Game_of_life

November 11, 2024

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import time

def neighbors_Moore(status):
    """
    Function to return the number of neighbors for each cell in status.

    Parameters
    ==========
    status : Current status.
    """

    # Initialize the neighbor count array
    n_nn = (
        np.roll(status, 1, axis=0) +  # Up.
        np.roll(status, -1, axis=0) +  # Down.
        np.roll(status, 1, axis=1) +  # Left.
        np.roll(status, -1, axis=1) +  # Right.
        np.roll(np.roll(status, 1, axis=0), 1, axis=1) +  # Up-Left.
        np.roll(np.roll(status, 1, axis=0), -1, axis=1) +  # Up-Right
        np.roll(np.roll(status, -1, axis=0), 1, axis=1) +  # Down-Left
        np.roll(np.roll(status, -1, axis=0), -1, axis=1)  # Down-Right
    )

    return n_nn


def apply_rule_2d(rule_2d, status):
    """
    Function to apply a 2-d rule on a status. Return the next status.

    Parameters
    ==========
    rule_2d : Array with size [2, 9]. Describe the CA rule.
    status : Current status.
    """
```

```
    Ni, Nj = status.shape  # Dimensions of 2-D lattice of the CA.
    next_status = np.zeros([Ni, Nj])

    # Find the number of neighbors.
    n_nn = neighbors_Moore(status)
    for i in range(Ni):
        for j in range(Nj):
            next_status[i, j] = rule_2d[int(status[i, j]), int(n_nn[i, j])]

    return next_status


# Apply Game of life
N = 200
repeat = 5

rule_2d = np.zeros([2, 9])

# Game of Life's rules.
rule_2d[0, :] = [0, 0, 0, 1, 0, 0, 0, 0, 0]  # New born from empty cell.
rule_2d[1, :] = [0, 0, 1, 1, 0, 0, 0, 0, 0]  # Survival from living cell.

T  = 400
alive_counts = []  # record A(t)
changed_counts = []  # record C(t)

all_alive_counts = []
all_changed_counts = []
average_density = []

average_density_all_runs = []

for run in range(repeat):

    alive_counts = []  # record A(t)
    average_density_per_run = []  # record density
    changed_counts = []  # record C(t)
    gol = np.random.randint(2, size=[N, N]) # Random initial state.

    for step in range(T):
        prev_status = gol.copy()
        gol = apply_rule_2d(rule_2d, gol)   # update cells

        # num of survive cells
        alive_counts.append(np.sum(gol))
```

```
        # density of every time step
        average_density_per_run.append(np.sum(gol)/ (N**2))

        # num of changed cells
        changed_counts.append(np.sum(gol != prev_status))


        time.sleep(0.05)  # timestep


    all_alive_counts.append(alive_counts)
    all_changed_counts.append(changed_counts)


    average_density_all_runs.append(average_density_per_run)
    print(f'Run = {run + 1}, A(t) C(t) have saved.')

# Task1 Q2

steady_state_densities = []

for run in average_density_all_runs:
    steady_state_density = np.mean(run[250:]) # steady state iteration >= 250
    steady_state_densities.append(steady_state_density)

mean_density = np.mean(steady_state_densities)
print(f'Q1: the average density of alive cell per unit area is {mean_density}')

# Task1 P1
plt.figure(1)
for run in range(repeat):
    plt.plot(range(T), all_alive_counts[run], label=f'Run {run + 1}',␣
 ↪color=f'C{run}')
plt.xlabel('Time Step t')
plt.ylabel('Number of Alive Cells A(t)')
plt.title('Alive Cells Over Time')
plt.legend()
plt.grid(True)
plt.savefig('alive_cells_over_time.png')
plt.pause(2)
plt.close()


# Task2 P2
plt.figure(2)
for run in range(repeat):
```

```
    plt.plot(range(T), all_changed_counts[run], label=f'Run {run + 1}',␣
 ↪color=f'C{run}')
plt.xlabel('Time Step t')
plt.ylabel('Number of State Changes C(t)')
plt.title('State Changes Over Time')
plt.legend()
plt.grid(True)
plt.savefig('state_changes_over_time.png')
plt.pause(2)
plt.close()
```