Homework 4.(a)

Modeling Complex Systems, Xing Jin & Javier Lobato

Due date: Tuesday, April 10, 2018

0 Problem overview

The boids are constructed as an array with position and velocity: two-dimensional position vector determined by its previous position and its current velocity and two dimensional velocity vector which is the weighted average of its old velocity, repulsion velocity, orientation velocity, attraction to the centroid velocity, and a possible predator avoidance velocity. To implement the boids swarming model using the Reynold's algorithm, the following MATLAB functions have been coded and implemented:

- `boundary.m`: bounce back the boid if it hits the limits of the domain
- `boundposition.m`: slows down boids if they reach an out-of-bounds domain
- `HW4a_driver.m`: driver to simulate the different cases
- `initialization.m`: initialize the boids (and predator) position and velocity
- `limitvelocity.m`: limit the maximum velocity at which a boid can move
- `pltdistribution.m`: plot the current boid/predator distribution
- `rule1.m`: calculate the relative centroid based on rule 1
- `rule2.m`: computes the zone of repulsion based on rule 2
- `rule3.m`: calculated the orientation velocity based on rule 3
- `rule4.m`: predator avoidance velocity correction
- `swarmModel.m`: function that executes the loop calling all necessary functions
- `testcase.m`: test call to prove repulsion area and field of view
- `updateboid.m`: update the position and velocity of the boids
- `updatepred.m`: update the position and velocity of the predator

All the code of these functions is listed at the end of this report. The code is ordered by apparition order instead of by alphabetical order.

1 Swarm model: attraction, orientation and repulsion

The following pictures prove that the code is working as it should:

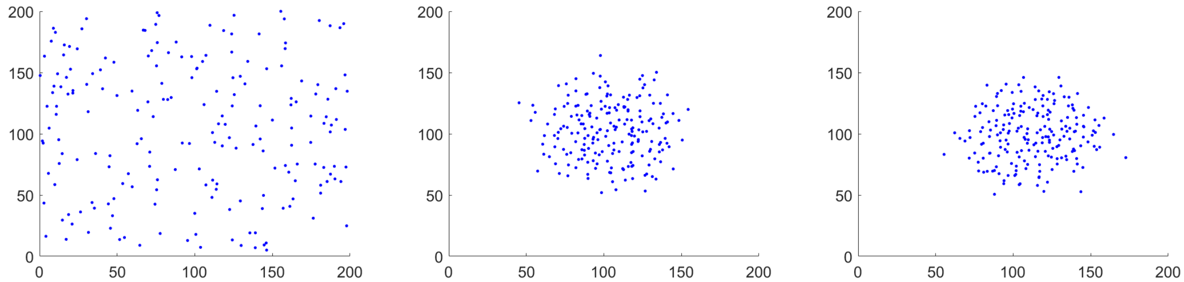

Figure 1: Attraction towards the center and zone of boid repulsion

Although the birds begin in scattered positions randomly distributed around the domain, they move towards other birds eventually creating a flock of birds. Those birds have cohesion among them but they also keep certain distance one to each other.

In the next figure, the velocity vector of the boids has been included so it can be seen where are they pointing and moving towards. The initial configuration of random position and velocity boids turns into an organized flock, having in the middle image that all boids are moving to the center of the flock and in the next one the boids rotate around the flock.

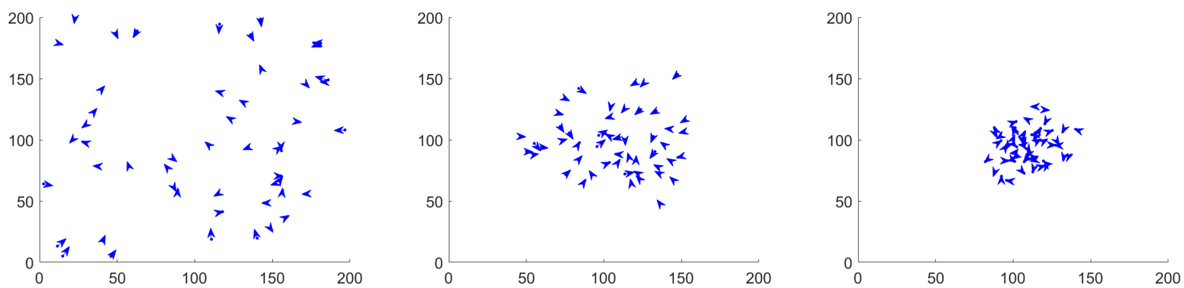

Figure 2: Attraction towards the center and velocity alignment

2 Zone of repulsion and field of view

In the figure below these lines, it can be seen two cases where the neighborhood of a bird is analyzed. The current bird has been chosen randomly and it has been fixed in the center of the flock to better see the surroundings. Both plots include a circle that represents the zone of repulsion of the bird and the dotted lines show the field of view of the bird. Points inside the zone of repulsion of the bird are represented with a downwards pointing triangle ($\blacktriangledown$) and the boids inside the field of view are represented with an upwards pointing triangle ($\blacktriangle$). Boids that are inside both zones have both symbols, that coalesce in a 6 pointed star (✶) as it can be seen in the left image. The right case has no coincidence between field of view and zone of repulsion. Boids outside both the zone of repulsion and field of view are represented by dots.

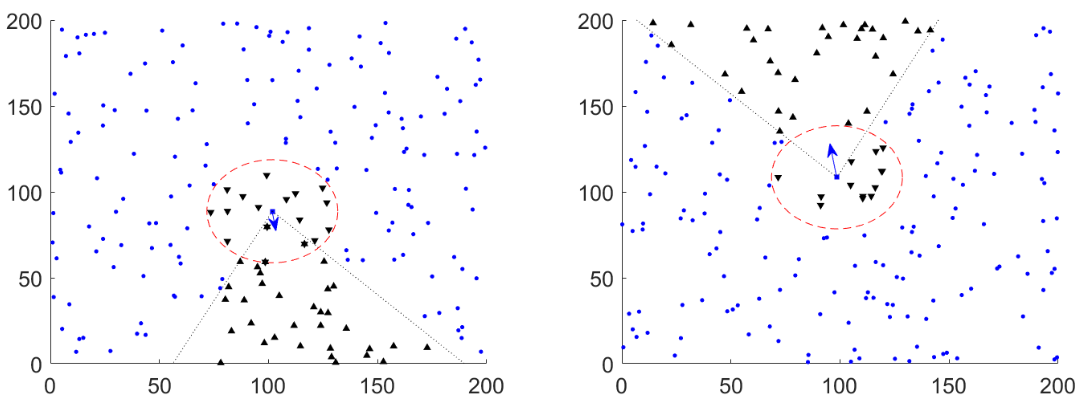

Figure 3: Zone of repulsion and field of view

3 Predator repulsion

The inclusion of a predator may have important consequences in the behavior of the flock. The arrow is the predator and the rule that it follows is to move towards the true centroid of the flock (the dots represent the birds). In the second and third image, it can be seen how the simulation advances in time, having that the boids repel the predator, leaving even the bound when the predator moves towards them in the third image.

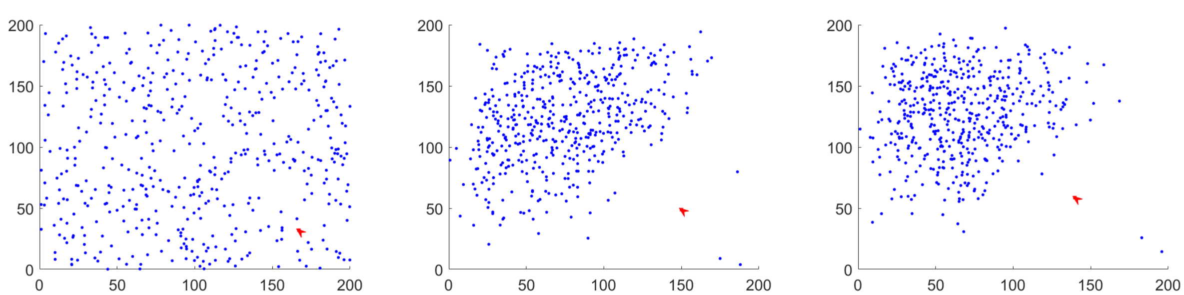

Figure 4: Predator repulsion

If instead of showing the behavior of a very large flock, the movement of just 10 boids and one predator is simulated, the mechanism of the code can be better analyzed. The boids that surround the predator them fly away from it in the completely opposite direction. In the second figure can be seen that the boids try to move towards a centroid of the flock, matching velocity with other birds. In the third figure, one boid was lagged from the flock, staying behind other birds - having a large increase in velocity to avoid the predator and move towards the centroid. In the fourth image the flock is formed but it will eventually break given that the predator moves towards its centroid.

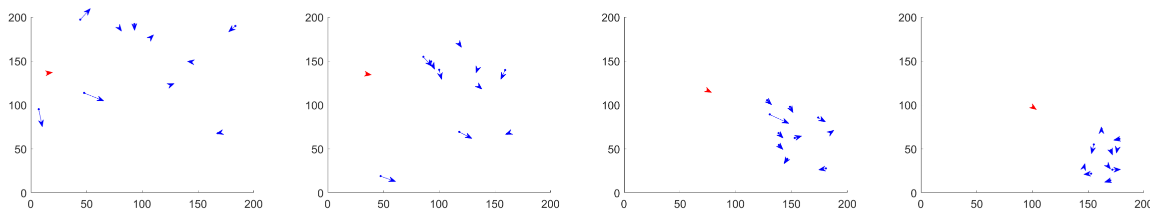

Figure 5: Boid movement and predator avoidance

4 Zone of repulsion and field of view: detailed approach

Another simulation with just two boids was carried out. If the two boids are facing one each other, when the zones of repulsions intersect, both boids will change immediately its direction (given that the other boid is inside the field of vision). This can be seen in the next set of images:

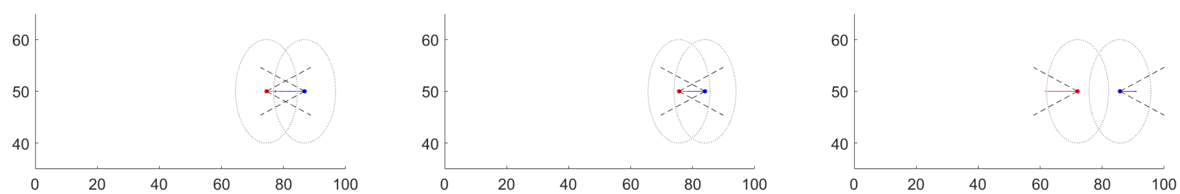

Figure 6: Facing boids

In the case that one boid is following another, only the rear boid will turn away from the leading boid because the field of vision of the leading one can't see the rear boid (although it may be inside its zone of repulsion). This is shown in the next figure:

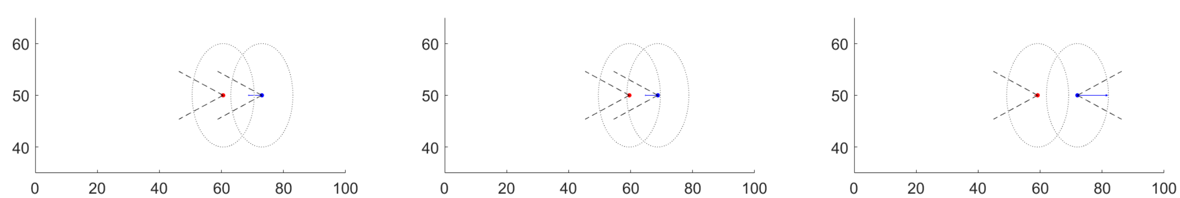

Figure 7: Rear vision boids

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

% 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)
% 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
% fov   = angle (in radians) of the boid field of view
%
% OUTPUTS:
% v2    = correction of the velocity for rule 2
%
% Xing Jin and Javier Lobato, created on 2018/04/03

% Preallocation of the output matrix
v2 = 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,:) = [];
    % Get the current i-th boid
    bi = boid(i,1:2);
    % Loop over the other n-1 boids
    for j = 1:n-1
        % Store the relative position vector between boid j and boid i
        dis = tem(j,:) - bi;
        % 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
    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

% 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

% 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

```
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

% 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) = - 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 )= - 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) = - 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) = - boid(j,4);
    end
end

end
```

`updateboid.m` file

```
function boid = updateboid(dim,n,boid,prd,pred,coe,zor,zop,fov,velLim,velCorr)
% 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
%
% OUTPUTS:
% boid  = array with updated position and velocities
%
% Xing Jin & Javier Lobato 2018/04/03

% Compute the variations in velocity for each rule
v1 = rule1(n,boid);
v2 = rule2(n,boid,zor,fov);
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,ind)
% 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
% ind     = index of the individual to plot the 'zor' and 'fov' over
%
% OUTPUTS:
% Plot with the results
%
% Xing Jin and Javier Lobato, created on 2018/04/03
```

```

% 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');

% 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
    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
    % 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
    % 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 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
        % If the i-th boid is inside the field of view
        if acos(dot(dis,vel)/(norm(dis)*norm(vel))) < fov/2
            % Plot an upwards pointing triangle
            hold on
            scatter(boid(i,1),boid(i,2),...
                25,'^','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',16)
end
```

`swarmModel.m` file

```
function swarmModel(n,dim,ts,zor,zop,fov,coe,prd,prdSpd,vLim,vCorr,pltFov,arr,figNo)
%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
% 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
% pltFov = zor and fov plotting flag
% arr = arrowd plotting flag
% figNo  = number of figure to avoid superposition
%
% OUTPUTS:
% plot of the desired simulation case setup
%
% Xing Jin and Javier Lobato, created on 2018/04/03

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

% Create a new figure to avoid superposition
figure(figNo)

% 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,vLim,vCorr);
        % If there is a predator, also update it
        if prd
             pred = updatepred(dim,n,boid,pred,prdSpd);
        end
    % Plot current position of boids and predator
    pltdistribution(dim,boid,prd,pred,arr,pltFov, zor, fov,ind)
    % Pause to see the figure
    pause(0.001)
end
end
```

`testcase.m` file

```
function testcase(dim,ts,zor,zop,fov,coe,prd,velLim,vCorr,figNo)
% TESTCASE: function to test some swarm behavior functionalities for a case
% with just 2 boids
%
% INPUTS:
% 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
% coe   = weighting of the velocity from each rule
% prd   = predator flag
% vLim  = velocity limit for the boids
% vCorr = velocity correction for the boids leaving bounds
% figNo = number of figure to avoid superposition
%
% OUTPUTS:
% Plot with the results of the swarming behavior
%
% Xing Jin and Javier Lobato, created on 2018/04/03

% Number of boids 2
n = 2;

% Preallocation of boid x-position, y-position, x-velocity and y-velocity
boid = zeros(n,4);

% Preallocation of pred x-position, y-position, x-velocity and y-velocity
pred = zeros(1,4);

% Angle correction
angle = fov/2;

% Initialization of the boids
boid(1,:) = [dim/2, dim/2, 10^-3, 0];
boid(2,:) = [0, dim/2, 1, 0];

% Create a new figure to avoid superposition
figure(figNo)

% Time loop
for i=1:ts
    % Update boids with the predefined configurations
    boid = updateboid(dim,n,boid,prd,pred,coe,zor,zop,fov,velLim, vCorr);
    % Plot current distribution without the outside function
    % First plot position of boid no. 1
    scatter(boid(1,1),boid(1,2),20,'o','filled','MarkerFaceColor','b')
    % Plot velocity of boid no. 1
    hold on
    quiver(boid(1,1),boid(1,2),boid(1,3),boid(1,4),3,'b');
    % Plot position of boid no. 2
    scatter(boid(2,1),boid(2,2),20,'o','filled','MarkerFaceColor','r')
    % Plot velocity of boid no. 2
    quiver(boid(2,1),boid(2,2),boid(2,3),boid(2,4),3,'r');
```

```
    % Zone of repulsion of boid no. 1
    viscircles([boid(1,1) boid(1,2)],zor,...
        'LineStyle',':','color','k','LineWidth',.5);
    % Zone of repulsion of boid no. 2
    viscircles([boid(2,1) boid(2,2)],zor,...
        'LineStyle',':','color','k','LineWidth',.5);

    % Field of vision plotting for boids no. 1 and no. 2
    xx = [boid(1,3) boid(1,3)*tan(angle)];
    xx = xx/norm(xx)*zor*1.5;
    yy = [boid(2,3) boid(2,3)*tan(angle)];
    yy = yy/norm(yy)*zor*1.5;
    if fov>pi
        xx = -xx;
        yy = -yy;
    end
    x1 = [boid(1,1) boid(1,1)+xx(1)];
    x2 = [boid(2,1) boid(2,1)+yy(1)];
    y1 = [boid(1,2) boid(1,2)+xx(2)];
    y2 = [boid(1,2) boid(1,2)-xx(2)];
    y3 = [boid(2,2) boid(2,2)+yy(2)];
    y4 = [boid(2,2) boid(2,2)-yy(2)];
    plot(x1,y1,'LineStyle','--','color','k','LineWidth',.5)
    plot(x1,y2,'LineStyle','--','color','k','LineWidth',.5)
    plot(x2,y3,'LineStyle','--','color','k','LineWidth',.5)
    plot(x2,y4,'LineStyle','--','color','k','LineWidth',.5)

    % Select axis limits and aspect ratio
    axis([0 dim 7/20*dim 13*dim/20])
    pbaspect([2 1 1])
    set(gca,'FontSize',16)
    hold off

    %Pause to see the figure instantaneously
    pause(0.001)
end

end
```

HW4a_driver.m file

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               HOMEWORK #4.A                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Xing Jin and Javier Lobato, created on 2018/04/03

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

%%
% Case 1: swarming behavior and general code testing
% (Variables are described in the function comments)

% Case variables
n = 200; dim = 200; ts = 100;
% Bird configuration
zor = 4; zop = 20; fov = 2.*pi; vLim = 20; vCorr = 5;
coe = [1/100,1/2,1/8,0];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configuration
arrows = false; pltFov = false; figNo = 1;
% Function calling for case 1
swarmModel(n,dim,ts,zor,zop,fov,coe,prd,prdSpd,vLim,vCorr,pltFov,arrows,figNo)

%%
% Case 2: implementation of the zone of repulsion and field of view

% Case variables
n = 200; dim = 200; ts = 2;
% Bird configuration
zor = 30; zop = 20; fov = 0.4*pi; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = false; prdSpd = 0;
% Plotting configuration
arrows = false; pltFov = true; figNo = 2;
% Function calling for case 2
swarmModel(n,dim,ts,zor,zop,fov,coe,prd,prdSpd,vLim,vCorr,pltFov,arrows,figNo)

%%
% Case 3: predator avoidance setup 1 - zone of predator avoidance

% Case variables
n = 500; dim = 200; ts = 20;
% Bird configuration
zor = 4; zop = 60; fov = 1.8*pi; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = true; prdSpd = 2;
% Plotting configuration
arrows = false; pltFov = false; figNo = 3;
% Function calling for case 3
swarmModel(n,dim,ts,zor,zop,fov,coe,prd,prdSpd,vLim,vCorr,pltFov,arrows,figNo)

```

```
%%
% Case 4: predator avoidance setup 2 - arrows implementation

% Case variables
n = 10; dim = 200; ts = 25;
% Bird configuration
zor = 8; zop = 50; fov = 1.8*pi; vLim = 20; vCorr = 5;
coe = [1/100,1,1/8,1];
% Predator configuration
prd = true; prdSpd = 4;
% Plotting configuration
arrows = true; pltFov = false; figNo = 4;
% Function calling for case 4
swarmModel(n,dim,ts,zor,zop,fov,coe,prd,prdSpd,vLim,vCorr,pltFov,arrows,figNo)

%%
% Case 5: repulsion and field of view extra test

% Case variables
dim = 100; ts = 100;
% Bird configuration
zor = 10; zop = 35; fov = 0.2*pi; vLim = 20; vCorr = 5;
coe = [1/100,1/2,1/8,1];
% Predator configuration
prd = false; prdSpd = 1;
% No extra plotting configuration due to internal plotting mechanism
figNo = 5;

% Function calling for case 5
testcase(dim,ts,zor,zop,fov,coe,prd,vLim,vCorr,figNo)
```