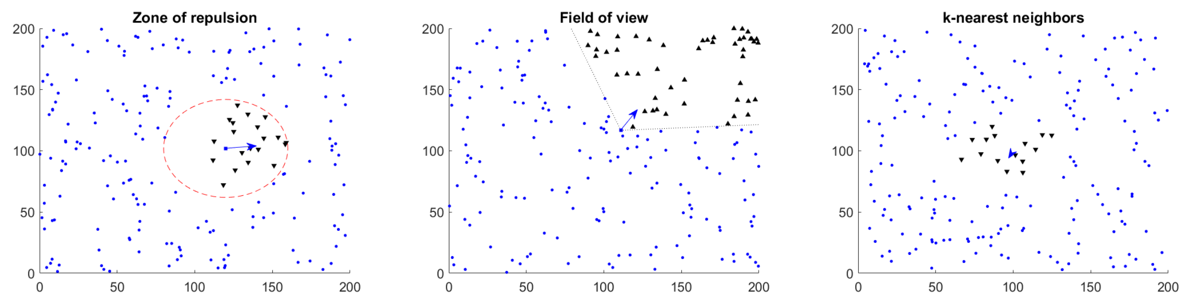


maxDist = `39.9860` maxDist = `77.8312` maxDist = `159.8045`

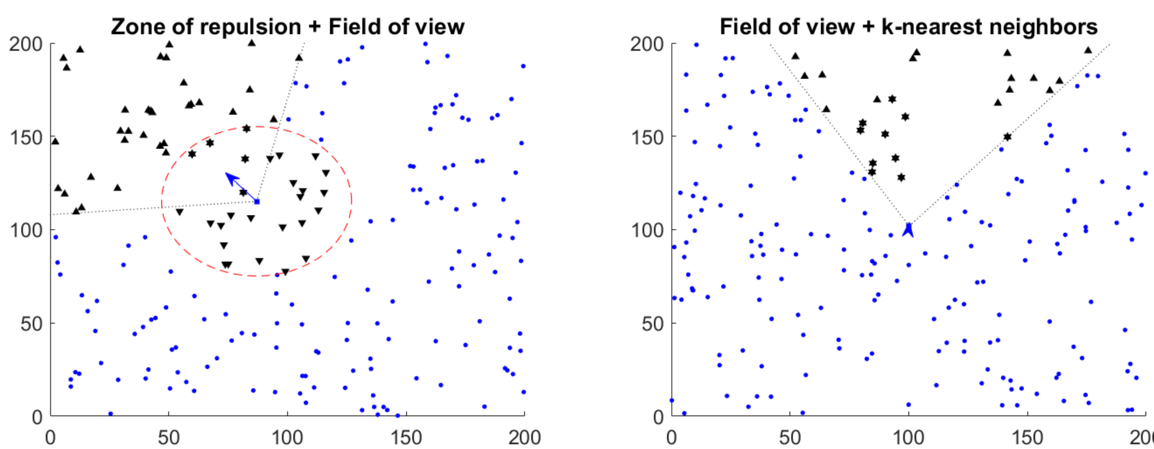
Figure 1: Metric used to measure the cohesion of the flock

The maximum distance was afterward normalized by the maximum possible flock distance. This case will happen if one bird is located at $(0, 0)$ and the other is located at $(\texttt{dim}, \texttt{dim})$. This distance will be $\sqrt{2\texttt{dim}^2}$, so the values are normalized by that distance in the discussion figures.

Also, the different cases of the perception of the boids were run to test if they were working correctly:



Zone of repulsion (▼) Field of view (▲) k-nearest (▼)



Zone of repulsion (▼) and field of view (▲)
Boids in `zor` and `fov` (✸)

k-nearest (▼) and field of view (▲)
Boids in `kn` and `fov` (✸)

Figure 3: Different perception of the boids configuration

4 Results

After having tried with the five cases, the results seen may be outlined as:

- Zone of repulsion alone type give nice results
- Field of view type is not stable by itself: given that all birds in the direction of flight are analyzed, boids will oscillate because they will turn around very quickly without barely moving
- k-nearest used in its own resulted in cohesion and good flock behavior
- Zone of repulsion and field of view combined gave the results discussed in the previous assignment
- Field of view with k-nearest yield the most interesting behavior and is the one that will be analyzed

The different configurations tested may be seen in the next table:

PARAMETER	**VALUE**
Grid size	200×200
Individuals	50
Timesteps	100
Predator	No
Velocity coefficients	(1/100, 1, 1/8, 1)
Vision modes tested	[k-nearest, k-nearest & field of vision]
Same simulation runs	10
k-nearest	[1,5,10,15,20]
Field of vision angle	[0.2π,0.4π,0.6π,0.8π,1.0π,1.2π]
Zone of repulsion radio	[1,2,3,4,5]

After running 10 times the same setup, the maximum distance of the flock was analyzed. Computing the mean and the standard deviation of the maximum distance the behavior of the system may be analyzed. This setup may be summarized in the three next simulations:

- Parameter sweep of the number of k nearest boids with the k-nearest & field of view mode:

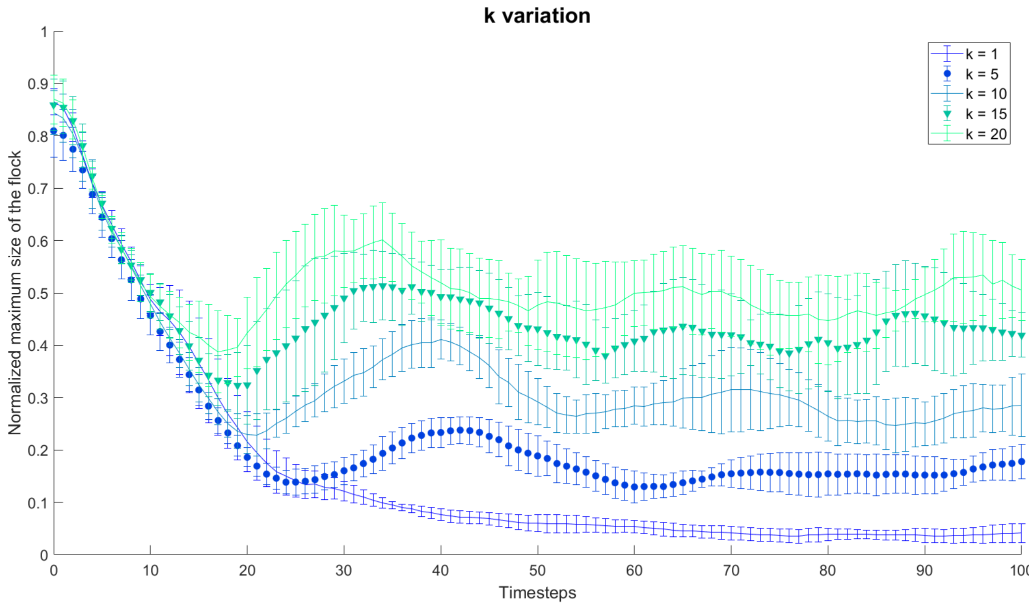


Figure 4: k-nearest parameter sweep (maximum distance mean and standard deviation)

- Parameter sweep of the number of the angle of the field of view with the k-nearest & field of view mode:

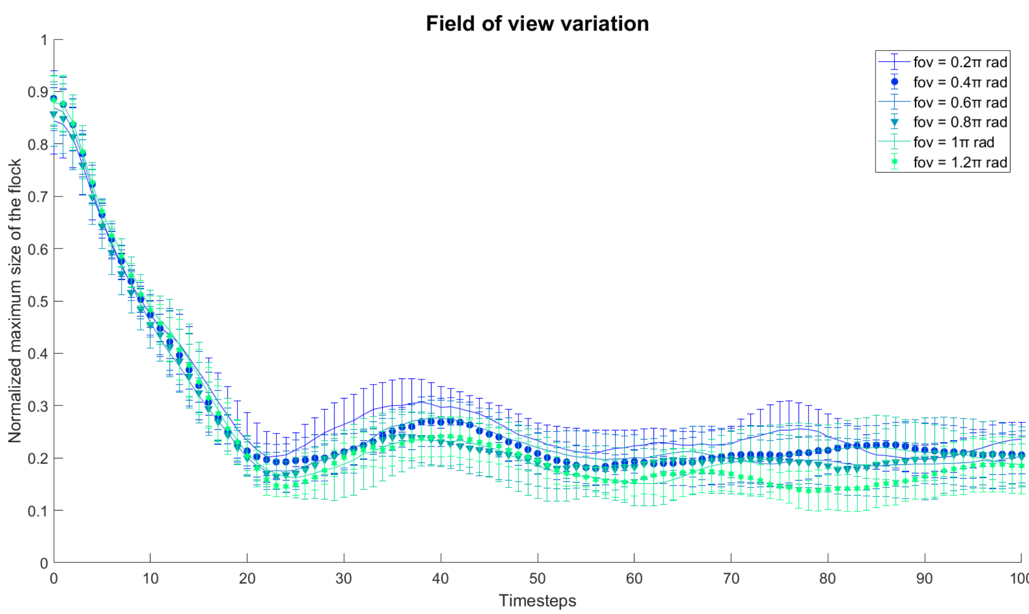


Figure 5: Field of vision angle parameter sweep: maximum distance mean and standard deviation

- Parameter sweep of the radius of the zone of repulsion with k-nearest neigbors (not considering field of view):

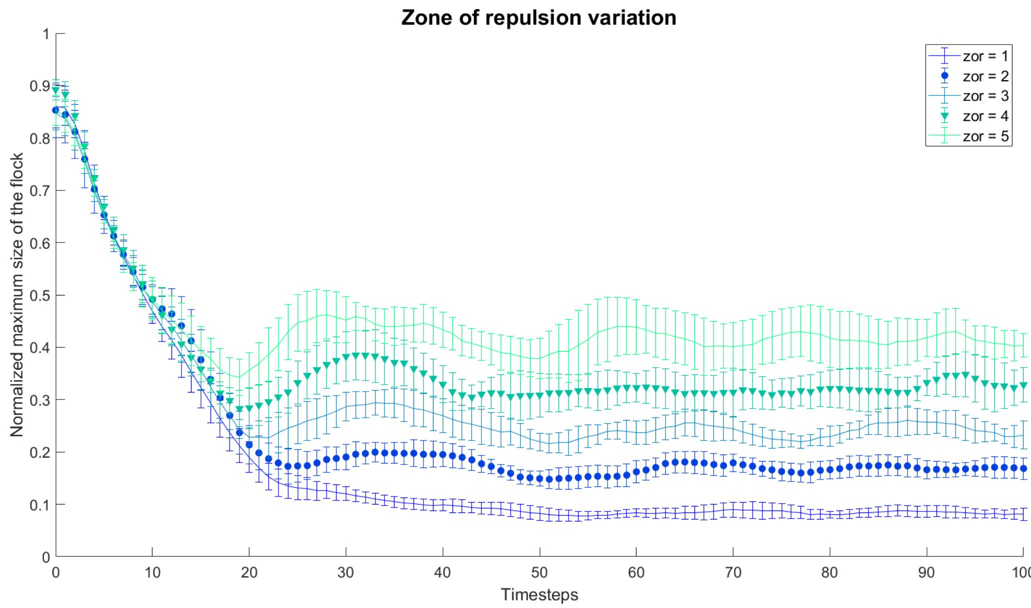


Figure 6: Zone of repulsion parameter sweep: maximum distance mean and standard deviation

5 Conclusion and discussion

The conclusions that can be drawn may be separated in three bullet items:

- When dealing with a relation between boids of the k-nearest neighbors that are located in the field of view of each boid, flock dispersion is very sensitive to the number of individuals k (see Fig. 4). The behavior that has been analyzed appears beginning from the timestep 20 - the decline of normalized maximum size from 0 to 20 is accounted to the random dispersion of boids at first until they form a flock. If k is low, flock distance is small, having that the normalized maximum distance goes even beyond 0.1. Increasing the number of boids which form the neighborhood of certain boid, rise the maximum size of the flock and also give more instabilities, having a size varying flock. Given that there are 50 individuals, when the number k is big enough, the increase in normalized maximum distance in smaller (the increase between the means of $k = 1$ and $k = 10$ is larger than the increase between $k = 10$ and $k = 20$).
- However, the same system as before with fixed $k = 6$ (as suggested from the paper) with a variation of the angle of the field of view, showed that the system is less susceptible to the variation of the `fov` angle than the variation of k (see Fig. 5).
- Finally, taking the system that only depends on the k-nearest neighbors and varying the minimum radius (see Fig. 6) it can be seen a predictable result: as the minimum distance between boids increases, the maximum size of the flock also increases. A less expected result is that the oscillations also increase when the radius of repulsion is bigger.

Further developments in this study can consist of a parameter sweep of multiple variables at the same time and the inclusion of other possible neighbors types to get a realistic model.

Code listing

`initialization.m` file

```
function [boid, pred] = initialization(dim,n,prd)
% INITIALIZATION: function to create random initial boid (and predator)
% positions and velocity
%
% INPUTS:
% n     = number of boids
% dim = size of the domain
% prd = if True, include the predator in the simulation
%
% OUTPUTS:
% boids = nx4 with n boids position and velocity in both components
% pred  = 1x4 with predator position and velocity in both components
%
% Xing Jin and Javier Lobato, created on 2018/04/03

% Loop over boids and fill in random position and velocity inside limits
boid(:,1:2) = rand(n,2)*dim;
boid(:,3:4) = (rand(n,2)*2-1)*dim*0.02;

% Initialize predator location and velocity if desired
if prd
    pred(:,1:2) = rand(1,2)*dim;
    % Contrary to the boids, initialize the predator with zero velocity
    pred(:,3:4) = [0 0];
else
    % Case where no predator is desired
    pred = [];
end

end
```

`rule1.m` file

```
function v1 = rule1(n,boid)
% RULE1: function that calculates the effect of Rule 1: boids try to fly
% towards the center of mass of the neighboring boids.
%
% INPUTS:
% n     = number of boids
% boids = boids position and velocity
%
% OUTPUTS:
% v1    = correction of the velocity for rule 1
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Given that the test will be carried out without any predator, this rule
```

```
% will not be updated with the different possible neighborhood cases

% Preallocation of the output matrix
v1 = zeros(n,2);

% Loop over n boids
for i = 1:n
    % Copy the position of all boids
    tem = boid(:,1:2);
    % Erase the row of the current i-th boid
    tem(i,:) = [];
    % Compute the mean of the position (relative center of gravity)
    xave = mean(tem(:,1));
    yave = mean(tem(:,2));
    % Direct the current boid position to the center of gravity
    v1(i,1:2) = [xave, yave] - boid(i,1:2);
end

end
```

rule2.m file

```
function v2=rule2(n,boid,zor,fov,k,vM)
% RULE2: function that calculates the effect of Rule 2: boids try to keep
% certain distance with their neighbors
%
% INPUTS:
% n     = number of boids
% boids = boids position and velocity
% zor   = (circular) zone of repulsion  / minimum distance (see below)
% fov   = angle (in radians) of the boid field of view
% k     = number of nearest boids
% vM    = string with the vision mode
%
% OUTPUTS:
% v2    = correction of the velocity for rule 2
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Preallocation of the output matrix
v2 = zeros(n,2);

% Preallocate space for the boids that are inside the field of view
fovInd = zeros([length(boid),1]);
fovCount = 1;

% Loop over n boids
for i = 1:n
    % Copy the position of all boids
    tem = boid(:,1:2);
    % Instead of erasing the row of the current i-th boid, a comparison
    % inside the loop will be done. This is done in order to mantain the
```

```
    % indexes of the original matrix the same
    % Get the current i-th boid
    bi = boid(i,1:2);
    % Loop over the other n-1 boids for zor, fov and zf
    for j = 1:n
        if j ~= i
            % Store the relative position vector between boid j and boid i
            dis = tem(j,:) - bi;
            switch vM
                case 'zor'
                    % If the modulus of the distance is smaller than the
                    % specified radius
                    if abs(norm(dis)) < zor
                        % Repel the i-th bird with respect the j-th bird
                        % and accumulate the consecutive repulsion of all
                        % the n-1 birds
                        v2(i,1:2) = v2(i,1:2) - dis;
                    end
                case 'fov'
                    % If the angle is included within the field of vision
                    if acos(dot(dis,boid(i,3:4))/(norm(dis)*norm(boid(i,3:4)))) < fov/2
                        % Repel the i-th bird with respect the j-th bird
                        % and accumulate the consecutive repulsion of all
                        % the n-1 birds
                        v2(i,1:2) = v2(i,1:2) - dis;
                    end
                case 'kn'
                    % Do nothing inside this for loop
                case 'zf'
                    % If the modulus of the distance is smaller than the
                    % specified radius and the angle is included within the
                    % field of vision
                    if abs(norm(dis)) < zor && acos(dot(dis,boid(i,3:4))/(norm(dis)*norm(boid(i,3:4)))) < fov/2
                        % Repel the i-th bird with respect the j-th bird
                        % and accumulate the consecutive repulsion of all
                        % the n-1 birds
                        v2(i,1:2) = v2(i,1:2) - dis;
                    end
                case 'fk'
                    % If the angle is included within the field of vision
                        if acos(dot(dis,boid(i,3:4))/(norm(dis)*norm(boid(i,3:4)))) < fov/2
                            % Just save the birds that are inside fov
                            fovInd(fovCount) = j;
                            fovCount = fovCount + 1;
                        end
                otherwise
                    error('Bad vision mode selection')
            end
        end
    end
    % Remove the zeros of the preallocated matrix
    fovInd = fovInd(1:fovCount-1);
    % Don't enter the loop if there is no birds in the field of view
    if sum(fovInd) ~= -2
        switch vM
            case 'zor'
```

```
                % Everything done
            case 'fov'
                % Everything done
            case 'kn'
                disXY = boid(:,1:2) - boid(i,1:2);
                dist = sqrt(disXY(:,1).^2+disXY(:,2).^2);
                [~,I] = sort(dist);
                % ind will have a distance of zero so let's remove it
                for j=1:length(I)
                    dis = tem(I(j),:) - bi;
                    % Given that these modes can't be combined with zone of
                    % repulsion, that input value will be used as the minimum
                    % distance between boids
                    if norm(dis) < zor
                        v2(i,1:2) = v2(i,1:2) - dis;
                    end
                end
            case 'zf'
                % Everything done
            case 'fk'
                % If field of view is also used, only the birds inside of
                % it will be considered to compute the closest ones
                disXY = boid(fovInd,1:2) - boid(i,1:2);
                dist = sqrt(disXY(:,1).^2+disXY(:,2).^2);
                [~,I] = sort(dist);
                I = fovInd(I);
                if k > length(I)
                    I = I(1:length(I));
                else
                    I = I(1:k);
                end
                for j=1:length(I)
                    dis = tem(I(j),:) - bi;
                    % Given that these modes can't be combined with zone of
                    % repulsion, that input value will be used as the minimum
                    % distance between boids
                    if norm(dis) < zor
                        v2(i,1:2) = v2(i,1:2) - dis;
                    end
                end
            otherwise
                error('Bad vision mode selection')
        end
    end
end

end
```

`rule3.m` file

```
function v3 = rule3(n,boid)
% RULE3: function that calculates the effect of Rule 3: boids try to match
% the velocity with the mean velocity
%
% INPUTS:
% n     = number of boids
% boids = boids position and velocity
%
% OUTPUTS:
% v3    = correction of the velocity for rule 3
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Given that the test will be carried out without any predator, this rule
% will not be updated with the different possible neighborhood cases

% Preallocation of the output matrix
v3 = zeros(n,2);

% Loop over n boids
for i = 1:n
    % Copy the velocity of all boids
    tem = boid(:,3:4);
    % Erase the row of the current i-th boid
    tem(i,:) = [];
    % Compute the mean of the veloicty
    vxave = mean(tem(:,1));
    vyave = mean(tem(:,2));
    % Match the mean velocity with the i-th boid's velocity
    v3(i,1:2) = [vxave,vyave] - boid(i,3:4);
end

end
```

`rule4.m` file

```
function v4 = rule4(n,boid,pred,zop,fov)
% RULE4: function that calculates the effect of Rule 4: boids try to avoid
% a possible incoming predator
%
% INPUTS:
% n     = number of boids
% boids = boids position and velocity
% pred  = current predator position and velocity
% zop   = (circular) zone of predator avoidance
% fov   = angle (in radians) of the boid field of view
%
% OUTPUTS:
% v4    = correction of the velocity for rule 4
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Given that the test will be carried out without any predator, this rule
% will not be updated with the different possible neighborhood cases

% Preallocation of the output matrix
v4 = zeros(n,2);

% Loop over n boids
for i = 1:n
    % Get the distance of the i-th boid to the  predator
    dis = boid(i,1:2) - pred(1:2);
    % Get the current boid velocity and transpose
    vel = (boid(i,3:4))';
    % If both the predator is inside the field of view and the radius of
    % the zone of predator avoidance
    if acos(dot(dis,vel)/norm(dis)/norm(vel))<fov/2 && abs(norm(dis))<zop
        % Modify the velocity of the i-th boid to avoid the predator
        v4(i,1:2) = (boid(i,1:2)-pred(1:2));
    end
end

end
```

`boundposition.m` file

```matlab
function v5 = boundposition(boids, limits, correction)
%BOUNDPOSITION: function that returns the boids into the limits
%
% INPUTS:
% boids      = boids position (do NOT include velocity)
% limits     = limits in which the position will be bound
% correction = velocity that will be used to revert the boid inside bounds
%
% OUTPUTS:
% v5         = correction of the velocity for rule 5
%
% Xing Jin and Javier Lobato, created on 2018/04/03

% Preallocation of the output matrix
v5 = zeros(length(boids),2);

% Loop over n boids
for i=1:length(boids)
    % Minimum horizontal axis position
    if boids(i,1) < limits(1)
        % Positive velocity increment
        v5(i,1) = correction;
    % Maximum horizontal axis position
    elseif boids(i,1) > limits(2)
        % Negative velocity increment
        v5(i,1) = - correction;
    end

    % Minimum vertical axis position
    if boids(i,2) < limits(3)
        % Positive velocity increment
        v5(i,2) = correction;
    % Maximum vertical axis position
    elseif boids(i,2) > limits(4)
        % Negative velocity increment
        v5(i,2) = - correction;
    end
end

end
```

`limitvelocity.m` file

```
function vel = limitvelocity(boidVel, vlim)
% LIMITVELOCITY: function that limitates the maximum possible velocity of
% the boids
%
% INPUTS:
% boidVel = boids velocity (do NOT include position)
% vlim    = maximum allowable boid velocity
%
% OUTPUTS:
% vel = new boid velocity
%
% Xing Jin and Javier Lobato, created on 2018/04/03

% Preallocation of the output matrix
vel = zeros(length(boidVel),2);

% Loop over all the n boids
for i = 1:length(boidVel)
    % If the modulus of the velocity of the i-th boid is over the limit
    if norm(boidVel(i,:)) > vlim
        % Set the velocity to keep the same direction but the magnitude
        % established in the limit velocity
        vel(i,:) = vlim*(boidVel(i,:)/norm(boidVel(i,:)));
    else
        % If it is under the limit, don't modify the velocity
        vel(i,:) = boidVel(i,:);
    end
end

end
```

`boundary.m` file

```
function boid = boundary(dim,boid)
% BOUNDARY: function that bounces the boids inside the grid if they get to
% the walls with the same velocity it hit the wall
%
% INPUTS:
% dim  = size of the domain
% boid = boids position and velocity
%
% OUTPUTS:
% boid = boids position and velocity
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% The changes made in this function are related to an increase in the
% velocity of the boids when they hit the wall. This was done to reduce the
% possibility that a boid hits the wall too slow and the bouncing velocity
% was also too slow (taking too much time to move towards the flock)

% Loop over all the boids
for j = 1:size(boid,1)
    % Bounce on the horizontal direction
    % If the boid moves to negative x-axis
    if boid(j,1) < 0
        boid(j,1) = - boid(j,1);
        boid(j,3) = - 10*boid(j,3);
    % If it leaves the grid on the right side
    elseif boid(j,1) > dim
        boid(j,1) = 2*dim - boid(j,1);
        boid(j,3 )= - 10*boid(j,3);
    end
    % Bounce on the vertical direction
    % If the boid moves to the negative y-axis
    if boid(j,2) < 0
        boid(j,2) = - boid(j,2);
        boid(j,4) = - 10*boid(j,4);
    % If the boid moves upper a distance of dim
    elseif boid(j,2) > dim
        boid(j,2) = 2*dim - boid(j,2);
        boid(j,4) = - 10*boid(j,4);
    end
end

end
```

`updateboid.m` file

```
function boid = updateboid(dim,n,boid,prd,pred,coe,zor,zop,fov,k,velLim,velCorr,vM)
% UPDATEBOID: updates the position of each boid using the different rules
%
% INPUTS:
% dim     = size of domain
% n       = number of boids
% boids   = boids position and velocity
% prd     = predator flag
% coe     = weighting of the velocity from each rule
% zor     = radius of the zone of repulsion
% zop     = radius of the zone of predator avoidence
% fov     = field of view angle
% k       = number of nearest neighbors
% velLim  = limit on the boid velocity
% velCorr = correction velocity if the boid moves outside bounds
% vM      = vision mode of the boids
%
% OUTPUTS:
% boid  = array with updated position and velocities
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Compute the variations in velocity for each rule
v1 = rule1(n,boid);
v2 = rule2(n,boid,zor,fov,k,vM);
v3 = rule3(n,boid);

% The fourth rule will one be computed if the predator flag is True
if prd
    v4 = rule4(n,boid,pred,zop,fov);
else
    v4 = 0;
end

% Update the velocity with the variations and their respective weights
boid(:,3:4) = boid(:,3:4) + coe(1)*v1 + coe(2)*v2 + coe(3)*v3 + coe(4)*v4;
% If the velocity if over the limit, correct it
boid(:,3:4) = limitvelocity(boid(:,3:4), velLim);

% Update the position with the new velocity
boid(:,1) = boid(:,1) + boid(:,3);
boid(:,2) = boid(:,2) + boid(:,4);

% Correct velocity if the boid is going out of bounds (bounds = +-0.1*dim)
boid(:,3:4) = boid(:,3:4) + boundposition(boid(:,1:2), ...
    [dim/10, 9*dim/10, dim/10, 9*dim/10], velCorr);

% If boids are in the boundary, bounce them
boid = boundary(dim,boid);

end
```

`updatepred.m` file

```
function pred = updatepred(dim, n, boid, pred, prdSpeed)
% UPDATEPRED: updates the position of the predator
%
% INPUTS:
% dim      = size of domain
% n        = number of boids
% boid     = boids position and velocity
% pred     = predator position and velocity
% prdSpeed = predator maximum speed
%
% OUTPUTS:
% pred = array with updated position and velocity of the predator
%
% Xing Jin & Javier Lobato 2018/04/03

% Get the true center of gravity of the flock
c = sum(boid(:,1:2))/n;

% Move the predator with the desired speed towards the center of gravity
pred(3:4) = prdSpeed*(c-pred(1:2))/norm(c-pred(1:2));

% Update the position of the predator with the new velocity
pred(1) = pred(1)+pred(:,3);
pred(2) = pred(2)+pred(:,4);

% If the predator is over the boundaries of the grid, it must also bounce
pred = boundary(dim, pred);

end
```

`pltdistribution.m` file

```
function pltdistribution(dim,boid,prd,pred,arrow,plotFov, zor, fov,k,ind,visionMode)
% PLTDISTRIBUTION: function that plots the position of the boids with a set
% of flags for customization
%
% INPUTS:
% dim        = dimension of the grid
% boid       = boids with position and velocity
% prd        = predator flag
% pred       = predator position and velocity
% arrow      = arrow plotting flag
% plotFov    = field of vision/zone of repulsion plotting
% zor        = zone of repulsion radius
% fov        = field of vision angles
% k          = number of closest neighbors
% ind        = index of the individual to plot the 'zor' and 'fov' over
% visionMode = vision mode configuration to plot
%
% OUTPUTS:
% Plot with the results
```

```
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Delete previous arrows (i.e. annotations)
delete(findall(gcf,'type','annotation'))
% Plot all points with blue circles
scatter(boid(:,1),boid(:,2),...
    10,'o','filled','MarkerFaceColor','b')
% Get the position of the current figure for the arrows
pos = get(gca, 'Position');

% Preallocate space for the boids that are inside the field of view
fovInd = zeros([length(boid),1]);
fovCount = 1;

% If the zor & fov is wanted to be shown
if plotFov
    % Get a list with all index from 1 to n
    listInd = linspace(1,length(boid),length(boid));
    % Set as empty the index of the individual ind
    listInd(ind) = [];
    hold on
    % Plot that individual as a blue square to make it bigger
    scatter(boid(ind,1),boid(ind,2),...
        30,'s','filled','MarkerFaceColor','b')
    hold off
    % Get the velocity vector of the individual ind
    vel=(-boid(ind,3:4))';
    % Create and arrow pointing in the direction of individual ind movement
    annotation('arrow', [boid(ind,1)/dim*pos(3) + pos(1),...
     (boid(ind,1)+boid(ind,3))/dim*pos(3) + pos(1)],...
     [boid(ind,2)/dim*pos(4) + pos(2),...
     (boid(ind,2)+boid(ind,4))/dim*pos(4) + pos(2)],...
     'Color','b');
    % Draw a circle on the zone of repulsion
    if visionMode(1) == 1
        th = 0:pi/50:2*pi;
        xunit = zor * cos(th) + boid(ind,1);
        yunit = zor * sin(th) + boid(ind,2);
        hold on
        h = plot(xunit, yunit, '--r');
        hold off
    end
    if visionMode(2) == 1
        % Draw two lines that will define the field of vision of the boid
        arc = atan(boid(ind,4)/boid(ind,3));
        % Taking into account the possible returns of the atan function
        if boid(ind,3) > 0
            hold on
            plot([boid(ind,1),boid(ind,1)-dim^2*cos(pi+arc+fov/2)],...
                [boid(ind,2),boid(ind,2)-dim^2*sin(pi+arc+fov/2)],':k')
            plot([boid(ind,1),boid(ind,1)-dim^2*cos(pi+arc-fov/2)],...
                [boid(ind,2),boid(ind,2)-dim^2*sin(pi+arc-fov/2)],':k')
            hold off
        else
            hold on
```

```
            plot([boid(ind,1),boid(ind,1)-dim^2*cos(arc+fov/2)],...
                [boid(ind,2),boid(ind,2)-dim^2*sin(arc+fov/2)],':k')
            plot([boid(ind,1),boid(ind,1)-dim^2*cos(arc-fov/2)],...
                [boid(ind,2),boid(ind,2)-dim^2*sin(arc-fov/2)],':k')
            hold off
        end
    end
    % Loop over all the possible individuals except ind
    for i=listInd
        % Compute the vectorial distance from ind to the i-th individual
        dis=boid(ind,1:2)-boid(i,1:2);
        if visionMode(1) == 1
            % If the i-th boid is inside the zor
            if abs(norm(dis)) < zor
                % Plot a downwards pointing triangle
                hold on
                scatter(boid(i,1),boid(i,2),...
                    25,'v','filled','MarkerFaceColor','k')
                hold off
            end
        end
        if visionMode(2) == 1
            % If the i-th boid is inside the field of view
            if acos(dot(dis,vel)/(norm(dis)*norm(vel))) < fov/2
                % Save the birds that are inside the field of view
                fovInd(fovCount) = i;
                fovCount = fovCount + 1;
                % Plot an upwards pointing triangle
                hold on
                scatter(boid(i,1),boid(i,2),...
                    25,'^','filled','MarkerFaceColor','k')
                hold off
            end
        end
    end
    % Remove the zeros of the array to avoid index problems
    fovInd = fovInd(1:fovCount-1);
    % Plotting of the k-nearest boids
    if visionMode(3) == 1
        % If field of view is also used, only the birds inside of it will
        % be considered to compute the closest ones
        if visionMode(2) == 1
            disXY = boid(fovInd,1:2) - boid(ind,1:2);
            dis = sqrt(disXY(:,1).^2+disXY(:,2).^2);
            [~,I] = sort(dis);
            I = fovInd(I);
            I = I(1:k);
            if k > length(I)
                I = I(1:length(I));
            else
                I = I(1:k);
            end
        % Otherwise all birds will be analyzed
        else
            disXY = boid(:,1:2) - boid(ind,1:2);
            dis = sqrt(disXY(:,1).^2+disXY(:,2).^2);
            [~,I] = sort(dis);
```

```matlab
                % ind will have a distance of zero so let's remove it
                I = I(2:k+1);
                if k > length(I)
                    I = I(1:length(I));
                else
                    I = I(1:k);
                end
            end
            % Plot a downwards pointing triangle for the k nearest boids
            for i=I
                hold on
                scatter(boid(i,1),boid(i,2),...
                        25,'v','filled','MarkerFaceColor','k')
                hold off
            end
        end
    end



% If the arrows are desired
if arrow
    % Loop over all possible boids
    for i=1:length(boid)
        % Plot the arrow as an annotation object
        annotation('arrow', [boid(i,1)/dim*pos(3) + pos(1),...
             (boid(i,1)+boid(i,3))/dim*pos(3) + pos(1)],...
             [boid(i,2)/dim*pos(4) + pos(2),...
             (boid(i,2)+boid(i,4))/dim*pos(4) + pos(2)],...
             'Color','b');
    end
end

% If there is a predator
if prd
    % Plot it as a red square
    hold on
    scatter(pred(1),pred(2),30,'s','filled','MarkerFaceColor','r')
    hold off
    % Plot the arrow of the predator to know where it is pointing
    annotation('arrow', [pred(1)/dim*pos(3) + pos(1),...
         (pred(1)+pred(3))/dim*pos(3) + pos(1)],...
         [pred(2)/dim*pos(4) + pos(2),...
         (pred(2)+pred(4))/dim*pos(4) + pos(2)],...
         'Color','r');
end

% Fix axis of the plot
axis([0 dim 0 dim])
% Set the fontsize of the axis as 16
set(gca,'FontSize',14)
end
```

`swarmModel.m` file

```
function [max_dist] =
↪   swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arr,figNo,visionMode)
%SWARMMODEL: simulation of a BOIDS swarm model based on the pseudo code
%taken from http://www.vergenet.net/~conrad/boids/pseudocode.html
%
% INPUTS:
% n          = number of boids
% dim        = size of the domain
% ts         = simulation time
% zor        = radius of the zone of repulsion
% zop        = radius of the zone of predator avoidance
% fov        = field of view angle
% k          = number of closest neeighbors
% coe        = weighting of the velocity from each rule
% prd        = predator flag
% prdSpd     = predator speed
% vLim       = velocity limit for the boids
% vCorr      = velocity correction for the boids leaving bounds
% plop       = plotting  options, true to plot, false to not
% pltFov     = zor and fov plotting flag
% arr        = arrows plotting flag
% figNo      = number of figure to avoid superposition
% visionMode = vector with three components for different neighborhoods
%
% OUTPUTS:
% plot of the desired simulation case setup
%
% Xing Jin and Javier Lobato, created on 2018/04/03
% Javier Lobato, last modified on 2018/04/15

% Initialize position and velocity of the flock
[boid,pred] = initialization(dim,n,prd);

% Create a new figure to avoid superposition
if plop
    figure(figNo)
end

% Test visionMode vector to analyze its possibilities:
%       [0 0 0] -> not valid configuration
%       [1 0 0] -> zor (zone of repulsion)
%       [0 1 0] -> fov (field of vision)
%       [0 0 1] -> kn (k-nearest boids)
%       [1 1 0] -> zf (zone of repulsion + field of vision)
%       [1 0 1] -> not valid configuration (additive configuration)
%       [0 1 1] -> fk (field of vision + k-nearest)
%       [1 1 1] -> not valid configuration (additive configuration)

% The invalid configurations will raise an error and exit the function
if sum(visionMode) == 3
    error('Not valid visionMode vector')
    % To force the system to leave the function, let's limit ts as zero
    ts = 0;
elseif sum(visionMode) == 0
```

```
    error('Not valid visionMode vector')
    ts = 0;
elseif sum(visionMode) == 2
    if visionMode(3) == 0
        vM = 'zf';
    elseif visionMode(1) == 0
        vM = 'fk';
    else
        error('Not valid visionMode vector')
        ts = 0;
    end
elseif sum(visionMode) == 1
    if visionMode(1) == 1
        vM = 'zor';
    elseif visionMode(2) == 1
        vM = 'fov';
    else
        vM = 'kn';
    end
end

% Preallocation of space for the maximum distance array
max_dist = zeros([(ts+1),1]);

% Calculation of the maxmimum distance for the initialization
max_dist(1) = maxDist(boid(:,1:2));

% If the pltFov flag is True, get the index for the boid to track
if pltFov
    ind = randsample(length(boid),1);
    boid(ind,1:2) = dim/2;
else
    ind = 1; % Otherwise define it as 1 although it won't be used
end

% Time loop from 1 until the specified ts
for i=1:ts
    % Update the position of the boids
    boid = updateboid(dim,n,boid,prd,pred,coe,zor,zop,fov,k,vLim,vCorr,vM);
        % (If there is a predator, also update it) though it will never be
        % a predator in the current experiment setup
        if prd
             pred = updatepred(dim,n,boid,pred,prdSpd);
        end
    % Plot current position of boids and predator if desired
    if plop
        pltdistribution(dim,boid,prd,pred,arr,pltFov, zor, fov, k, ind, visionMode)
    end
    % Pause to see the figure
    pause(0.001)
    % Calculations of the largest distance between birds
    max_dist(i+1) = maxDist(boid(:,1:2));
end

end
```

`maxDist.m` file

```
function [ max_dist ] = maxDist(boidPos)
% MAXDIST: function that computes the maximum distance between boids in the
% current flock distribution. Computing the distance between one bird to
% every other bird, taking the maximum, repeating that for all birds and
% taking the maximum of the maximums is an inefficient way of computing the
% maximum distance between two points. Following the advice desrcibed:
%            https://stackoverflow.com/a/8006849
% the maximum distance between the boids can be computed without the need
% of looping twice the array
%
% INPUTS:
% boidPos  = position of the boids in the flock
%
% OUTPUTS:
% max_dist = value of the maximum distance
%
% Javier Lobato, created on 2018/04/15

% Let's add the components x+y and store them
adding = boidPos(:,1) + boidPos(:,2);
% Let's substract the components x-y and store them
substracting = boidPos(:,1) - boidPos(:,2);

% Lower left point is the min(x+y)
[~, ill] = min(adding);
% Upper left point is the min(x-y)
[~, iul] = min(substracting);
% Upper right point is the max(x+y)
[~, iur] = max(adding);
% Lower right point is the max(x-y)
[~, ilr] = max(substracting);

% Let's compute the distance between the most lower left and most upper
% right point
d1 = sqrt((boidPos(ill,1)-boidPos(iur,1))^2 + (boidPos(ill,2)-boidPos(iur,2))^2);
% Let's compute the distance between the most upper left and most upper
% left point
d2 = sqrt((boidPos(iul,1)-boidPos(ilr,1))^2 + (boidPos(iul,2)-boidPos(ilr,2))^2);

% Take the biggest of these two measures
max_dist = max(d1,d2);

end
```

HW4b_driver.m file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              HOMEWORK #4.B                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Javier Lobato, created on 2018/04/15
close all; clear all; clc

% Mode of vision for the flock of birds
% The elements of the vector are:
%    - proximity (circle of repulsion)
%    - field of view (angle of the bird vision)
%    - k-nearest birds (taking into account only the closest birds)

%% Mode of vision: zone of repulsion
% Case variables
n = 200; dim = 200; ts = 1;
% Bird configuration
zor = 40; zop = 20; fov = 0.4*pi; k = 6; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configuration
plop = true; arrows = false; pltFov = true; figNo = 1;
% Mode of vision
visionMode = [true, false, false];
% Function calling
swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
title('Zone of repulsion');

%% Mode of vision: field of view
% Case variables
n = 200; dim = 200; ts = 1;
% Bird configuration
zor = 2; zop = 20; fov = 0.6*pi; k = 6; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configuration
plop = true; arrows = false; pltFov = true; figNo = 2;
% Mode of vision
visionMode = [false, true, false];
% Function calling
swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
title('Field of view');

%% Mode of vision: k-nearest neighbors
% Case variables
n = 200; dim = 200; ts = 1;
% Bird configuration
zor = 2; zop = 20; fov = 0.4*pi; k = 15; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = true; figNo = 3;
```

```
% Mode of vision
visionMode = [false, false, true];
% Function calling
swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
title('k-nearest neighbors');

%% Mode of vision: zone of repulsion + field of view
% Case variables
n = 200; dim = 200; ts = 1;
% Bird configuration
zor = 40; zop = 20; fov = 0.6*pi; k = 6; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = true; figNo = 4;
% Mode of vision
visionMode = [true, true, false];
% Function calling
swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
title('Zone of repulsion + Field of view');

%% Mode of vision: field of view + k-nearest neighbors
% Case variables
n = 200; dim = 200; ts = 1;
% Bird configuration
zor = 2; zop = 20; fov = 0.4*pi; k = 10; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = true; figNo = 5;
% Mode of vision
visionMode = [false, true, true];
% Function calling
swarmModel(n,dim,ts,zor,zop,fov,k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
title('Field of view + k-nearest neighbors');
```

`dataAnalysis.m` file

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              HOMEWORK #4.B                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Javier Lobato, created on 2018/04/15
close all; clear all; clc;

%% Parameter sweep of the k-nearest neighbors
% Case variables
n = 50; dim = 200; ts = 100;
% Bird configuration
zor = 2; zop = 20; fov = 0.8*pi; vLim = 30; vCorr = 3;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = false; figNo = 1;
% Mode of vision
visionMode = [false, true, true];
% Parameter sweep vector
k = [1,5,10,15,20];
% Number of runs
runs = 10;
% Maximum diameter for this parameter sweep
diam = zeros([length(k),runs,ts+1]);
% Function calling
for j=1:length(k)
    for i=1:runs
        diam(j,i,:) = swarmModel(n,dim,ts,zor,zop,fov,...
        k(j),coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
    end
end

figure(2)
% Plot the mean and standard deviation for k-neearest parameter sweep
c = (winter(length(k)));
lw = ['-o','-v','-h','-h','-^','-v'];
hold on
for j=1:length(k)
    errorbar(0:1:100,reshape(mean(diam(j,:,:)/sqrt(2*dim^2),2),[1,101]),...
    reshape(std(diam(j,:,:)/sqrt(2*dim^2)),[1,101]),lw(j),'Color',c(j,:),'MarkerFaceColor',c(j,:))
    legendList{j} = ['k = ',num2str(k(j))];
end
hold off
ylim([0,1])
ley = legend(legendList);
set(ley,'FontSize',14);
xlabel('Timesteps')
ylabel('Normalized maximum size of the flock')
set(gca,'FontSize',14)
title(['k variation'],'FontSize',20)

%% Parameter sweep of the field of vision
% Case variables
n = 50; dim = 200; ts = 100;
```

```
% Bird configuration
zor = 2; zop = 20; k = 6; vLim = 30; vCorr = 3;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = false; figNo = 1;
% Mode of vision
visionMode = [false, true, true];
% Parameter sweep vector
fov = [0.2*pi,0.4*pi,0.6*pi,0.8*pi,1.0*pi,1.2*pi];
% Number of runs
runs = 10;
% Maximum diameter for this parameter sweep
diam = zeros([length(k),runs,ts+1]);
% Function calling
for j=1:length(fov)
    for i=1:runs
        diam(j,i,:) = swarmModel(n,dim,ts,zor,zop,fov(j),...
        k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
    end
end

figure(3)
% Plot the mean and standard deviation for field of view parameter sweep
c = (winter(length(fov)));
lw = ['-o','-v','-h','-h','-^','-v'];
hold on
for j=1:length(fov)
    errorbar(0:1:100,reshape(mean(diam(j,:,:)/sqrt(2*dim^2),2),[1,101]),...
    reshape(std(diam(j,:,:)/sqrt(2*dim^2)),[1,101]),lw(j),'Color',c(j,:),'MarkerFaceColor',c(j,:))
    legendList{j} = ['fov = ',num2str(fov(j)/pi),'? rad'];
end
hold off
ylim([0,1])
ley = legend(legendList);
set(ley,'FontSize',14);
xlabel('Timesteps')
ylabel('Normalized maximum size of the flock')
set(gca,'FontSize',14)
title(['Field of view variation'],'FontSize',20)

%% Parameter sweep of the size of the zone of repulsion
% Case variables
n = 50; dim = 200; ts = 100;
% Bird configuration
zop = 20; fov = 0.8*pi; k = 6; vLim = 30; vCorr = 3;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configurationc
plop = true; arrows = false; pltFov = false; figNo = 1;
% Mode of vision
visionMode = [false, false, true];
% Parameter sweep vector
zor = [1,2,3,4,5];
% Number of runs
```

```
runs = 10;
% Maximum diameter for this parameter sweep
diam = zeros([length(k),runs,ts+1]);
% Function calling
for j=1:length(zor)
    for i=1:runs
        diam(j,i,:) = swarmModel(n,dim,ts,zor(j),zop,fov,...
        k,coe,prd,prdSpd,vLim,vCorr,plop,pltFov,arrows,figNo,visionMode);
    end
end

figure(4)
% Plot the mean and standard deviation for zone of repulsion parameter sweep
c = (winter(length(zor)));
lw = ['-o','-v','-h','-h','-^'];
hold on
for j=1:length(zor)
    errorbar(0:1:100,reshape(mean(diam(j,:,:)/sqrt(2*dim^2),2),[1,101]), ...
    reshape(std(diam(j,:,:)/sqrt(2*dim^2)),[1,101]),lw(j),'Color',c(j,:),'MarkerFaceColor',c(j,:))
    legendList{j} = ['zor = ',num2str(zor(j))];
end
hold off
ylim([0,1])
ley = legend(legendList);
set(ley,'FontSize',14);
xlabel('Timesteps')
ylabel('Normalized maximum size of the flock')
set(gca,'FontSize',14)
title(['Zone of repulsion variation'],'FontSize',20)
```