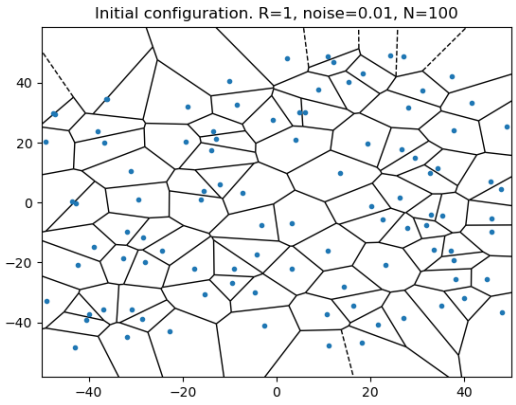
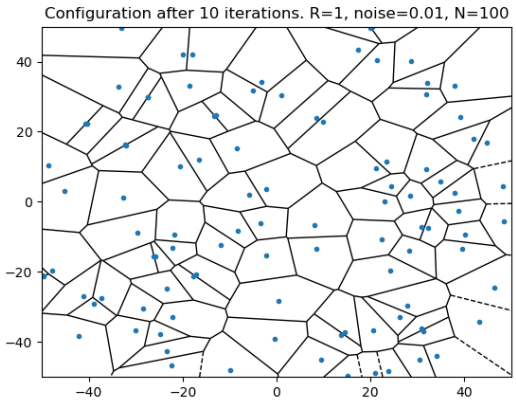
Home work 2 chap.8 Vicsek model

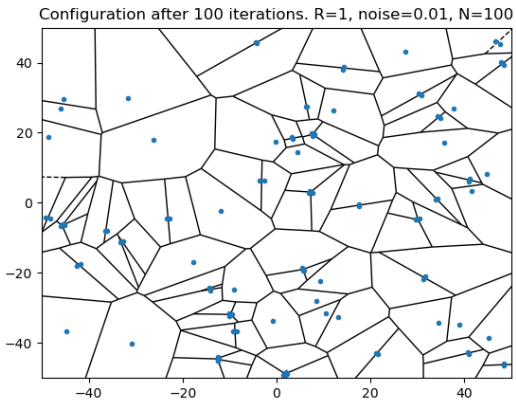
8.4

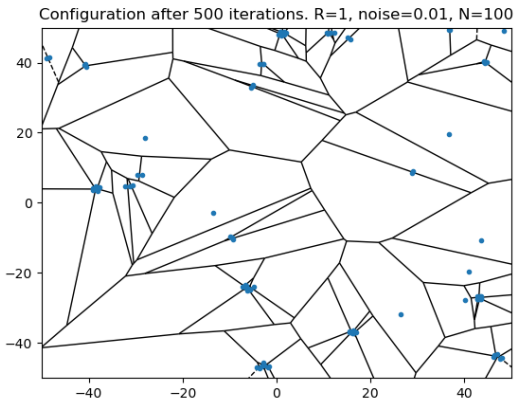
Under I will show all figures that are asked for the different a)-d) but I will in the order for the of the different R to make it easier to compare

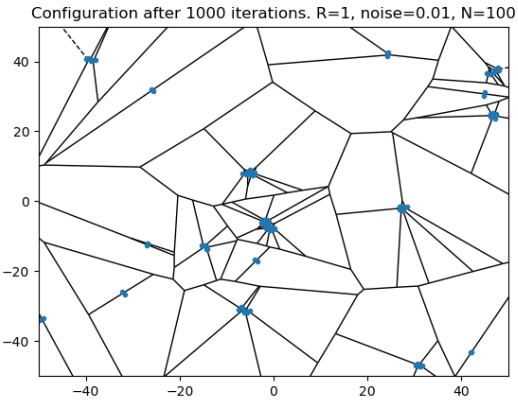
For R=1:

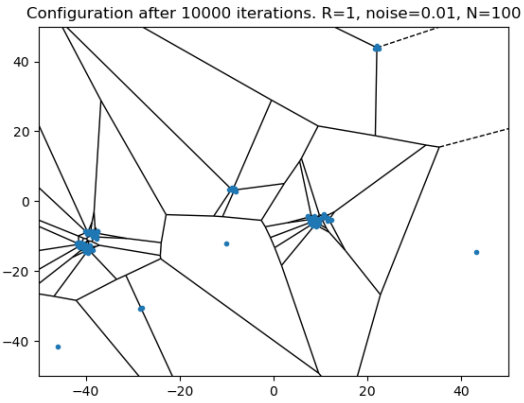


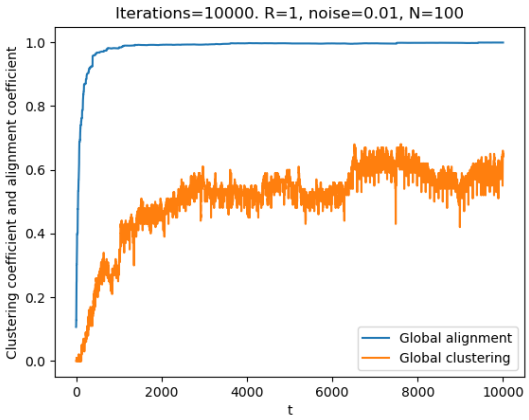




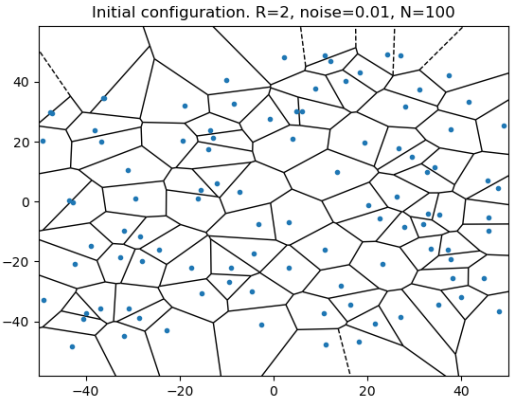


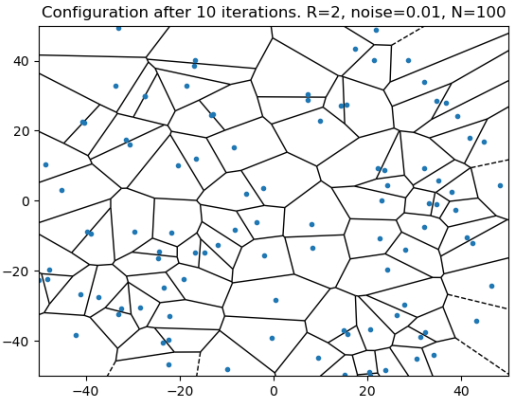

Configuration after 1000 iterations. R=1, noise=0.01, N=100

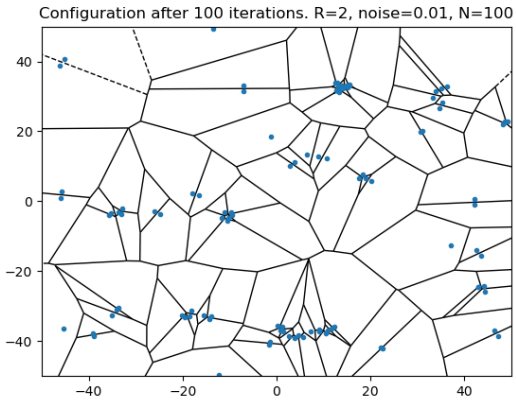

Configuration after 10000 iterations. R=1, noise=0.01, N=100

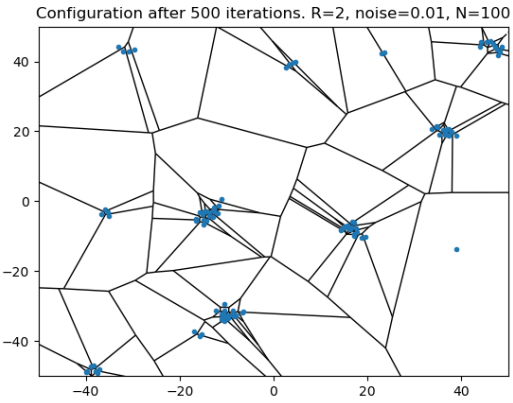

Iterations=10000. R=1, noise=0.01, N=100
Clustering coefficient and alignment coefficient
t
Global alignment
Global clustering

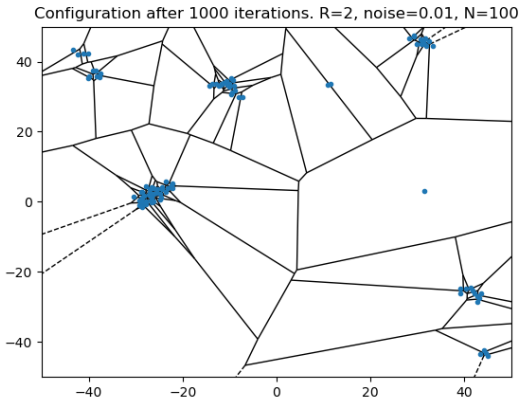
For R=2:

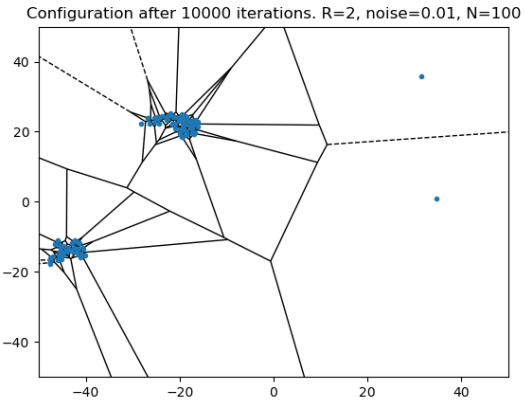

Initial configuration. R=2, noise=0.01, N=100

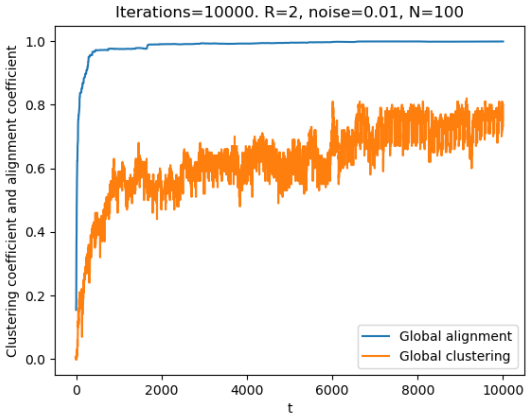

Configuration after 10 iterations. R=2, noise=0.01, N=100


Configuration after 100 iterations. R=2, noise=0.01, N=100

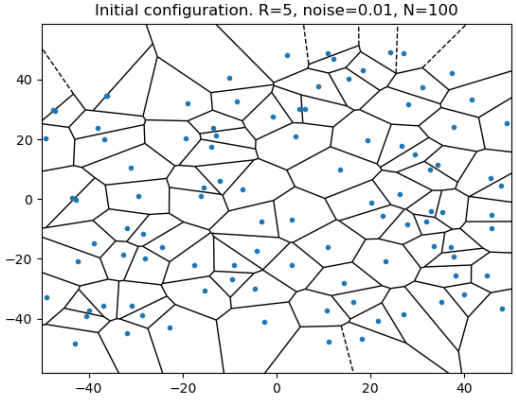

Configuration after 500 iterations. R=2, noise=0.01, N=100

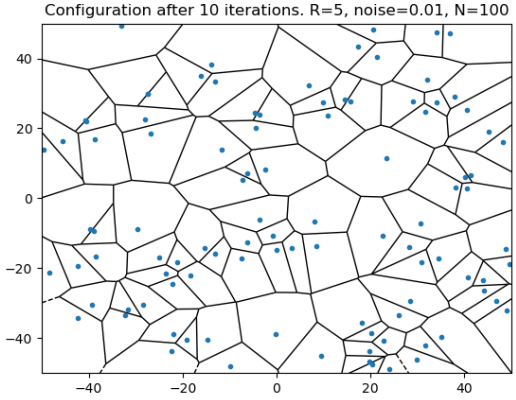

Configuration after 1000 iterations. R=2, noise=0.01, N=100

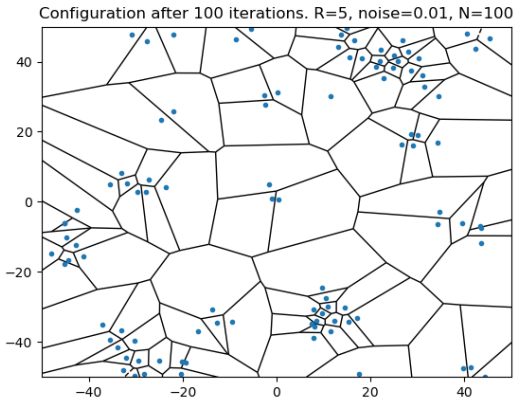

Configuration after 10000 iterations. R=2, noise=0.01, N=100

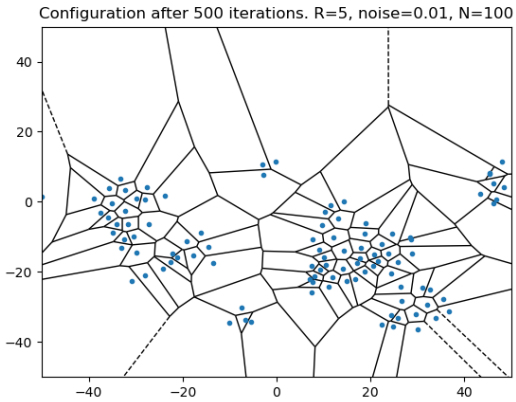

Iterations=10000. R=2, noise=0.01, N=100
Clustering coefficient and alignment coefficient
t
Global alignment
Global clustering

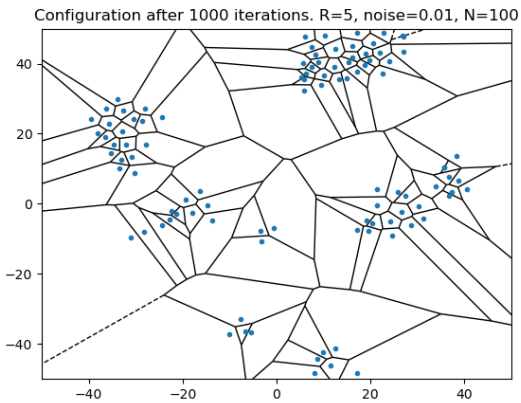
For R=5:

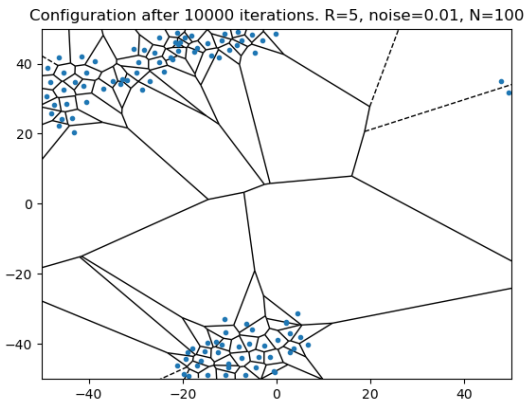

Initial configuration. R=5, noise=0.01, N=100

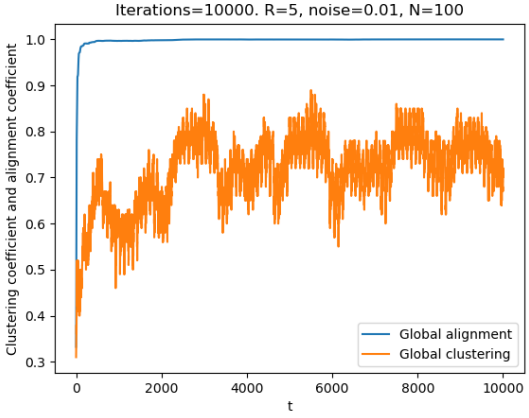

Configuration after 10 iterations. R=5, noise=0.01, N=100


Configuration after 100 iterations. R=5, noise=0.01, N=100

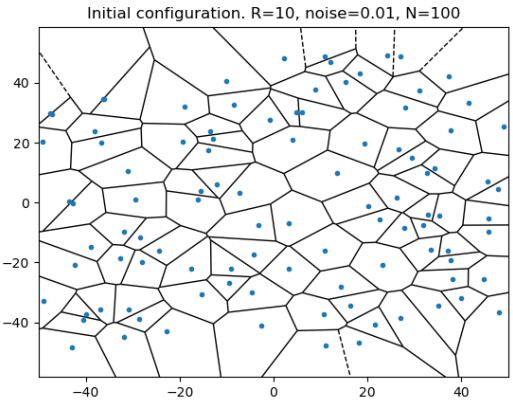

Configuration after 500 iterations. R=5, noise=0.01, N=100

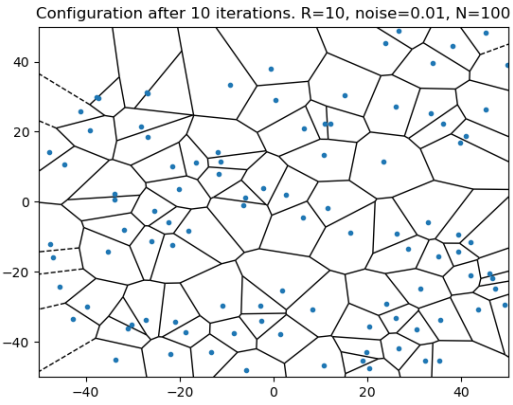

Configuration after 1000 iterations. R=5, noise=0.01, N=100

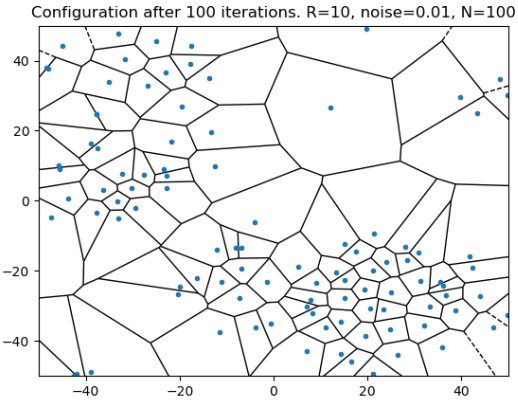

Configuration after 10000 iterations. R=5, noise=0.01, N=100

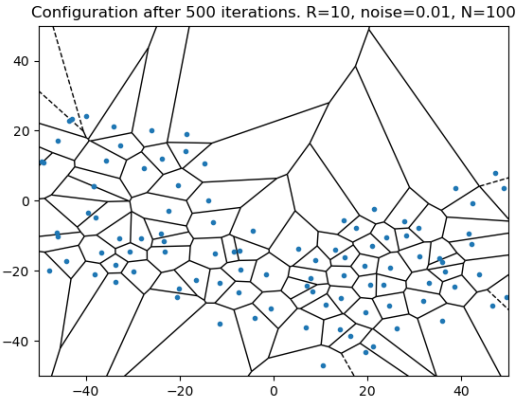

Iterations=10000. R=5, noise=0.01, N=100
Clustering coefficient and alignment coefficient
t
Global alignment
Global clustering

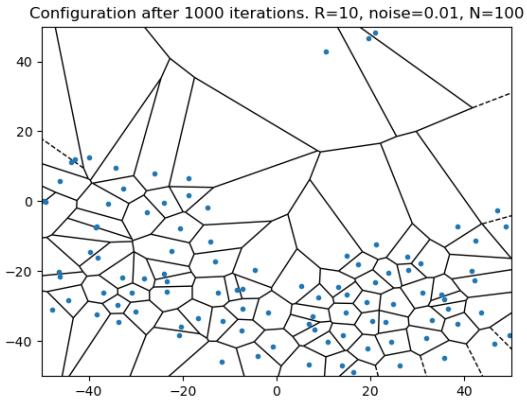
For R=10:

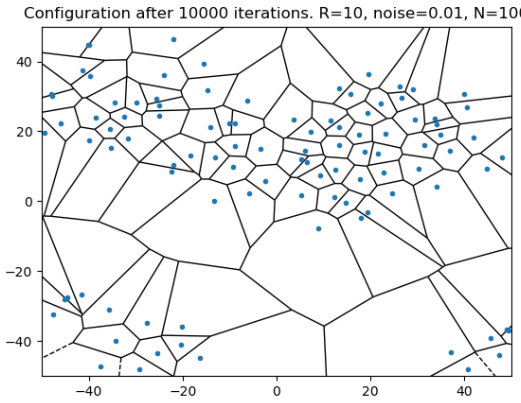

Initial configuration. R=10, noise=0.01, N=100

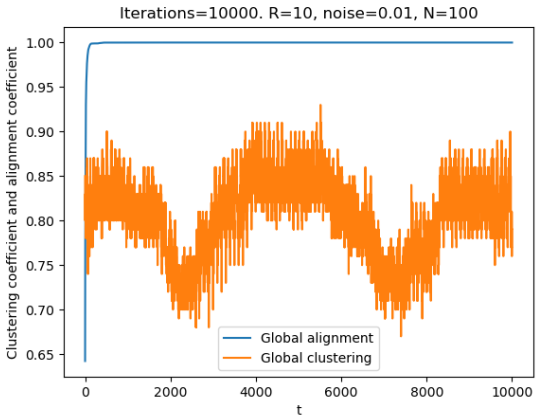

Configuration after 10 iterations. R=10, noise=0.01, N=100


Configuration after 100 iterations. R=10, noise=0.01, N=100


Configuration after 500 iterations. R=10, noise=0.01, N=100



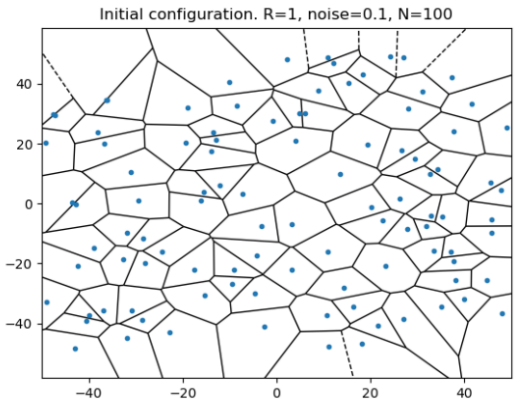


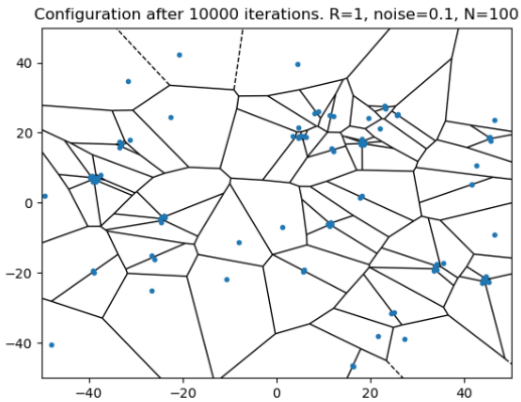


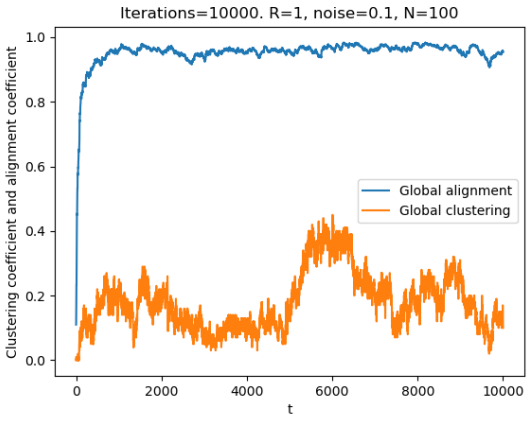
In conclusion we can see from the generated figures that it clusters less for higher R, which is logic since the update of orientations for every particle depends on the particles neighbours which is the other particles that are within a distance R from the consider particle, so bigger R more particles comes into play.

8.5 Increased noise

In this report I will only show the initial and the last iteration because the other plots are kind of irrelevant when we compare.

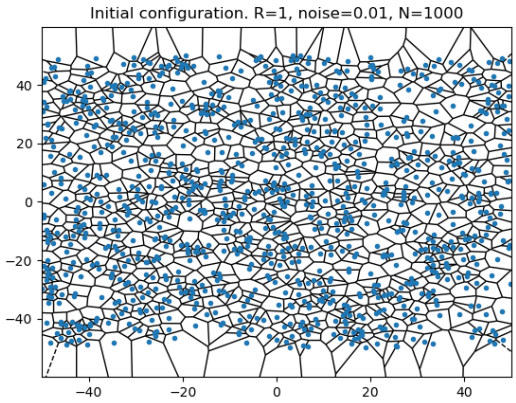


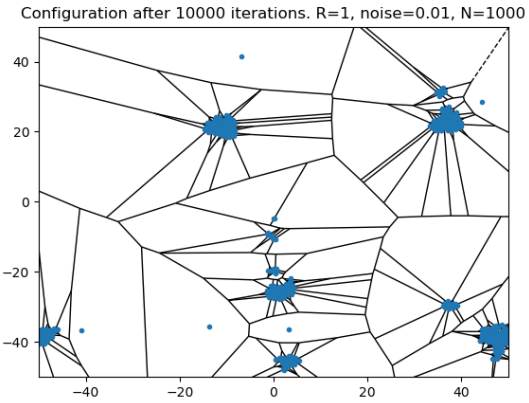


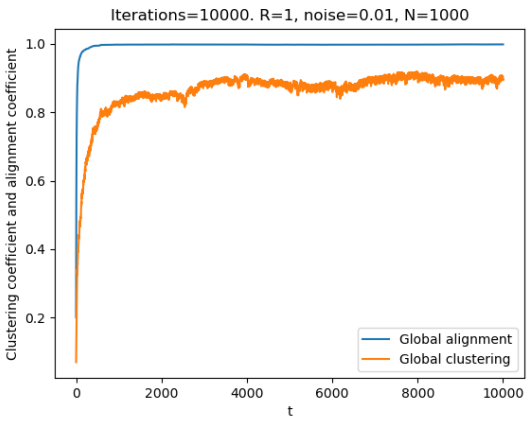


We can see when we compare with the same configuration in 8.4 but with a higher noise level that we get less clustering, there is more spread in the last configuration plot and we can also se on the global clustering coefficient that it converges to a higher value in 8.4 but in this plot it is more chaotic(goes more up and down).

8.6 Higher density

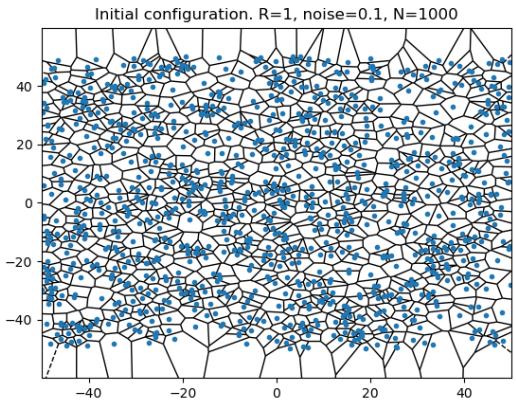


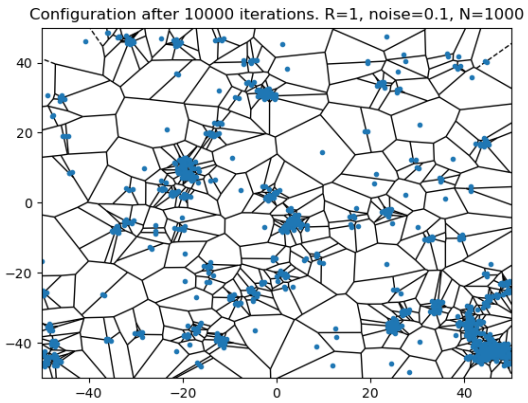


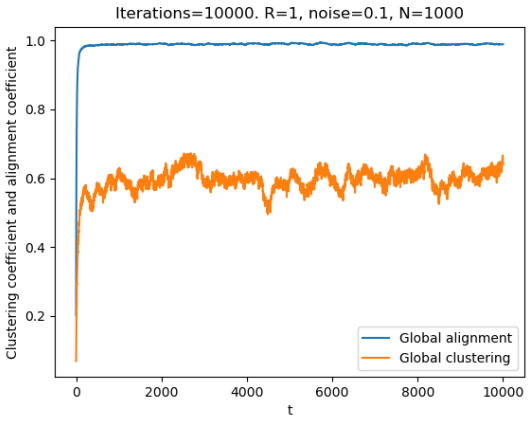


We can see from this that this looks kind of similar with the figure 8.5 in the course book. The global alignment coefficient shows a better and faster alignment and the clustering coefficient do not fluctuate as much as with lower density and it reaches a higher value.

8.7 Higher density and increased noise



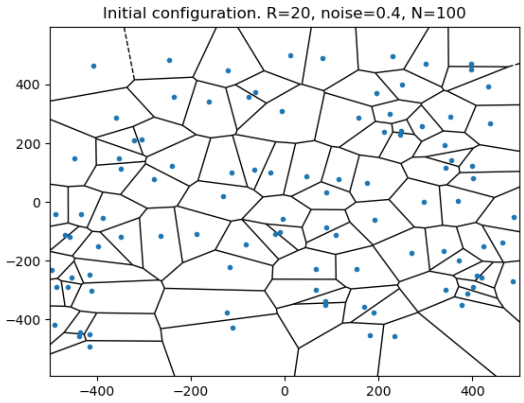


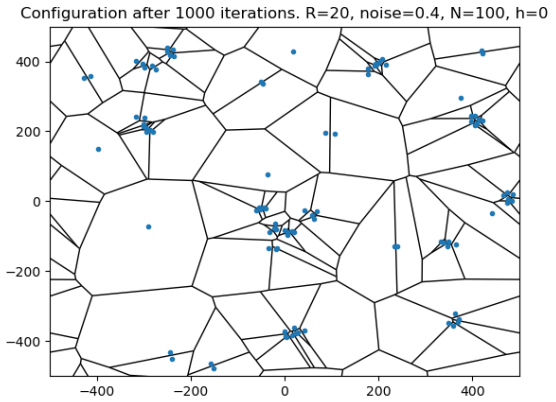


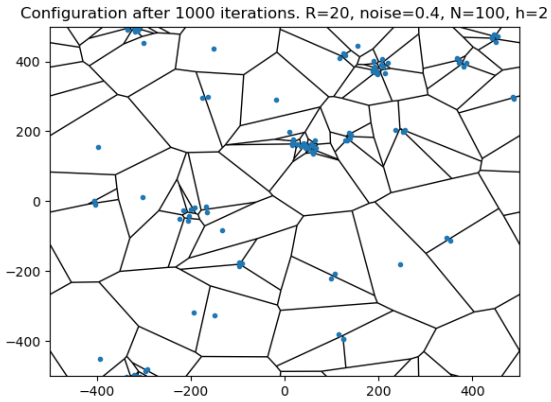
We can see from those figures that they look kind of similar to figure 8.6 in the course book. The alignment do not change so much comparing with task 8.6 when we only increased the density but the clustering coefficient fluctuates more.

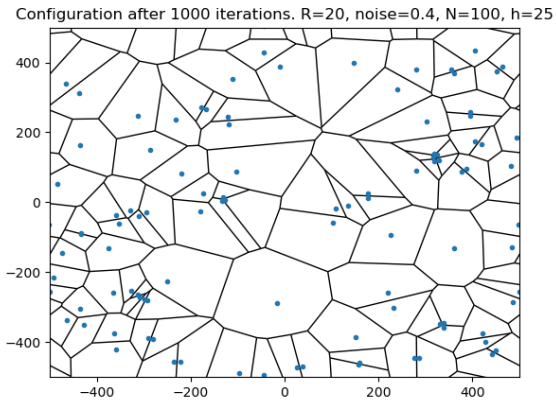
8.8 Delayed vicsek model

a)-b):



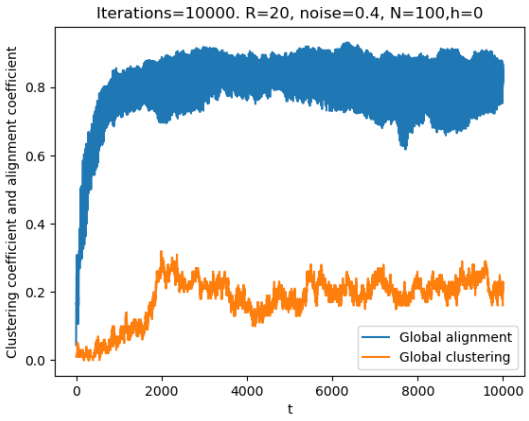


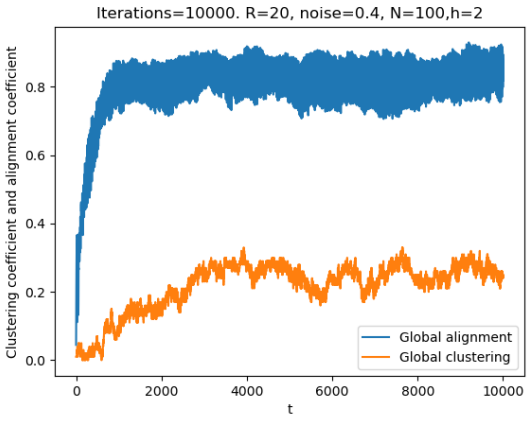


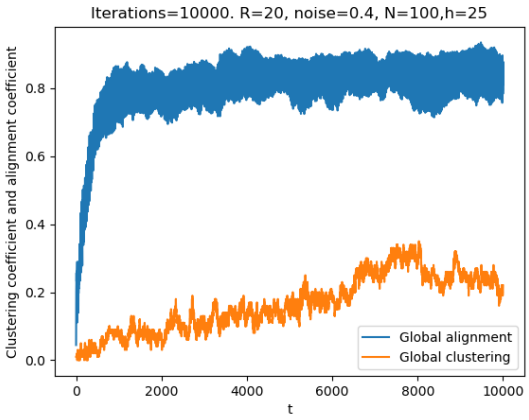


We can see from this that we get the right behaviour. Delay of 2(h=2) cluster most, while the big delay(h=25) gets less clustering.

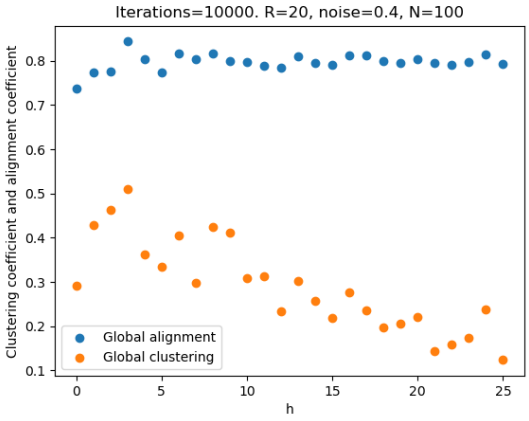
c):





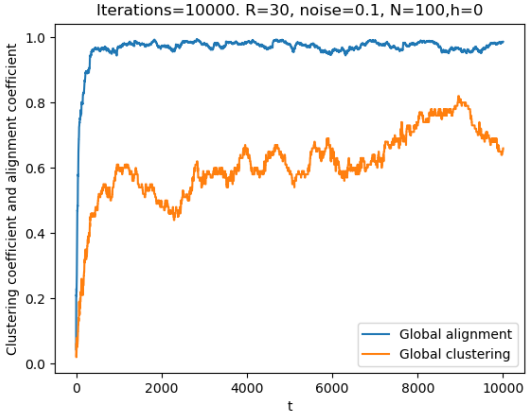


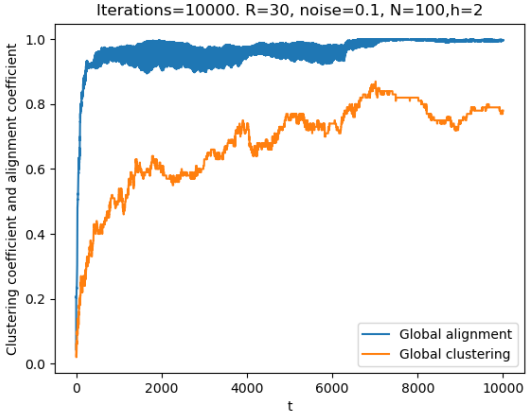
This plots of coefficients as function of time is wrong. All corresponds for different simulations for h=25. It was a bug in my code that overwrote the different coefficients when plotting but the plot below is correct where it shows how the coefficients change due to the different delays(h)

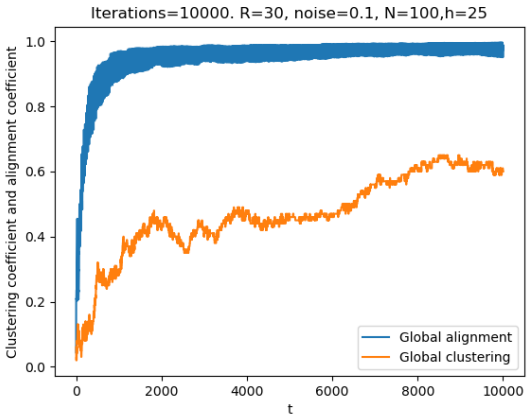


We can see from the last plot which is the average of the coefficients over 10000 iterations for different delays (different h) that it is kind of similar to figure 8.8 in the book. More delays lead to less clustering, but a little delay like h=2 gives little more clustering.

d):



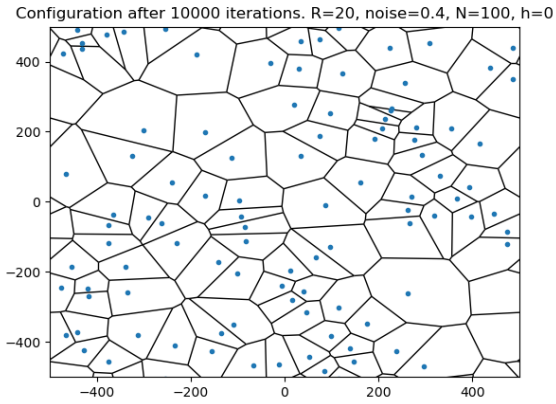


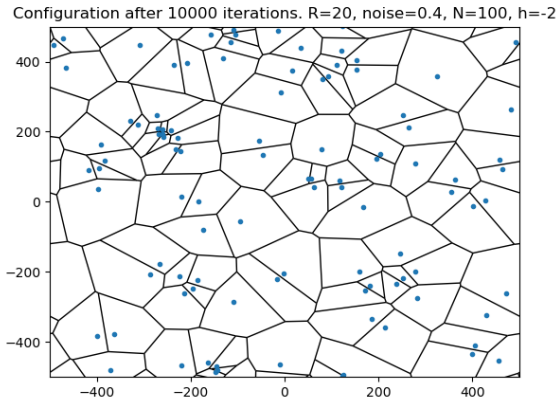


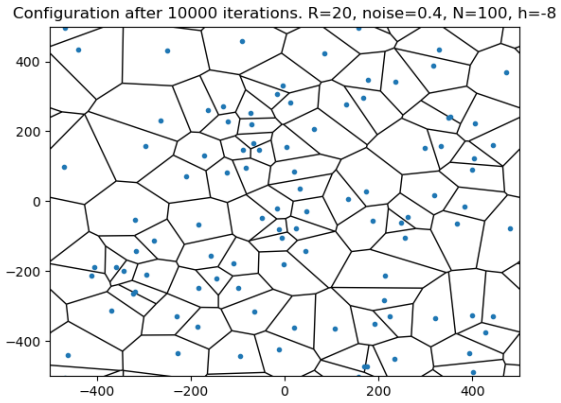
This one is correct. It is for less noise (eta=0.1) and bigger radius R=30. The biggest different between the delays is the alignment coefficient, it is more noisy for delay. We also get less clustering for higher delay

8.9) Negative delays

a)-b):

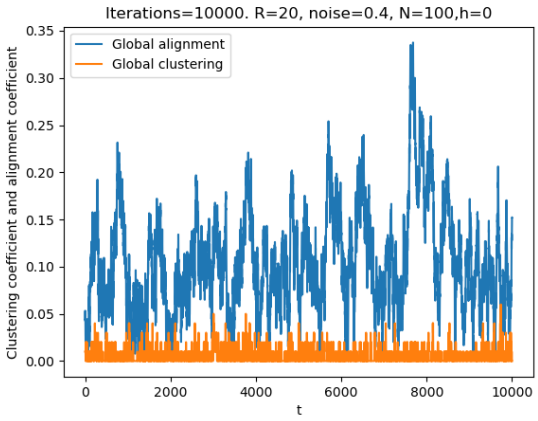


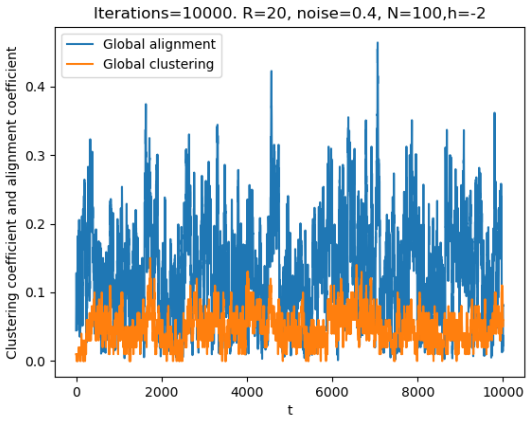


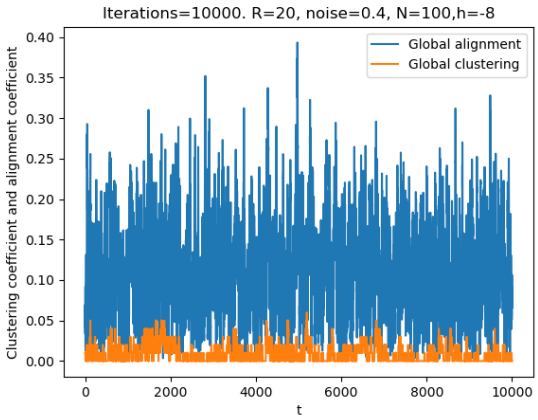


We can see that when h=0(no delay) using the extrapolation procedure, it gets not the same as in the original vicsek model. This one clusters much less. We can also see that for more steps in the future there is less clustering.

C):



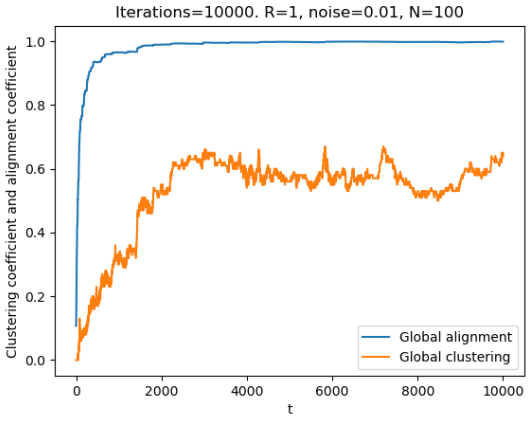




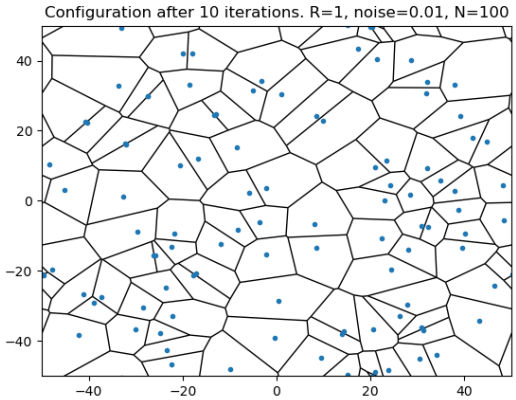
These coefficient plots looks kind of logical. We can see that it clusters less for more steps into the future (h=-8). For h=-8 it both coefficients gets more noisy. Comparing these with figure 8.9 in the course book, they look quite similar.

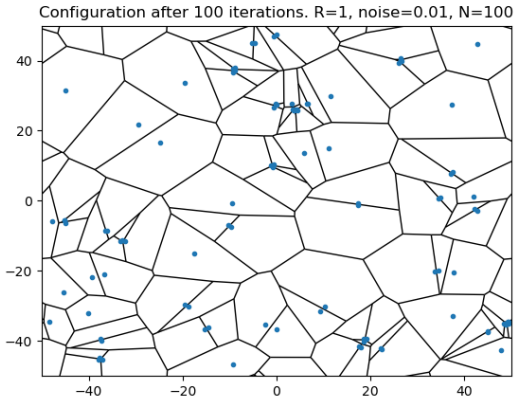
One note is that for the first tasks, 8.4-8.7. I Thought that I did right but there is some errors in the plots when using Voronoi in python, since some off the Voronoi regions(polygons) gets dashes lines which means that the polygon area is infinite. I makes some of the plots in 8.4 for example more noisy than it should be. I succeed to correct this but I will not give all figures again but I can just show what it should look like in 8.4 for instance.

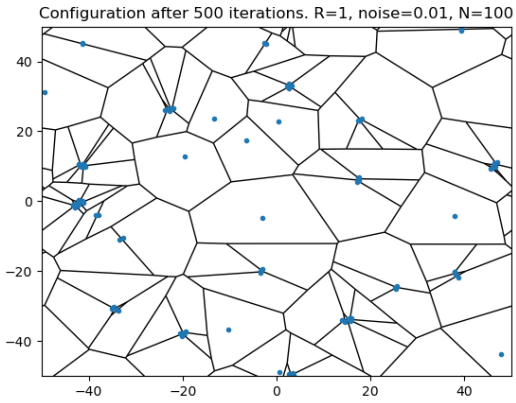
8.4 for R=1:

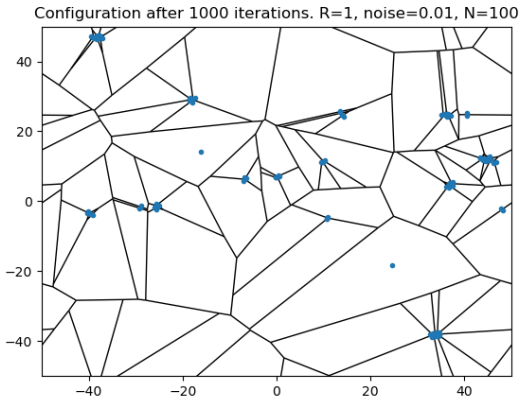


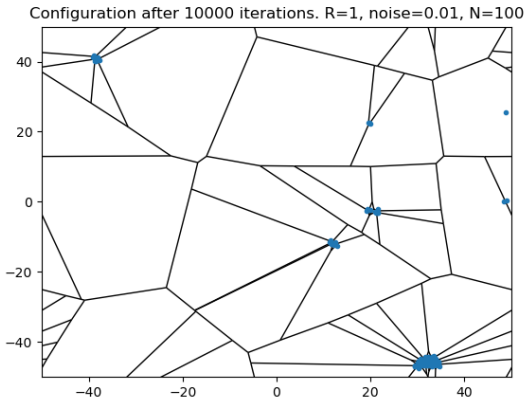
We can see comparing with the 8.4 in the beginning of the file that the clustering coefficient graph is much less noisy which is because of my change in the code that makes all polygons finite in the Voronoi plot, which makes the area calculation correct. Here are the corresponding vicsek models for this:





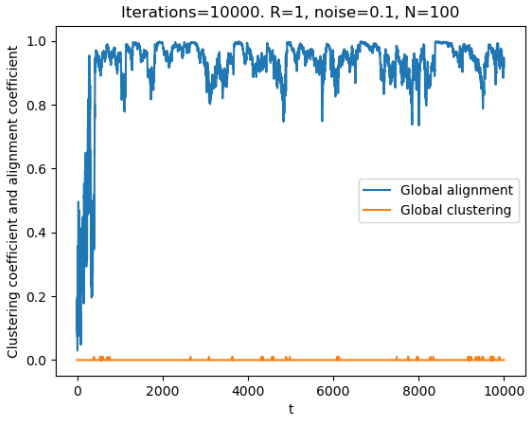






8.10)K-neirest neighbours

I did this but I do not get a great result. There must been some mistake in the code but I do not see where, I got this plot of the coefficients:



It do not cluster which is not what we want for this.

Code

```
    import numpy as np
from scipy.spatial import Voronoi, voronoi_plot_2d
import matplotlib.pyplot as plt
PI = np.pi


def generate_positions_orientations(N, L, disp_factor=None):
    positions = np.random.rand(N, 2) * L - L/2
    orientations = np.random.rand(N) * 2*PI
    if disp_factor:
        positions[:, 0] = positions[:, 0] + disp_factor
        positions[:, 1] = positions[:, 1] + disp_factor
    positions = particle_boundary(positions, L)
    return [positions, orientations]

def particle_boundary(positions, L):
    for i in range(len(positions)):
        x = positions[i, 0]
        y = positions[i, 1]

        if x < -L/2:
            positions[i, 0] = x + L
        elif x > L/2:
            positions[i, 0] = x - L

        if y < - L/2:
            positions[i, 1] = y + L
        elif y > L/2:
            positions[i, 1] = y - L

    return positions

def particle_within_radius(positions, R, L):
    N = len(positions)
    index_list = []
    for i in range(N):
        x_distance = np.abs(positions[i, 0] - positions[:, 0])
        y_distance = np.abs(positions[i, 1] - positions[:, 1])
        min_x_distance = np.minimum(x_distance, L-x_distance)
        min_y_distance = np.minimum(y_distance, L - y_distance)
        distance = np.sqrt(min_x_distance**2 + min_y_distance**2)
        indexes = np.where(distance < R)
        index_list.append(indexes)
    return index_list

def get_velocities(orientations, v=1):
    N = len(orientations)
```

```
    velocities = np.zeros((N, 2))
    for i in range(N):
        theta = orientations[i]
        velocities[i, 0] = v * np.cos(theta)
        velocities[i, 1] = v * np.sin(theta)
    return velocities

def global_alignment(velocities, v=1):
    N = len(velocities)
    sum = np.sum(velocities, axis=0)
    global_alignment_fac = np.linalg.norm(sum) / (v*N)
    return global_alignment_fac


def global_clustering(positions, R, L):
    N = len(positions)
    clone = cloning(positions, L)
    expand_positions = np.copy(positions)
    for i in range(len(clone)):
        expand_positions = np.append(expand_positions, clone[i], axis=0)

    vor = Voronoi(expand_positions)
    counter = 0
    for i in range(N):
        polygon_index = vor.point_region[i]
        index_of_corners = vor.regions[polygon_index]
        polygon_corners = vor.vertices[index_of_corners]
        area = PolyArea(polygon_corners)
        if area < (PI * R**2):
            counter += 1
    clustering_coeff = counter / N
    return clustering_coeff


def PolyArea(corners):
    x = corners[:, 0]
    y = corners[:, 1]
    return 0.5*np.abs(np.dot(x,np.roll(y,1))-np.dot(y,np.roll(x,1)))


def update_orientations(positions, orientations, eta, delta_t, R, L):
    N = len(orientations)
    W = np.random.uniform(-1/2, 1/2, N)
    particles_with_neighbours = particle_within_radius(positions, R, L)
    new_orientation = np.zeros(N)
    for i in range(N):
        neighbours = particles_with_neighbours[i]
        neighbours_orientation = orientations[neighbours]
```

```
        if len(neighbours_orientation) == 1:
            average_orientation = neighbours_orientation
        else:
            average_orientation = np.arctan(np.mean(np.sin(neighbours_orientation))/np.mean(np.co
        new_orientation[i] = average_orientation + eta * W[i] * delta_t
    return new_orientation

def update_positions(positions, velocities, delta_t, L):
    new_positions = positions + velocities*delta_t
    new_positions = particle_boundary(new_positions, L)
    return new_positions



def cloning(positions, L):
    clone_right = np.copy(positions)
    clone_right[:, 0] = clone_right[:, 0] + L

    clone_left = np.copy(positions)
    clone_left[:, 0] = clone_left[:, 0] - L

    clone_down = np.copy(positions)
    clone_down[:, 1] = clone_down[:, 1] - L

    clone_up = np.copy(positions)
    clone_up[:, 1] = clone_up[:, 1] + L

    clone_diagonal_1 = np.copy(positions)
    clone_diagonal_1[:, :] = clone_diagonal_1[:, :] + L

    clone_diagonal_2 = np.copy(positions)
    clone_diagonal_2[:, :] = clone_diagonal_2[:, :] - L

    clone_diagonal_3 = np.copy(positions)
    clone_diagonal_3[:, 0] = clone_diagonal_3[:, 0] + L
    clone_diagonal_3[:, 1] = clone_diagonal_3[:, 1] - L

    clone_diagonal_4 = np.copy(positions)
    clone_diagonal_4[:, 0] = clone_diagonal_4[:, 0] - L
    clone_diagonal_4[:, 1] = clone_diagonal_4[:, 1] + L

    clone_positions = [clone_down, clone_up, clone_right, clone_left,
             clone_diagonal_1, clone_diagonal_2, clone_diagonal_3, clone_diagonal_4]
    return clone_positions


def plot_vicsek_model(positions, L):
    clone = cloning(positions, L)
```

```
    expand_positions = np.copy(positions)

    for j in range(len(clone)):
        expand_positions = np.append(expand_positions, clone[j], axis=0)

    vor = Voronoi(expand_positions)
    voronoi_plot_2d(vor, show_vertices=False)
    plt.xlim([-L/2, L/2])
    plt.ylim([-L/2, L/2])


def particle_within_radius_V2(positions, R, L):
    N = len(positions)
    index_list = []
    for i in range(N):
        x_distance = np.abs(positions[i, 0] - positions[:, 0])
        y_distance = np.abs(positions[i, 1] - positions[:, 1])
        min_x_distance = np.minimum(x_distance, L-x_distance)
        min_y_distance = np.minimum(y_distance, L - y_distance)
        distance = np.sqrt(min_x_distance**2 + min_y_distance**2)
        indexes = np.where(distance < R)
        indexes = indexes[0][0]
        index_list.append(indexes)
    return index_list



def update_orientations_V2(orientations, neighbours_index, eta, delta_t, R, L):
    N = len(orientations)
    W = np.random.uniform(-1/2, 1/2, N)
    # particles_with_neighbours = particle_within_radius(positions, R, L)
    new_orientation = np.zeros(N)
    for i in range(N):
        # neighbours = particles_with_neighbours[i]
        neighbours = neighbours_index[i]
        neighbours_orientation = orientations[neighbours]
        neighbours_orientation = [neighbours_orientation]
        if len(neighbours_orientation) == 1:
            average_orientation = neighbours_orientation
        else:
            average_orientation = np.arctan(np.mean(np.sin(neighbours_orientation))/np.mean(np.c
        new_orientation[i] = average_orientation + eta * W[i] * delta_t
    return new_orientation


def particle_within_radius_k(positions, R, L, k):
    N = len(positions)
    index_list = []
```

```
    for i in range(N):
        x_distance = np.abs(positions[i, 0] - positions[:, 0])
        y_distance = np.abs(positions[i, 1] - positions[:, 1])
        min_x_distance = np.minimum(x_distance, L-x_distance)
        min_y_distance = np.minimum(y_distance, L - y_distance)
        distance = np.sqrt(min_x_distance**2 + min_y_distance**2)
        sort_distance = np.sort(distance)
        sort_distance = sort_distance[0:k+2]
        for dist in sort_distance:
            indexes = np.where(distance == dist)
            index_list.append(indexes)
    return index_list

def update_orientations_k(positions, orientations, eta, delta_t, R, L, k):
    N = len(orientations)
    W = np.random.uniform(-1/2, 1/2, N)
    particles_with_neighbours = particle_within_radius_k(positions, R, L, k)
    new_orientation = np.zeros(N)
    for i in range(N):
        neighbours = particles_with_neighbours[i]
        neighbours_orientation = orientations[neighbours]
        if len(neighbours_orientation) == 1:
            average_orientation = neighbours_orientation
        else:
            average_orientation = np.arctan(np.mean(np.sin(neighbours_orientation))/np.mean(np.co
        new_orientation[i] = average_orientation + eta * W[i] * delta_t
    return new_orientation

    import matplotlib.pyplot as plt
import numpy as np
import vicsek_model_funs as vicsek
from scipy.spatial import Voronoi, voronoi_plot_2d

# Parameters
L = 100
N = 100
v = 1
delta_t = 1
eta = 0.01
iterations = 1000
R = 1


positions = np.load('initial_positions.npy')
orientations = np.load('initial_orientations.npy')
# vor = Voronoi(positions)
# fig1 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-50, 50])
# plt.title(f'Initial configuration. R={R}, noise={eta}, N={N}')
```

```
vicsek.plot_vicsek_model(positions, L)


global_alignment_coeffs = np.zeros(iterations)
clustering_coeffs = np.zeros(iterations)
times = np.arange(iterations)
for i in range(iterations):
    orientations = vicsek.update_orientations(positions, orientations, eta, delta_t, R, L)
    velocities = vicsek.get_velocities(orientations, v)
    positions = vicsek.update_positions(positions, velocities, delta_t, L)
    global_alignment_coeffs[i] = vicsek.global_alignment(velocities, v)
    clustering_coeffs[i] = vicsek.global_clustering(positions, R, L)

    if i == 9:
        positions_10 = positions
    if i == 99:
        positions_100 = positions
    if i == 499:
        positions_500 = positions
    if i == 999:
        positions_1000 = positions


# vor = Voronoi(positions)
# fig2 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {iterations} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_10)
# fig3 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {10} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_100)
# fig4 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {100} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_500)
# fig4 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {500} iterations. R={R}, noise={eta}, N={N}')
#
#
```

```
# vor = Voronoi(positions_1000)
# fig5 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}')

vicsek.plot_vicsek_model(positions, L)



fig, ax = plt.subplots()
ax.plot(times, global_alignment_coeffs, label='Global alignment')
ax.plot(times, clustering_coeffs, label='Global clustering')
leg = ax.legend()
plt.xlabel('t')
plt.ylabel('Clustering coefficient and alignment coefficient')
plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N}')
plt.show()


    import matplotlib.pyplot as plt
import numpy as np
import vicsek_model_funs as vicsek
from scipy.spatial import Voronoi, voronoi_plot_2d
from tqdm import trange
# Parameters
L = 1000
N = 100
v = 3
delta_t = 1
eta = 0.4
iterations = 10000
R = 20
# H = np.arange(26)
H = [0, 2, 25]




positions = np.load('initial_positions_8.8.npy')
orientations = np.load('initial_orientations_8.8.npy')

vor = Voronoi(positions)
fig1 = voronoi_plot_2d(vor, show_vertices=False)
plt.xlim([-L / 2, L / 2])
plt.title(f'Initial configuration. R={R}, noise={eta}, N={N}')

global_alignment_coeffs = np.zeros(iterations)
clustering_coeffs = np.zeros(iterations)
```

```
times = np.arange(iterations)
positions_1000 = []
positions_10000 = []
mean_clusterings = []
mean_alignments = []
clustering_coeffs_list = []
aligment_coeffs_list = []
for h in H:
    positions_list = []
    old_orientations = []
    positions = np.load('initial_positions_8.8.npy')
    orientations = np.load('initial_orientations_8.8.npy')
    old_orientations = [orientations]
    velocities = vicsek.get_velocities(orientations, v)
    positions_list = [positions]
    for i in trange(iterations):
        if i > h > 0:
            global_alignment_coeffs[i] = vicsek.global_alignment(velocities, v)
            clustering_coeffs[i] = vicsek.global_clustering(positions, R, L)
            orientations = vicsek.update_orientations(positions_list[i-h], old_orientations[i-h]
            velocities = vicsek.get_velocities(old_orientations[i-h], v)
            positions = vicsek.update_positions(positions, velocities, delta_t, L)

        else:
            global_alignment_coeffs[i] = vicsek.global_alignment(velocities, v)
            clustering_coeffs[i] = vicsek.global_clustering(positions, R, L)
            orientations = vicsek.update_orientations(positions, orientations, eta, delta_t, R, I
            velocities = vicsek.get_velocities(orientations, v)
            positions = vicsek.update_positions(positions, velocities, delta_t, L)

        if h == 0:
            clustering_coeffs_list.append(clustering_coeffs)
            aligment_coeffs_list.append(global_alignment_coeffs)

        elif h == 2:
            clustering_coeffs_list.append(clustering_coeffs)
            aligment_coeffs_list.append(global_alignment_coeffs)

        elif h == 25:
            clustering_coeffs_list.append(clustering_coeffs)
            aligment_coeffs_list.append(global_alignment_coeffs)


        old_orientations.append(orientations)
        positions_list.append(positions)


        if i == 999:
```

```
            positions_1000.append(positions)
        if i == 9999:
            positions_10000.append(positions)

    mean_clustering = np.mean(clustering_coeffs)
    mean_alignment = np.mean(global_alignment_coeffs)
    mean_clusterings.append(mean_clustering)
    mean_alignments.append(mean_alignment)


# 10 000
# vicsek.plot_vicsek_model(positions_10000[0], L)
# plt.title(f'Configuration after {10000} iterations. R={R}, noise={eta}, N={N}, h={0}')
#
# vicsek.plot_vicsek_model(positions_10000[2], L)
# plt.title(f'Configuration after {10000} iterations. R={R}, noise={eta}, N={N}, h={2}')
#
# vicsek.plot_vicsek_model(positions_10000[25], L)
# plt.title(f'Configuration after {10000} iterations. R={R}, noise={eta}, N={N}, h={25}')

# 1000
# vicsek.plot_vicsek_model(positions_1000[0], L)
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}, h={0}')
#
# vicsek.plot_vicsek_model(positions_1000[2], L)
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}, h={2}')
#
# vicsek.plot_vicsek_model(positions_1000[25], L)
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}, h={25}')

# coefficients as function of time
# fig1, ax1 = plt.subplots()
# ax1.plot(times, aligment_coeffs_list[0], label='Global alignment')
# ax1.plot(times, clustering_coeffs_list[0], label='Global clustering')
# leg = ax1.legend()
# plt.xlabel('t')
# plt.ylabel('Clustering coefficient and alignment coefficient')
# plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N},h={0}')

# fig, ax = plt.subplots()
# ax.plot(times, aligment_coeffs_list[1], label='Global alignment')
# ax.plot(times, clustering_coeffs_list[1], label='Global clustering')
# leg = ax.legend()
# plt.xlabel('t')
# plt.ylabel('Clustering coefficient and alignment coefficient')
# plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N},h={2}')
#
fig, ax = plt.subplots()
```

```
ax.plot(times, aligment_coeffs_list[2], label='Global alignment')
ax.plot(times, clustering_coeffs_list[2], label='Global clustering')
leg = ax.legend()
plt.xlabel('t')
plt.ylabel('Clustering coefficient and alignment coefficient')
plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N},h={25}')


# coefficients as fun of h
fig, ax = plt.subplots()
ax.scatter(H, mean_alignments, label='Global alignment')
ax.scatter(H, mean_clusterings, label='Global clustering')
leg = ax.legend()
plt.xlabel('h')
plt.ylabel('Clustering coefficient and alignment coefficient')
plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N}')
plt.show()


import matplotlib.pyplot as plt
import numpy as np
import vicsek_model_funs as vicsek
from scipy.spatial import Voronoi, voronoi_plot_2d
from tqdm import trange
# Parameters
L = 1000
N = 100
v = 3
delta_t = 1
eta = 0.4
iterations = 10000
R = 20
h = -2




positions = np.load('initial_positions_8.8.npy')
orientations = np.load('initial_orientations_8.8.npy')

vor = Voronoi(positions)
fig1 = voronoi_plot_2d(vor, show_vertices=False)
plt.xlim([-L / 2, L / 2])
plt.title(f'Initial configuration. R={R}, noise={eta}, N={N}')

global_alignment_coeffs = np.zeros(iterations)
clustering_coeffs = np.zeros(iterations)
times = np.arange(iterations)
positions_1000 = []
```

```
positions_10000 = []
positions = np.load('initial_positions_8.8.npy')
orientations = np.load('initial_orientations_8.8.npy')
velocities = vicsek.get_velocities(orientations, v)
neighbours_index = vicsek.particle_within_radius_V2(positions, R, L)

for i in trange(iterations):

    global_alignment_coeffs[i] = vicsek.global_alignment(velocities, v)
    clustering_coeffs[i] = vicsek.global_clustering(positions, R, L)
    clone_neighbours_index = np.copy(neighbours_index)
    clone_positions = np.copy(positions)
    clone_velocities = np.copy(velocities)

    for j in range(-h):
        clone_neighbours_index = vicsek.particle_within_radius_V2(clone_positions, R, L)
        orientations = vicsek.update_orientations_V2(orientations, clone_neighbours_index, eta, c
        clone_velocities = vicsek.get_velocities(orientations, v)
        clone_positions = vicsek.update_positions(clone_positions, clone_velocities, delta_t, L)

    clone_neighbours_index = vicsek.particle_within_radius_V2(clone_positions, R, L)
    orientations = vicsek.update_orientations_V2(orientations, neighbours_index, eta, delta_t, R
    velocities = vicsek.get_velocities(orientations, v)
    positions = vicsek.update_positions(positions, velocities, delta_t, L)


    if i == 999:
        positions_1000.append(positions)
    if i == 9999:
        positions_10000.append(positions)







# 10 000
vicsek.plot_vicsek_model(positions_10000[0], L)
plt.title(f'Configuration after {10000} iterations. R={R}, noise={eta}, N={N}, h={h}')




# 1000
# vicsek.plot_vicsek_model(positions_1000[0], L)
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}, h={-2}')
```

```
# coefficients as function of time
fig1, ax1 = plt.subplots()
ax1.plot(times, global_alignment_coeffs, label='Global alignment')
ax1.plot(times, clustering_coeffs, label='Global clustering')
leg = ax1.legend()
plt.xlabel('t')
plt.ylabel('Clustering coefficient and alignment coefficient')
plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N},h={h}')
plt.show()
```

```
    import matplotlib.pyplot as plt
import numpy as np
import vicsek_model_funs as vicsek
from tqdm import trange

# Parameters
L = 100
N = 100
v = 1
delta_t = 1
eta = 0.1
iterations = 10000
R = 1
k = 8

positions = np.load('initial_positions.npy')
orientations = np.load('initial_orientations.npy')
# vor = Voronoi(positions)
# fig1 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-50, 50])
# plt.title(f'Initial configuration. R={R}, noise={eta}, N={N}')
vicsek.plot_vicsek_model(positions, L)


global_alignment_coeffs = np.zeros(iterations)
clustering_coeffs = np.zeros(iterations)
times = np.arange(iterations)
for i in trange(iterations):
    orientations = vicsek.update_orientations_k(positions, orientations, eta, delta_t, R, L, k)
    velocities = vicsek.get_velocities(orientations, v)
    positions = vicsek.update_positions(positions, velocities, delta_t, L)
    global_alignment_coeffs[i] = vicsek.global_alignment(velocities, v)
    clustering_coeffs[i] = vicsek.global_clustering(positions, R, L)
```

```
    if i == 9:
        positions_10 = positions
    if i == 99:
        positions_100 = positions
    if i == 499:
        positions_500 = positions
    if i == 999:
        positions_1000 = positions


# vor = Voronoi(positions)
# fig2 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {iterations} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_10)
# fig3 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {10} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_100)
# fig4 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {100} iterations. R={R}, noise={eta}, N={N}')
#
# vor = Voronoi(positions_500)
# fig4 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {500} iterations. R={R}, noise={eta}, N={N}')
#
#
# vor = Voronoi(positions_1000)
# fig5 = voronoi_plot_2d(vor, show_vertices=False)
# plt.xlim([-L/2, L/2])
# plt.ylim([-L/2, L/2])
# plt.title(f'Configuration after {1000} iterations. R={R}, noise={eta}, N={N}')

vicsek.plot_vicsek_model(positions, L)
plt.title(f'Configuration after {iterations} iterations. R={R}, noise={eta}, N={N}, k={k}')



fig, ax = plt.subplots()
ax.plot(times, global_alignment_coeffs, label='Global alignment')
```

```
ax.plot(times, clustering_coeffs, label='Global clustering')
leg = ax.legend()
plt.xlabel('t')
plt.ylabel('Clustering coefficient and alignment coefficient')
plt.title(f'Iterations={iterations}. R={R}, noise={eta}, N={N}')
plt.show()
```