Семинар 16

1. Модель сегрегации Шеллинга

Модель Шеллинга была предложена томасом Шеллингом в 1969 году для описания и моделирования расовой сегрегации. Шеллинг показал, что в равномерно перемешанном сообществе наличие хотя бы небольшого перевеса представителей одной расы (этноса, класса) приводит к очень быстрому агрегированию сообщества.

2. Модель

Предположим, что в некотором сообществе проживают агенты двух типов. Сообщество расположено в квадрате единичной площади. Координаты агента, таким образом, $0 \leqslant x,\ y \leqslant 1$.

Агент считается «счастливым», если не менее половины из его 10 ближайших соседей одного с ним типа. Расстояние определяется как евклидово.

При этом агент не против проживания в смешанном сообществе: если ровно половина его ближайших соседей другого типа, то он считается счастливым.

Если агент счастлив, то он не двигается; если же он несчастлив, то он случайным образом выбирает новые координаты и переезжает в них при условии, что в них он будет счастлив. Цикл продолжается, пока есть желающие переехать.

3. Эксперимент

```
from random import uniform, seed
from math import sqrt

import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = (11, 5)
```

```
class Agent:
    def __init__(self, type):
        self.type = type
        self.draw_location()

    def draw_location(self):
```

```
        self.location = uniform(0, 1), uniform(0, 1)

    def get_distance(self, other):
        a = (self.location[0] - other.location[0]) ** 2
        b = (self.location[1] - other.location[1]) ** 2
        return sqrt(a + b)

    def happy(self, agents, num_neighbours=10, require_same=5):
        distances = []
        for agent in agents:
            if self != agent:
                distance = self.get_distance(agent)
                distances.append((distance, agent))
        distances.sort()
        neighbours = [agent for _, agent in distances[:
↪num_neighbours]]
        num_same_type = sum(self.type == agent.type for agent in␣
↪neighbours)
        return num_same_type >= require_same

    def update(self, agents, num_neighbours=10, require_same=5):
        while not self.happy(agents, num_neighbours,␣
↪require_same):
            self.draw_location()
```

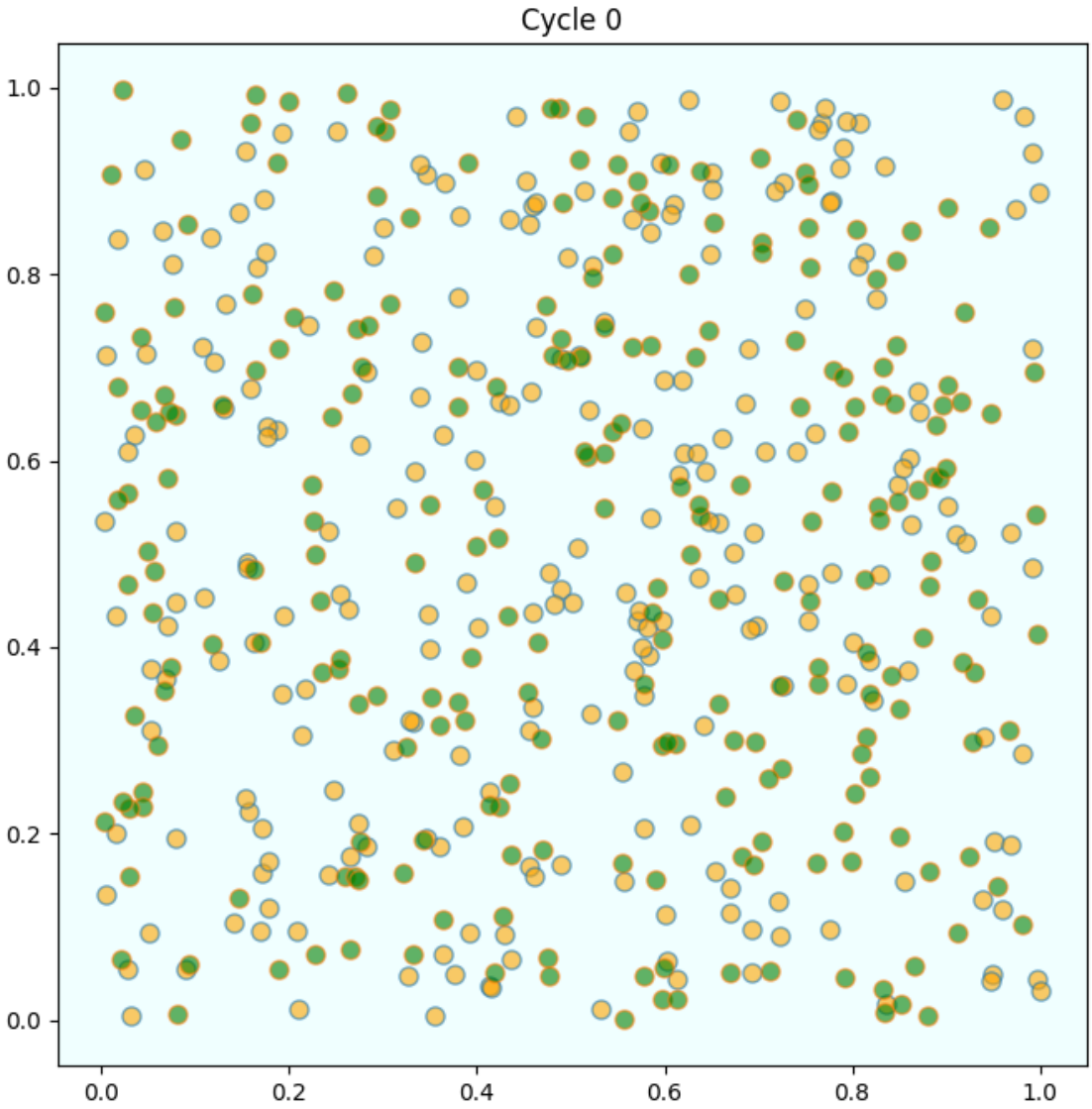
```
def plot_distribution(agents, cycle_num):
    "Plot the distribution of agents after cycle_num rounds of␣
↪the loop."
    x_values_0, y_values_0 = [], []
    x_values_1, y_values_1 = [], []

    for agent in agents:
        x, y = agent.location
        if agent.type == 0:
            x_values_0.append(x)
            y_values_0.append(y)
        else:
            x_values_1.append(x)
            y_values_1.append(y)

    plot_args = {"markersize": 8, "alpha": 0.6}

    fig, (ax) = plt.subplots(figsize=(8, 8))
    ax.set_facecolor("azure")
    ax.plot(x_values_0, y_values_0, "o",␣
↪markerfacecolor="orange", **plot_args)
    ax.plot(x_values_1, y_values_1, "o",␣
↪markerfacecolor="green", **plot_args)
    ax.set_title(f"Cycle {cycle_num-1}")
```

```
    plt.show()
```

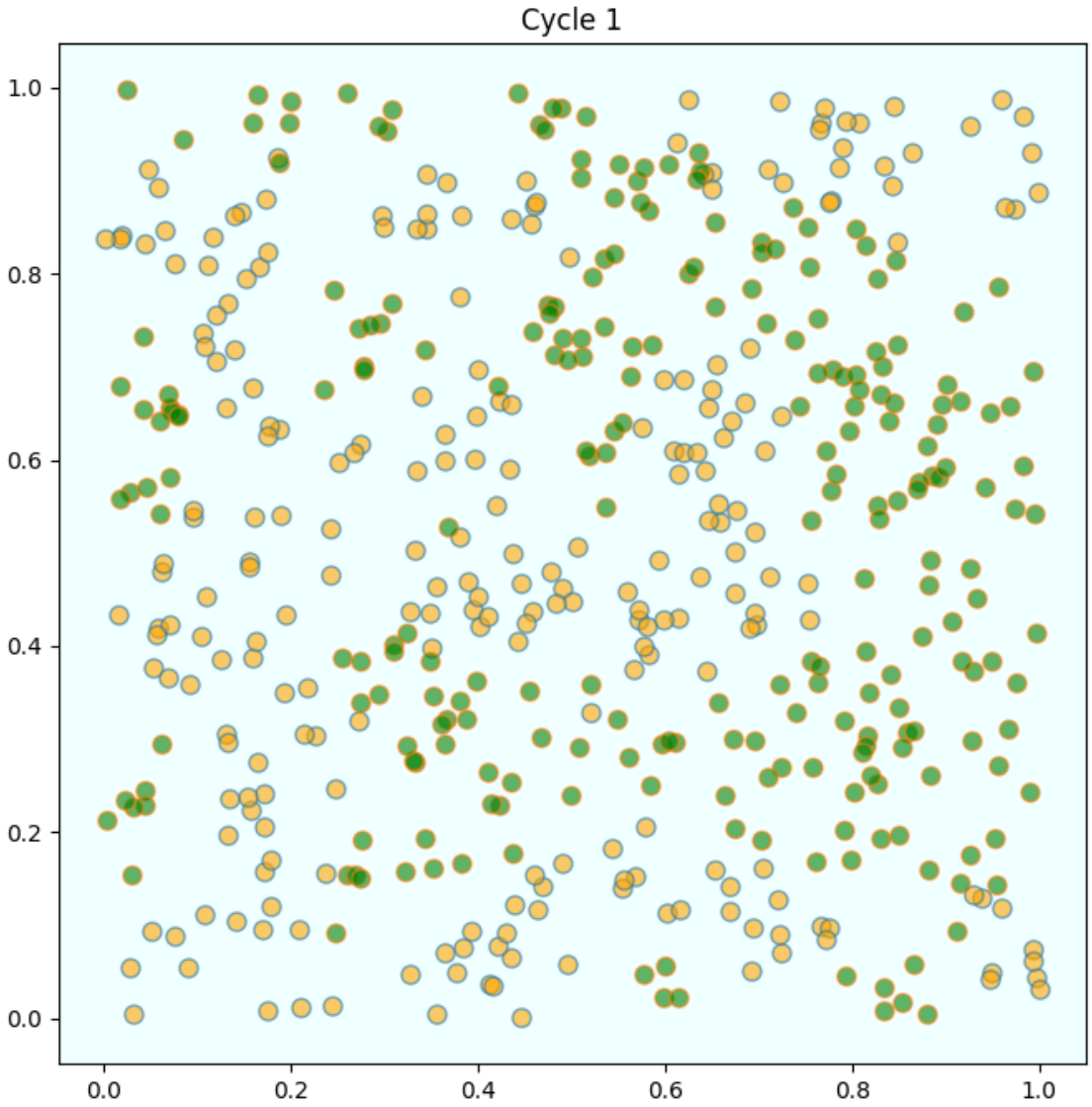
```
def run_model(num=(250, 250), neighbours=10, required=5):
    agents = [Agent(0) for i in range(num[0])]
    agents.extend(Agent(1) for i in range(num[1]))

    count = 1

    while True:
        print("Entering loop ", count)
        plot_distribution(agents, count)

        count += 1

        no_one_moved = True
        for agent in agents:
            old_location = agent.location
            agent.update(agents, neighbours, required)
            if agent.location != old_location:
                no_one_moved = False
        if no_one_moved:
            break

    print("Converged, terminating.")
```

```
seed(10)
run_model((250, 250), 10, 5)
```

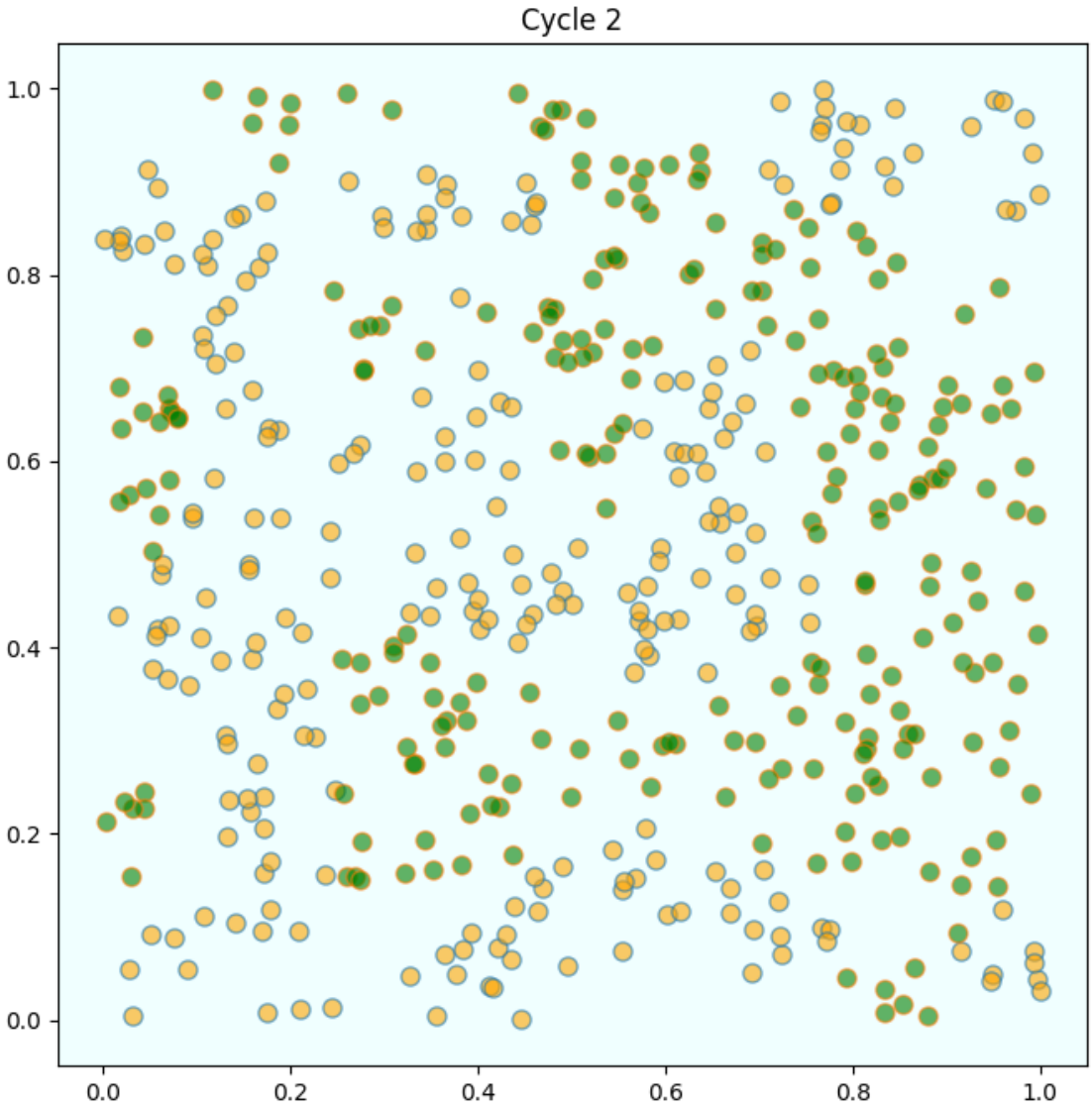
```
Entering loop  1
```



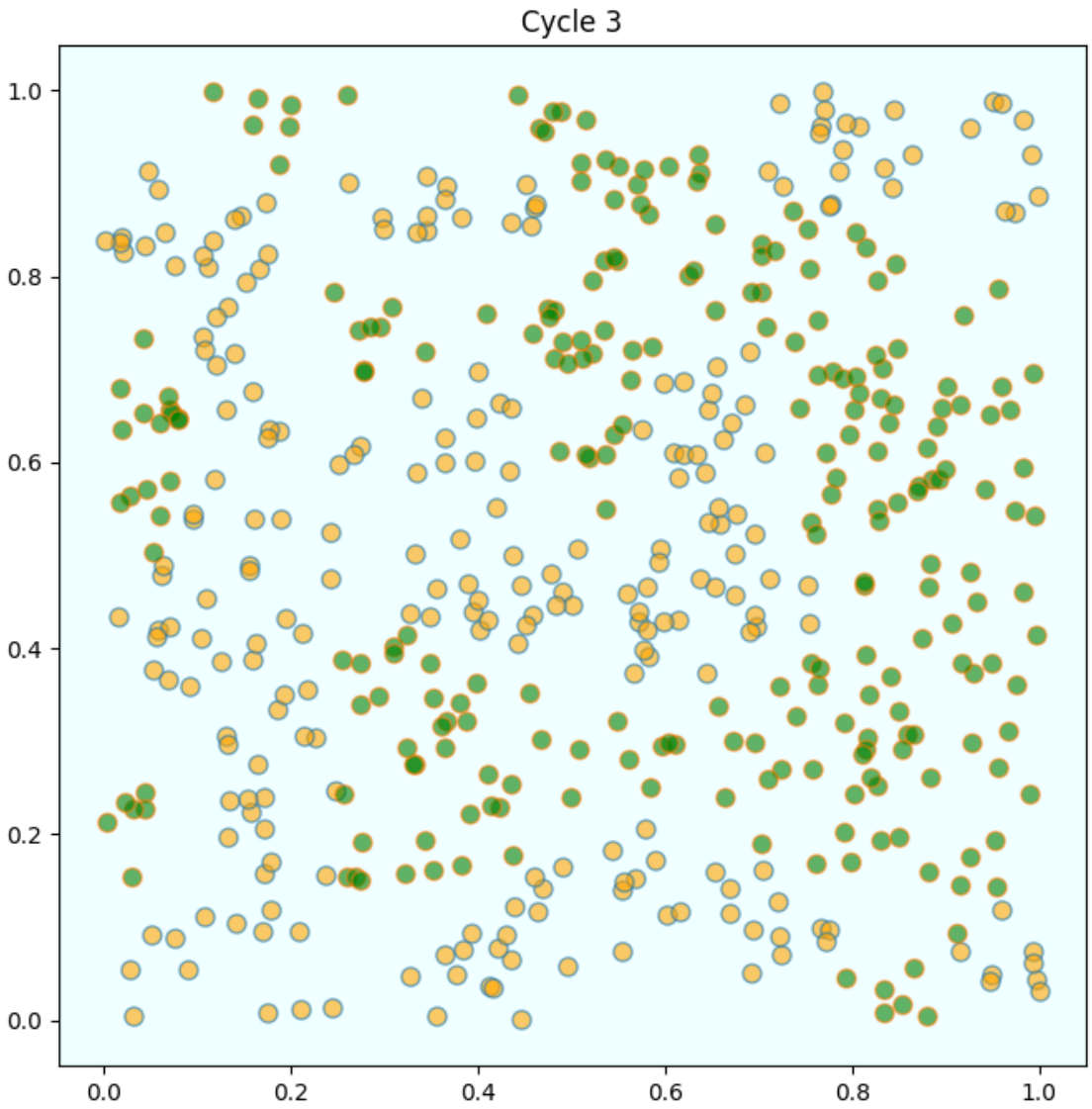

```
Entering loop  2
```




```
Entering loop  3
```



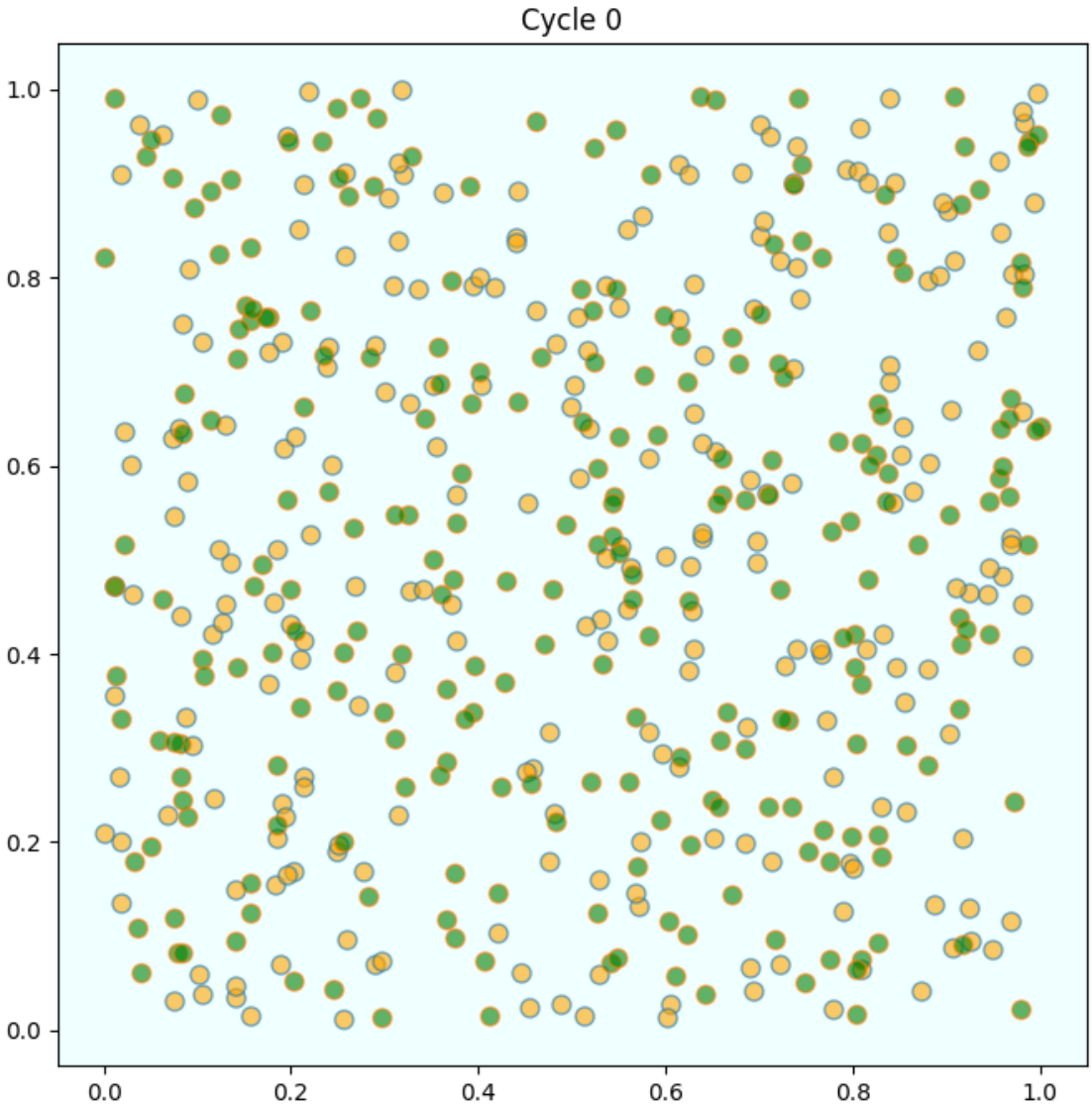

```
Entering loop  4
```



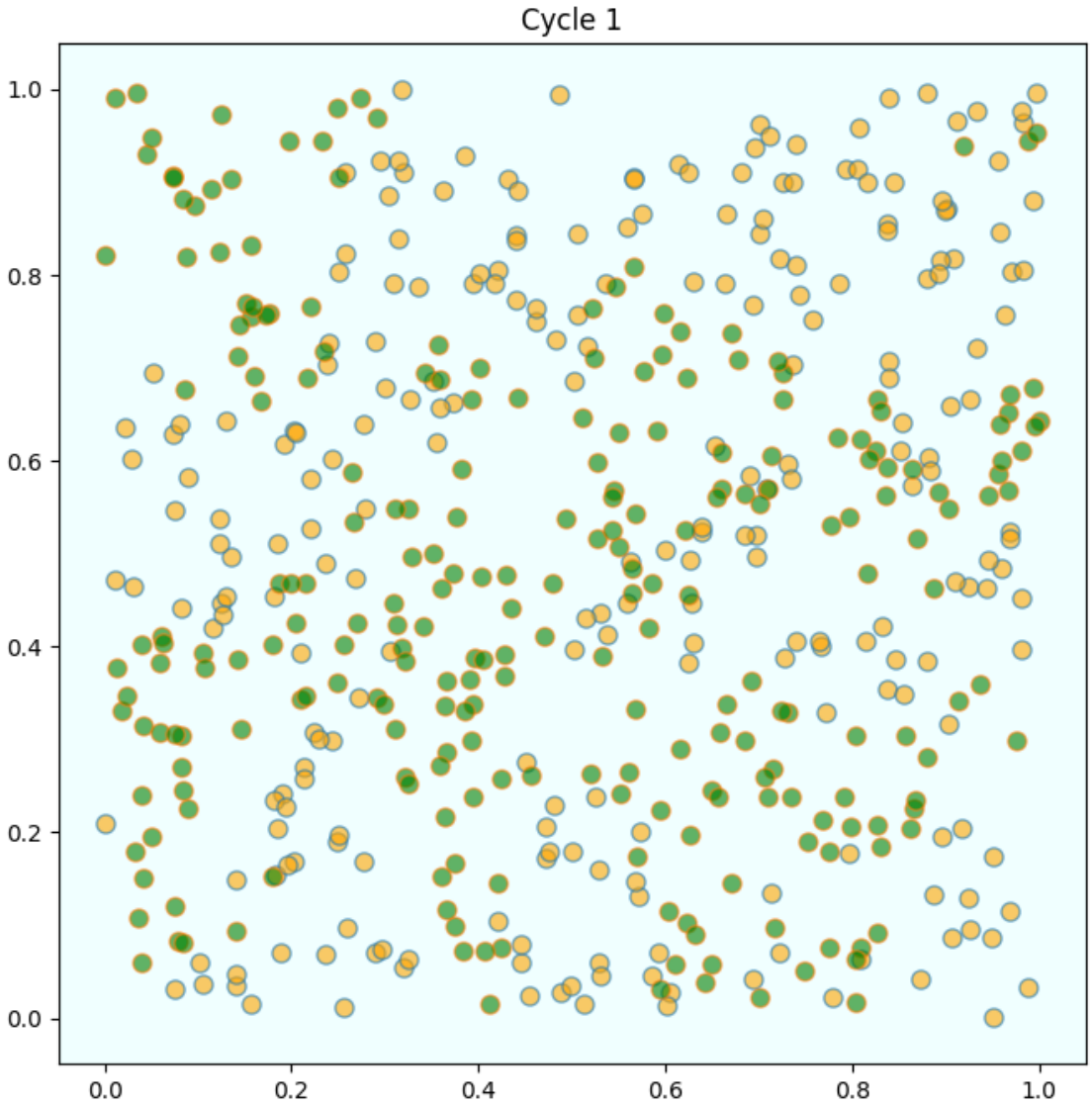

```
Converged, terminating.
```

```
seed(11)
run_model((250, 250), 10, 4)
```

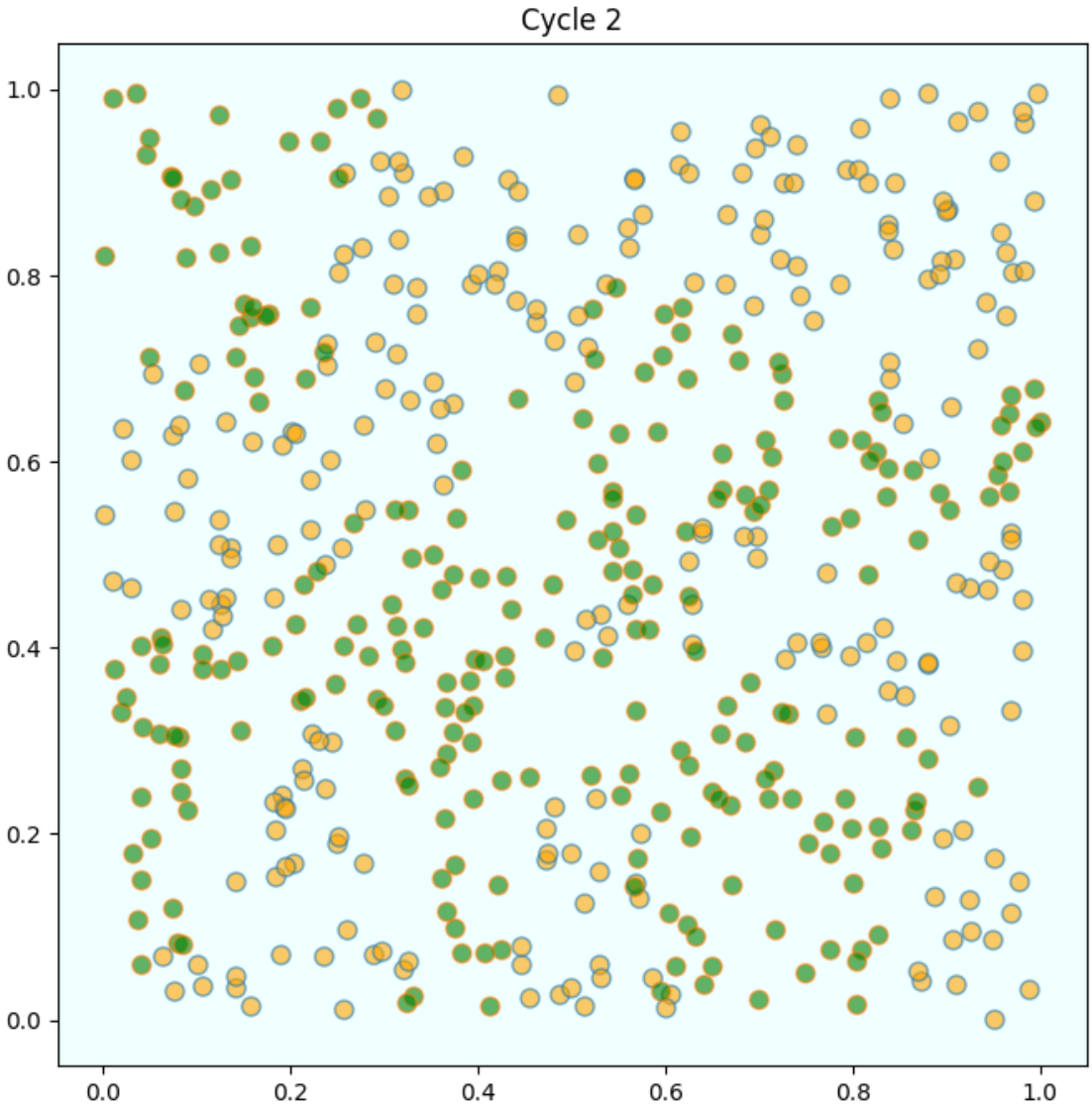
```
Entering loop  1
```



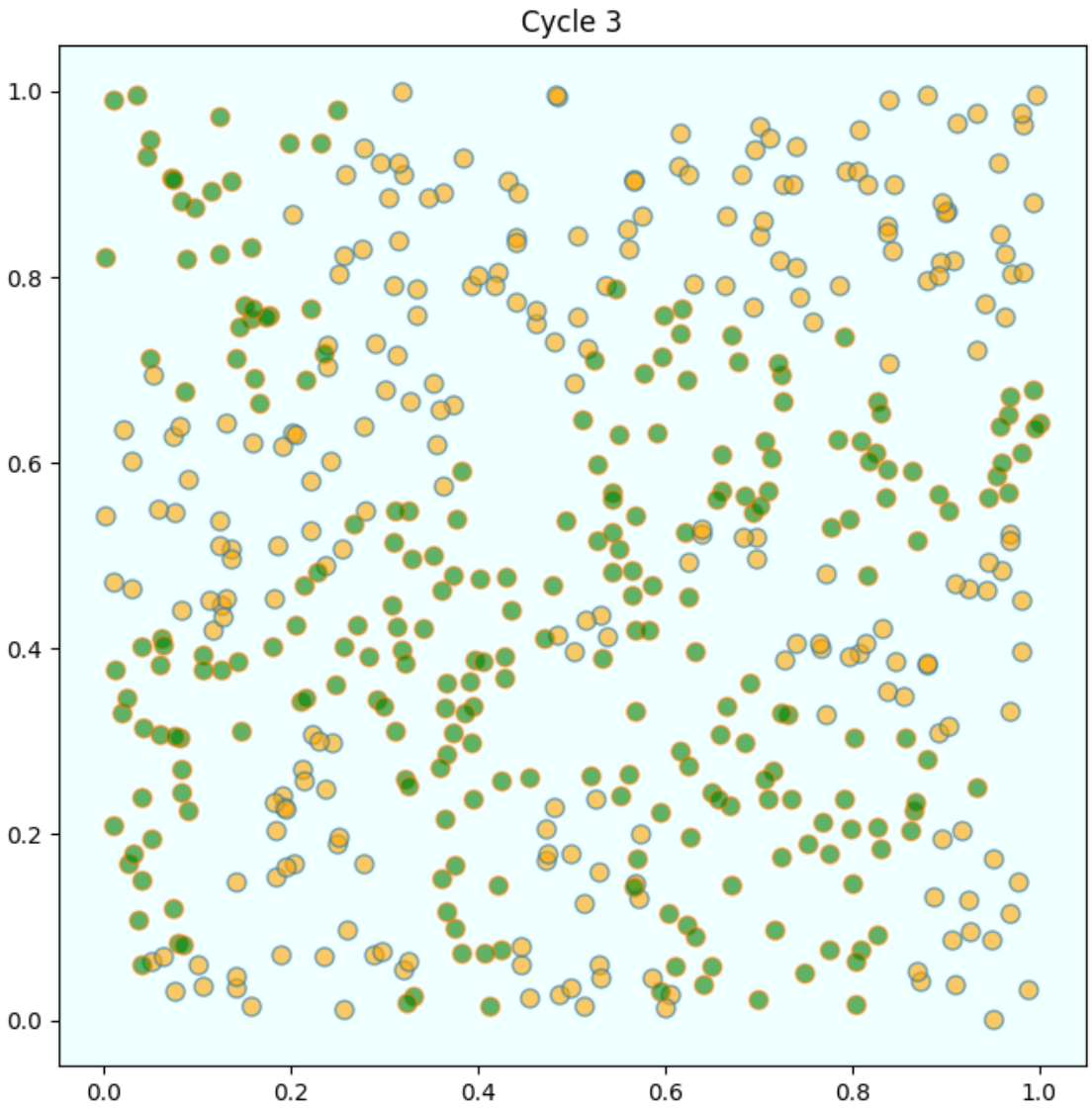

```
Entering loop  2
```



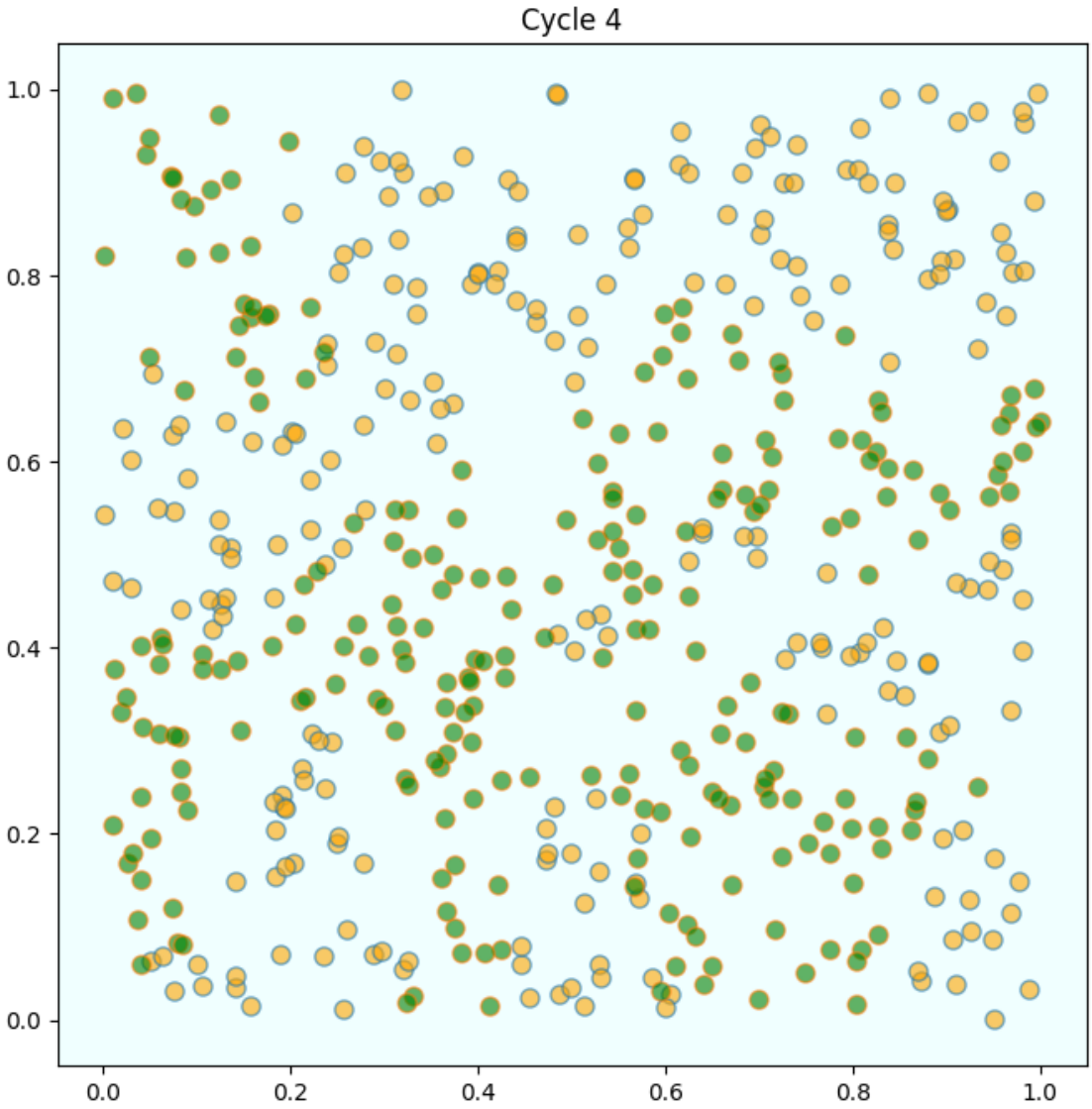

```
Entering loop  3
```



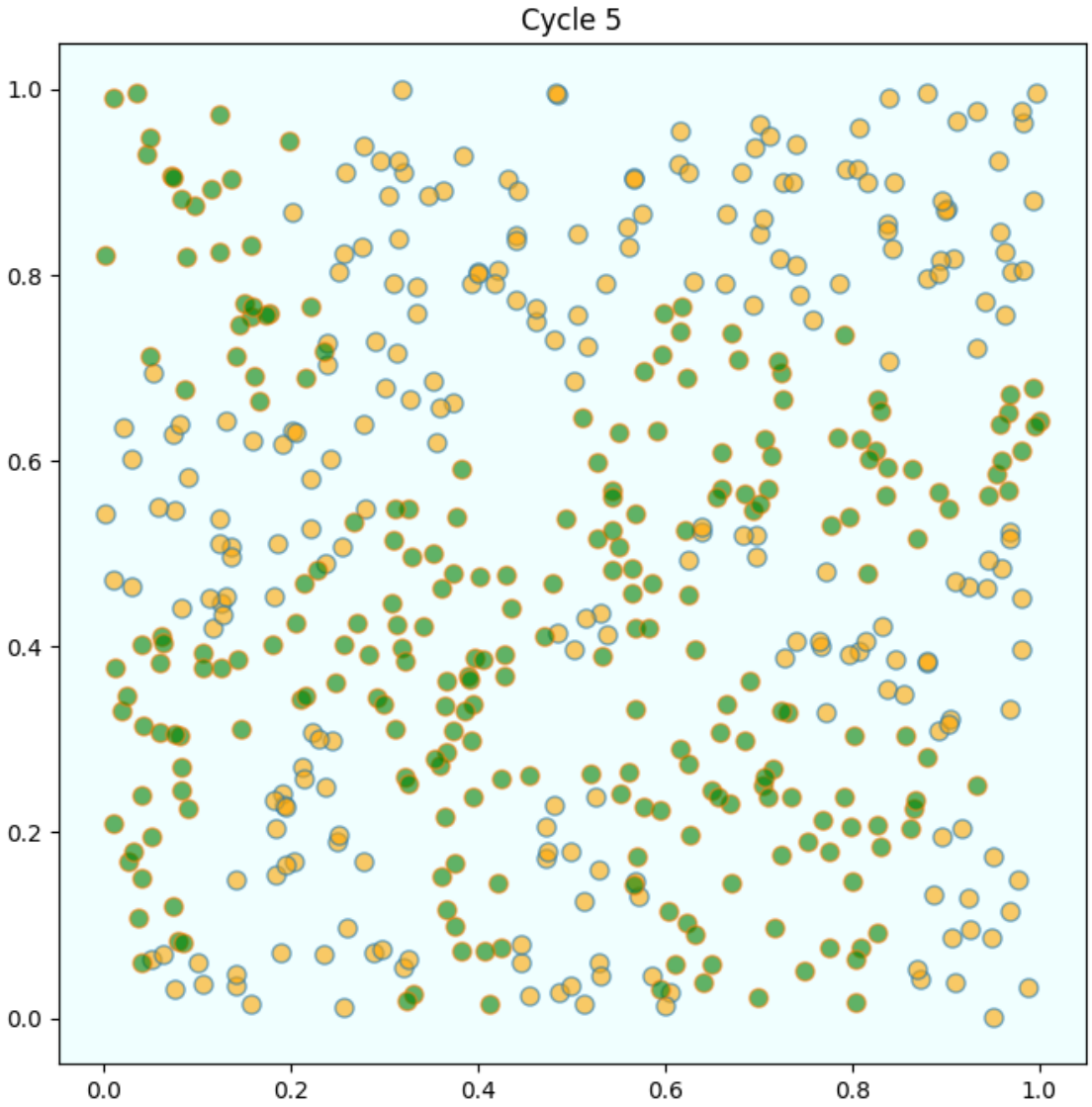

```
Entering loop  4
```



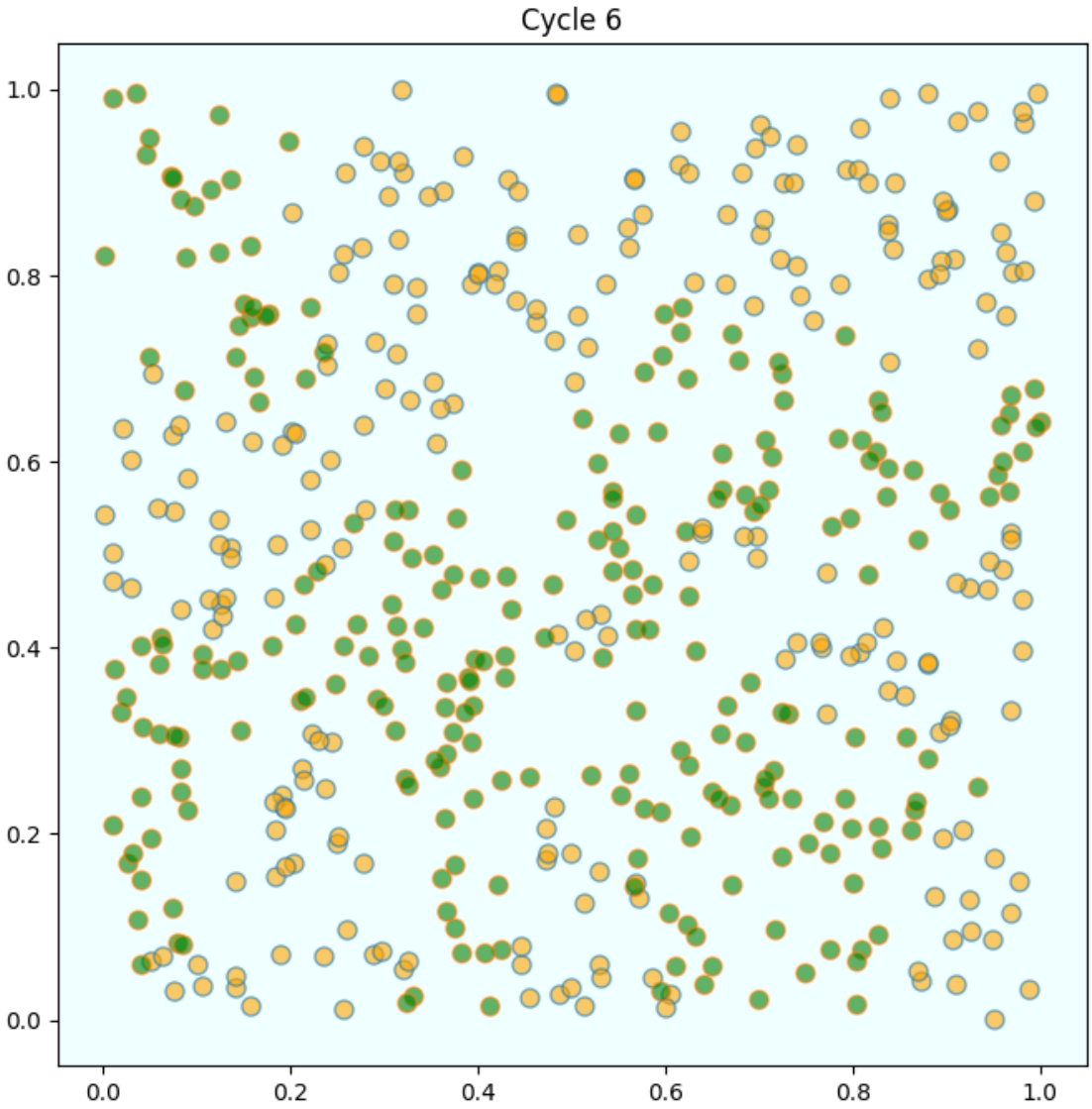

```
Entering loop  5
```




```
Entering loop  6
```




```
Entering loop  7
```




```
Converged, terminating.
```