题目描述

A．虫口模型与Logistic映射

1．对于迭代方程

$$x_{n+1} = 1 - \mu x_n^2$$

分析不同的 $\mu \in (0,2)$ 下 x_n 的长时间行为，以及其与初值 $x_0 \in (-1,1)$ 的关系。（2分）

2．将上述的方程换成

$$x_{n+1} = \cos x - \mu x_n^2$$

合理选择 μ, x_0 的区间，并分析 x_n 的长时间行为。（1分）
3．比较两种情况，你发现了什么？（1分）

B．开心消消乐

在一个 $L \times L$ 的网格（即每行每列均有 L 个格子）上定义如下的消消乐游戏：
1．在最开始，格子上没有任何的方块。
2．每次挑选任意一个格子，放上去一个方块。每放上去一个方块，系统开始演化一个时间步。一个时间步内的演化规则见后面一条。
3．如果一个格子上的方块数目大于等于 4 ，则消去 4 个方块，并记录一次消去，这四个方块会移动累加到上下左右的 4 个格点（它们各自方块数 +1 ），（如果有一个方块的两个邻居同时消去并给到它，则它的方块数量 +2 ，也就是说影响是叠加的；考虑开放边界条件，即如果一个格子在边界发生消去，则有一个或者两个方块移动出去了）。每次同时消去所有当前高度大于等于 4 的格子，并记录下总的消去次数。直到场上没有任何格子的方块数 ≥ 4 ，这一个时间步结束。
4．每一步的得分 s 即为这一步的总消去次数。

1. 选取网格大小 $L = 32$ ，演化足够多的时间步，并记录网格平均方块密度 n（ $n = \frac{N}{L^2}$ ，其中 N 是场上的方块总数）随着演化时间 t 的关系。你发现了什么？（2分）
2. 选取并固定一个合适的系统尺寸 L（取决于你的计算能力），统计长时间演化时，每步得分的频率分布 $P(s)$ 。你发现了什么规律？（2分）
3. 选取多个不同的系统尺寸 L 重复（2）的过程，验证你发现的统计规律。系统的尺寸对这一规律有什么影响？（1分）
4. 你能解释你在（B）中发现的规律吗？有什么别的思考？（1分）

算法浅析

A．虫口模型与Logistic映射

本题目考察了迭代方程和Logistic映射的一些性质。该题所用算法简单，将x_n的值代入方程，得到新的x_{n+1}值，重复这一过程，即可得到x_n的序列。

具体来说，本程序通过 `logistic_map` 重复这一过程，并将x_n序列的值输出。

B．开心消消乐

本题的实现关键在于如何快速找到大于等于4的格子，也即需要消去的格子。

在 `B.py` 中，我们使用了一个 `numpy` 数组来表示网格，并在 `exist_4_or_more` 对x,y坐标进行检索，找到大于等于4的格子。

在 `B_tree.py` 中，我使用深度搜索(BFS)的方法进行搜索，找到大于等于4的格子。伪代码如下：

```python
def find_4_or_more(L,gridll,x,y,s):
    if gridll[x][y] < 4:
        return
    if gridll[x][y] >=4 :
        delete_block(L,gridll,x,y)
        s[0] += 1
        if x >=1:
            find_4_or_more(L,gridll,x-1,y,s)
        if x < L-1:
            find_4_or_more(L,gridll,x+1,y,s)
        if y >=1:
            find_4_or_more(L,gridll,x,y-1,s)
        if y < L-1:
            find_4_or_more(L,gridll,x,y+1,s)
```

项目结构

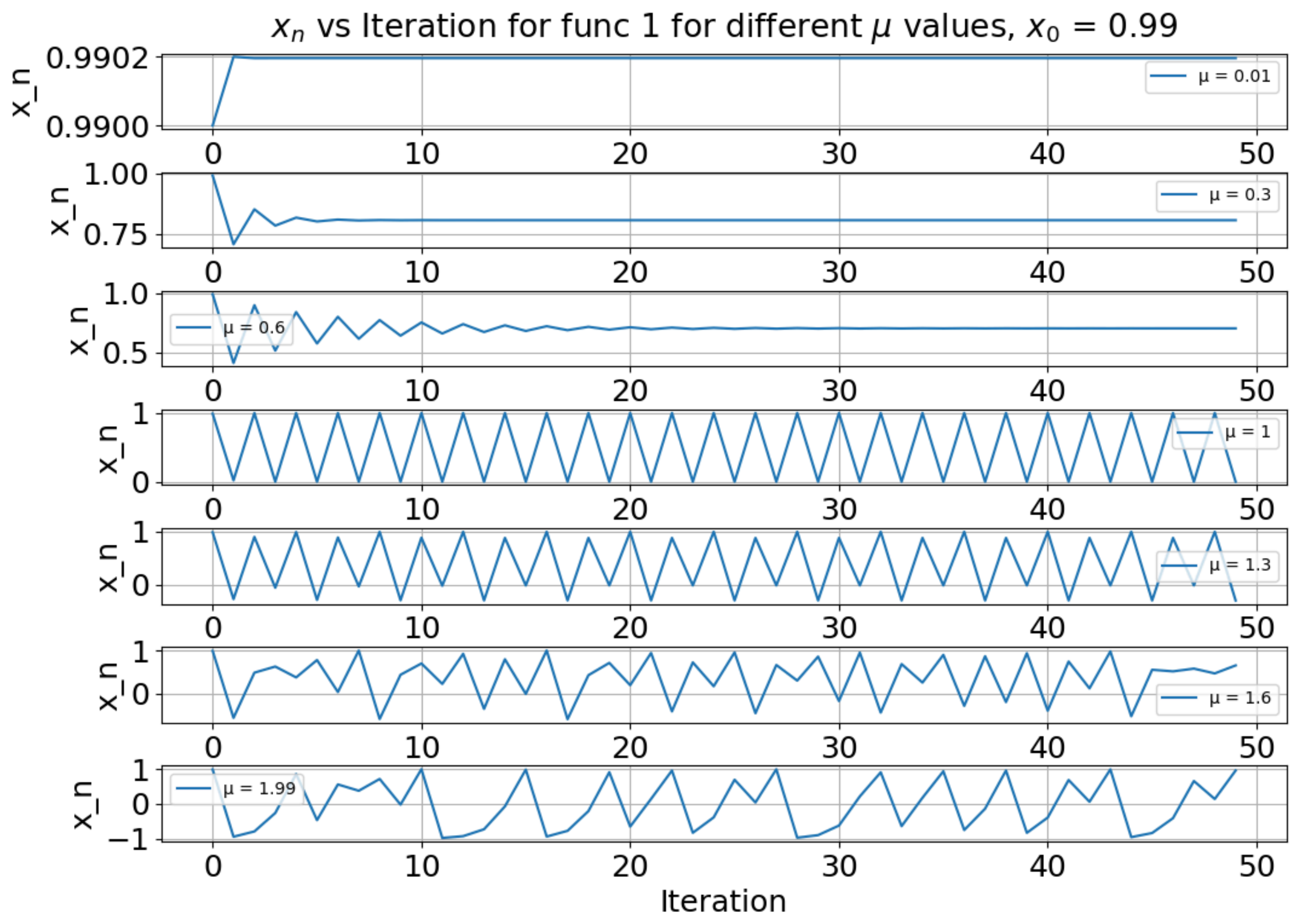
```
.
├── README.md
├── code
│   ├── A_1.py 对应第一个迭代方程
│   ├── A_2.py 对应第一个迭代方程
│   ├── B.py  B题的普通算法
│   ├── B_tree.py  使用深度有限搜索的B题算法
├── figure
```

结果及分析

A．虫口模型与Logistic映射

迭代方程一（$x_{n+1}=1-\mu x_n^2$）

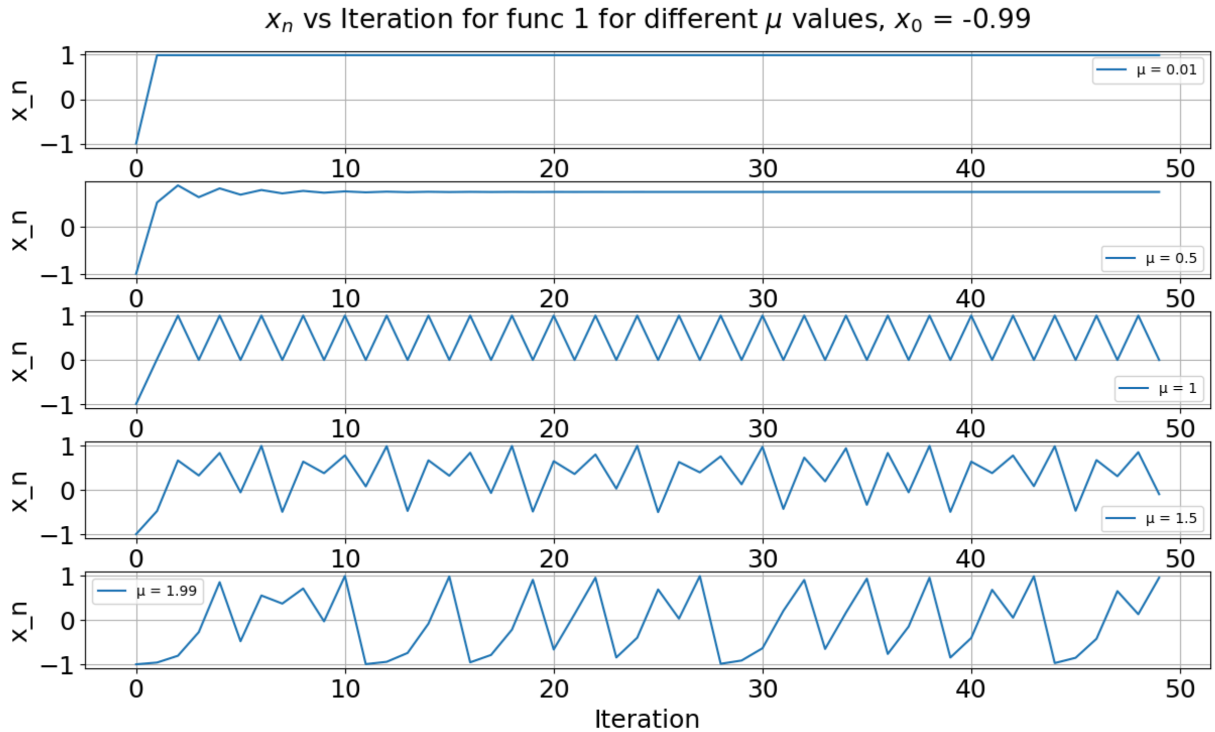
不妨选取$x_0=0.5$，选取$\mu=0.01$，0.3，$0.6,1,1.3,1.6$，1.99，画出x_n的长时间行为。代码运行结果如下：



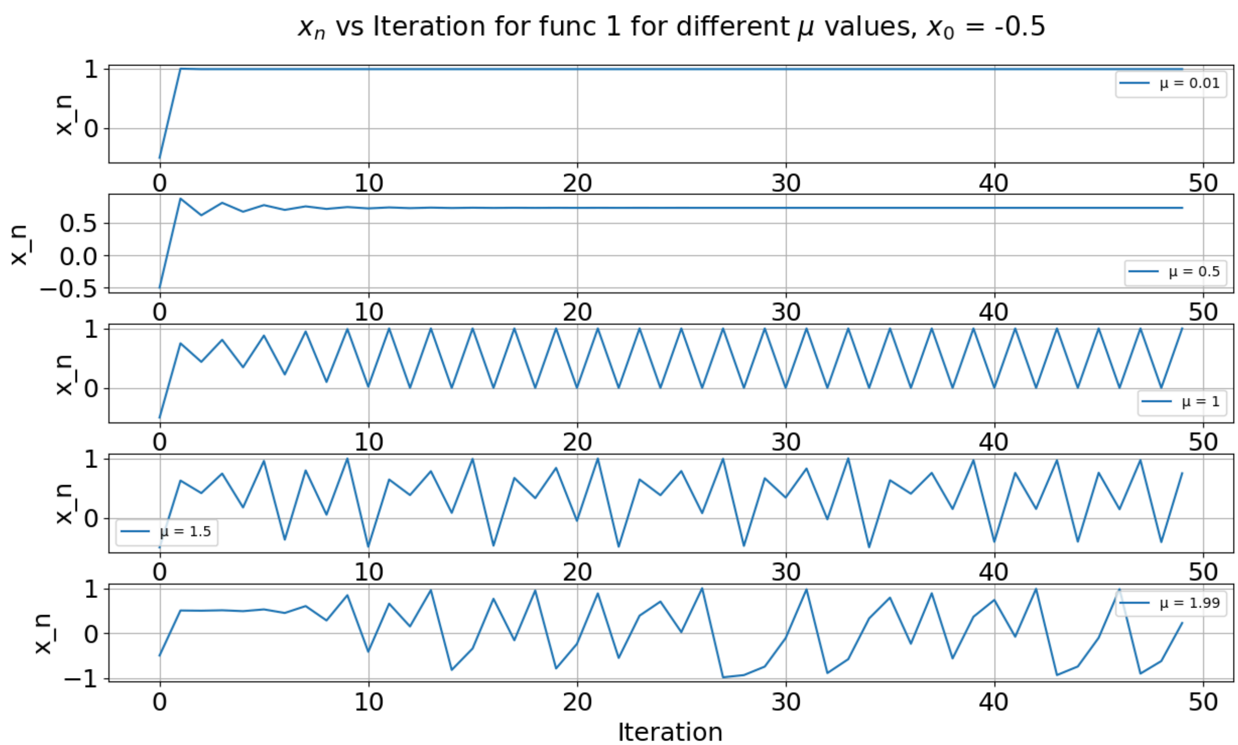
由上图可以看出，当μ接近于0时，x_n迅速收敛于1，保持稳定；当μ较小的时候，x_n也可以迅速收敛到一个稳定值；随着μ的增大，x_n的收敛速度变慢；当μ等于1时，x_n规律地在0和1之间震荡；当μ继续增大的同时，x_n的波动幅度变大，由规律振荡过渡为无规律性、无周期性的振荡；当μ接近于2时，x_n的振荡幅度变得很大，且呈现出混沌性。

不妨选取不同的x_0，画出x_n的长时间行为，研究其与x_0初值的关系：

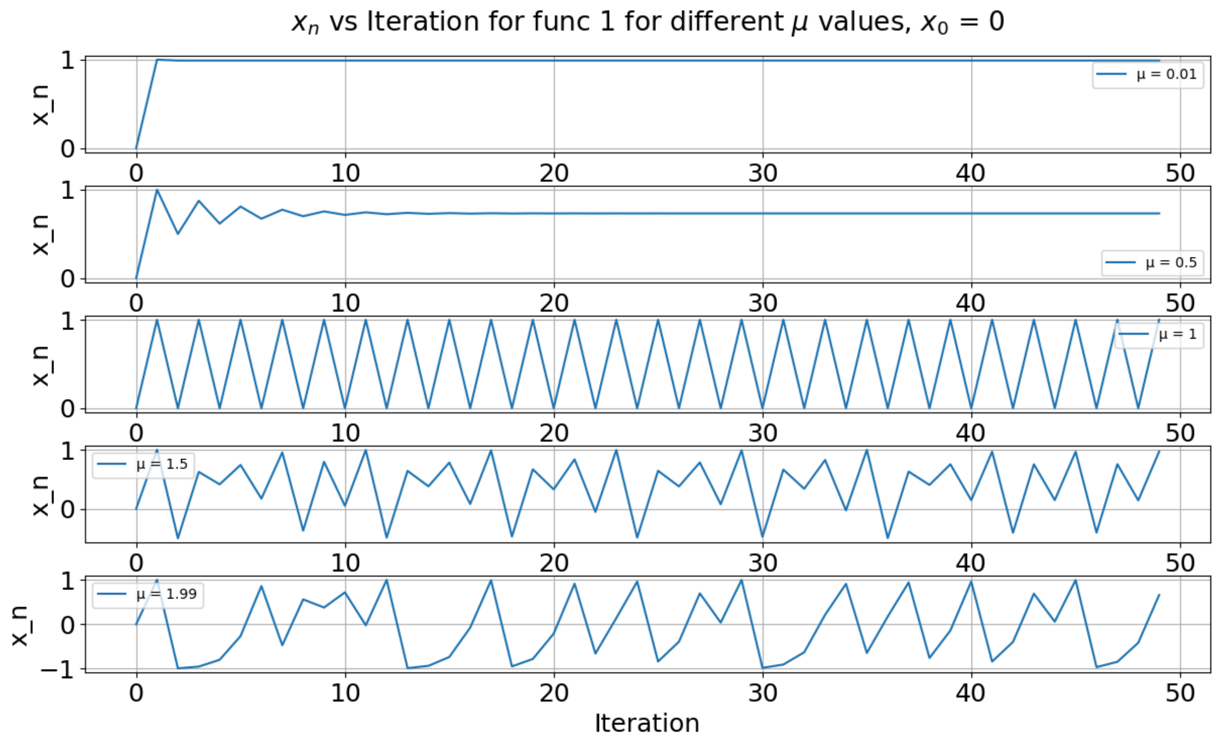
$x_0 = -0.99$:


x_n vs Iteration for func 1 for different μ values, x_0 = -0.99
μ = 0.01
μ = 0.5
μ = 1
μ = 1.5
μ = 1.99
x_n
Iteration

$x_0 = -0.5$:

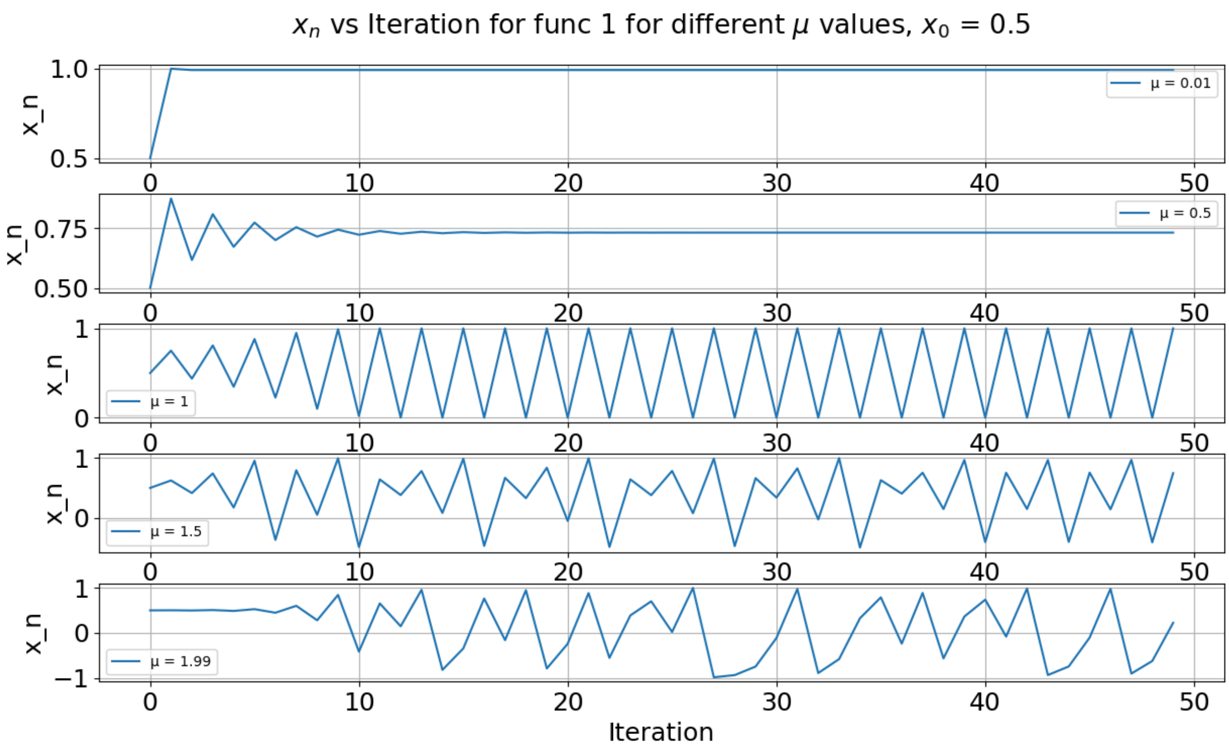

x_n vs Iteration for func 1 for different μ values, x_0 = -0.5
μ = 0.01
μ = 0.5
μ = 1
μ = 1.5
μ = 1.99
x_n
Iteration

$x_0 = 0$:



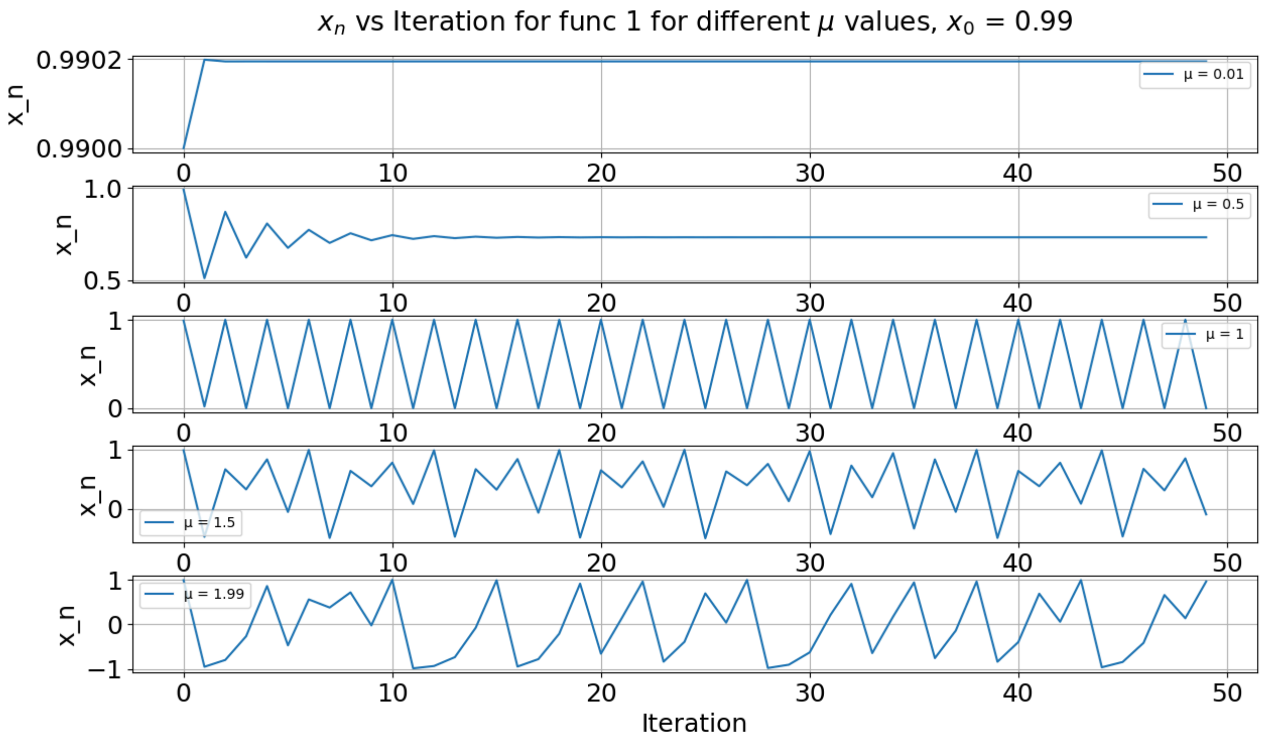
x_n vs Iteration for func 1 for different μ values, x_0 = 0
x_n
μ = 0.01
μ = 0.5
μ = 1
μ = 1.5
μ = 1.99
Iteration

$x_0 = 0.5$:



x_n vs Iteration for func 1 for different μ values, x_0 = 0.5
x_n
μ = 0.01
μ = 0.5
μ = 1
μ = 1.5
μ = 1.99
Iteration

$x_0 = 0.99$:

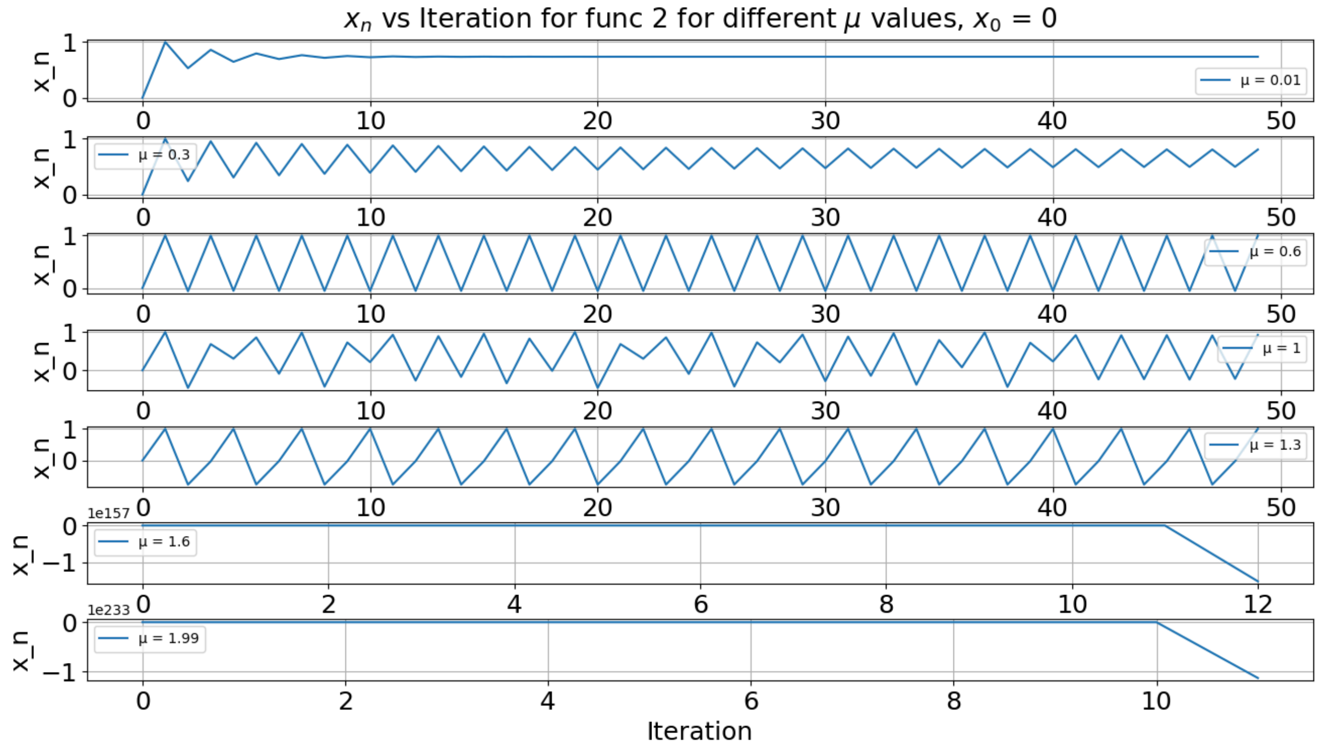


综合以上结果，我们发现对于不同的$x-0$取值，以上得到的基本现象是一致的。也就是说x_n的长时间行为受x_0的初始值影响不大。具体分析来说：

当μ接近于0时，x_n迅速收敛于1，保持稳定；随着μ的增大，x_n迅速收敛到一个稳定值，且x_0越大，x_n的稳定解越小，收敛到稳定值的速度越慢；当μ等于1时，x_n逐渐过渡到规律地在0和1之间震荡；当μ继续增大的同时，x_n的波动幅度变大，由规律振荡过渡为无规律性、无周期性的振荡；当μ接近于2时，x_n的振荡幅度变得很大，且呈现出混沌性。

迭代方程二（$x_{n+1} = \cos x - \mu x_n^2$）

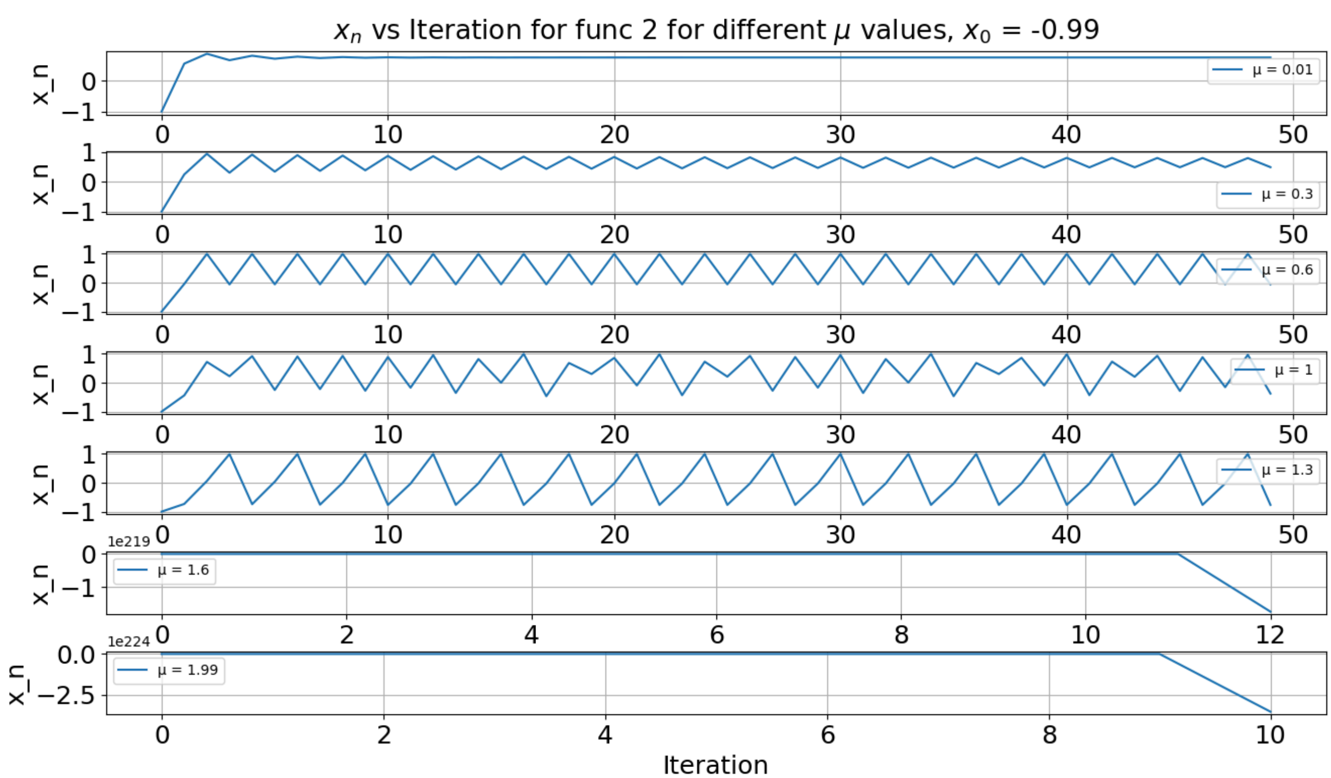
不妨选取$x_0 = 0$，选取$\mu = 0.01$，0.3，$0.6, 1, 1.3, 1.6$，1.99，画出x_n的长时间行为。代码运行结果如下：



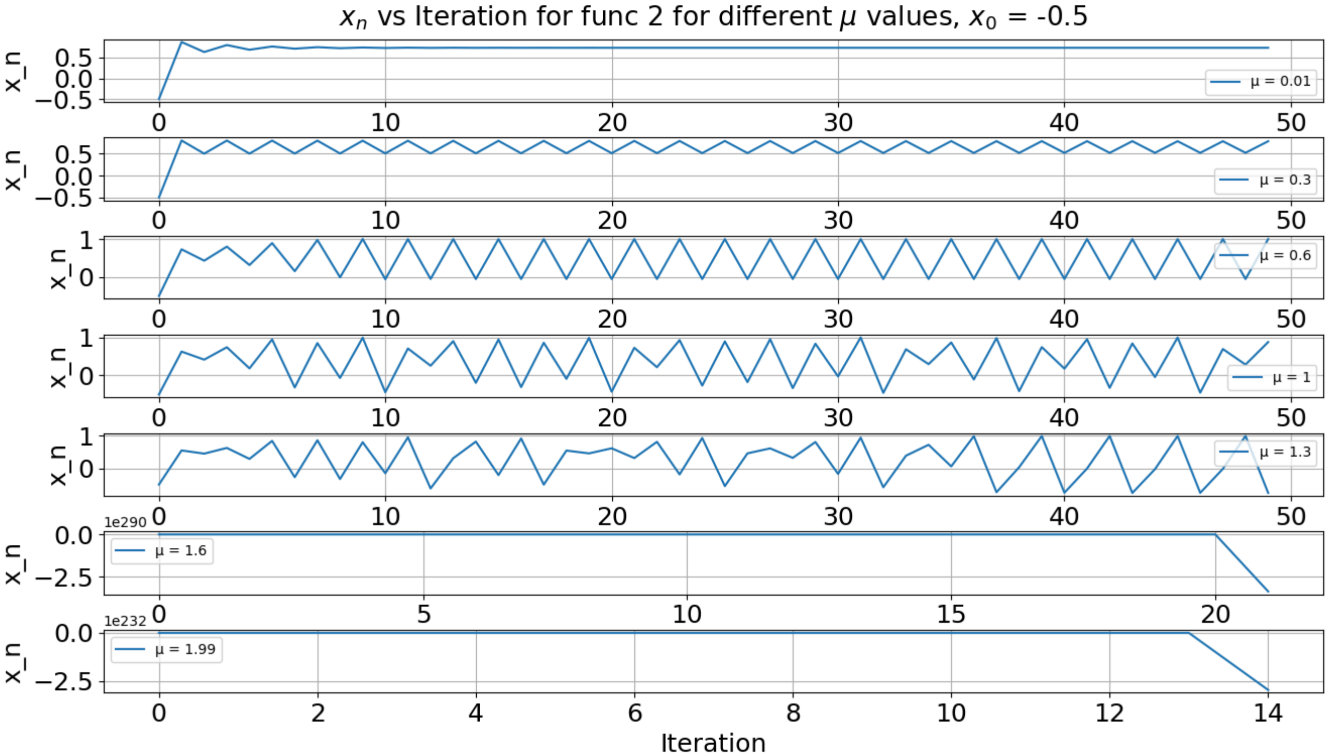
由上图可以看出，当μ接近于0时，x_n迅速收敛，保持稳定；随着μ的增大，x_n保持周期性振荡；随着μ的继续增大，当μ接近于1时，x_n的显现出无周期性的混沌；当μ继续增加时，x_n又重新显现出周期性；当μ继续增大的同时，x_n不再收敛，迅速发散至负无穷。

不妨选取不同的x_0，画出x_n的长时间行为，研究其与x_0初值的关系：

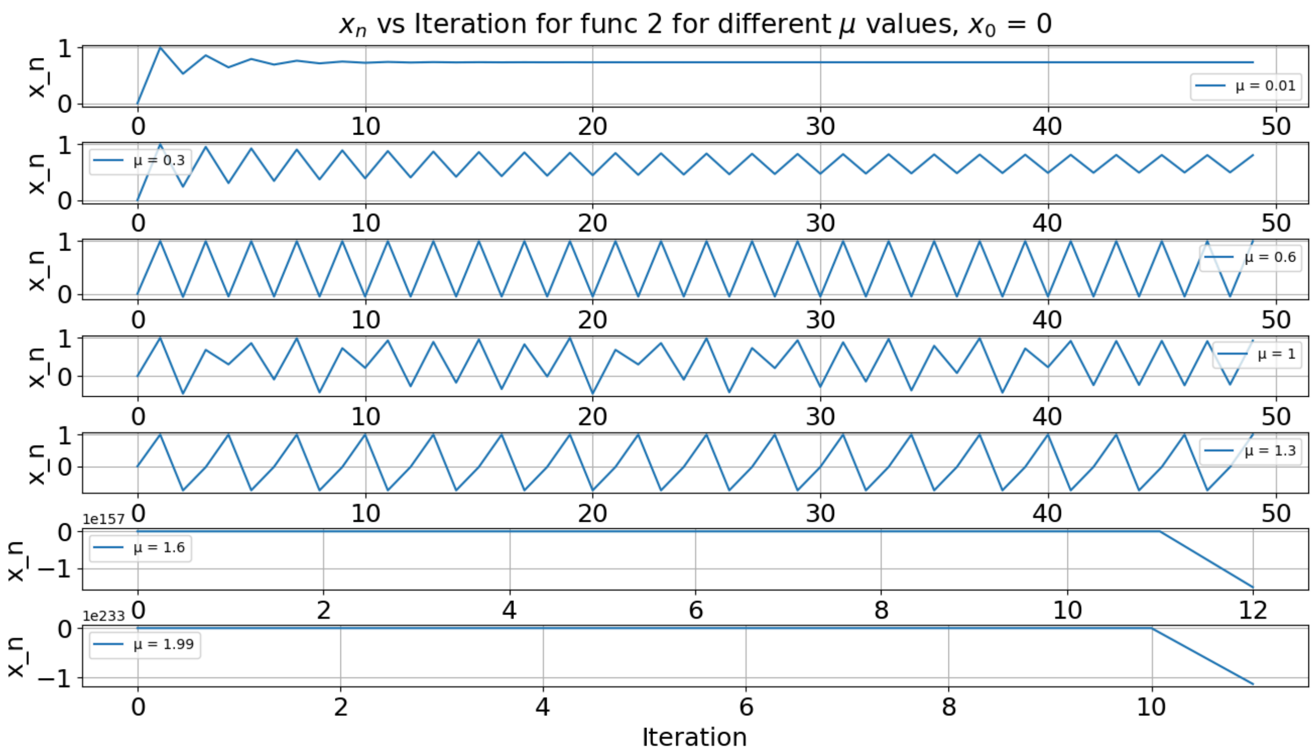
$x_0 = -0.99$:



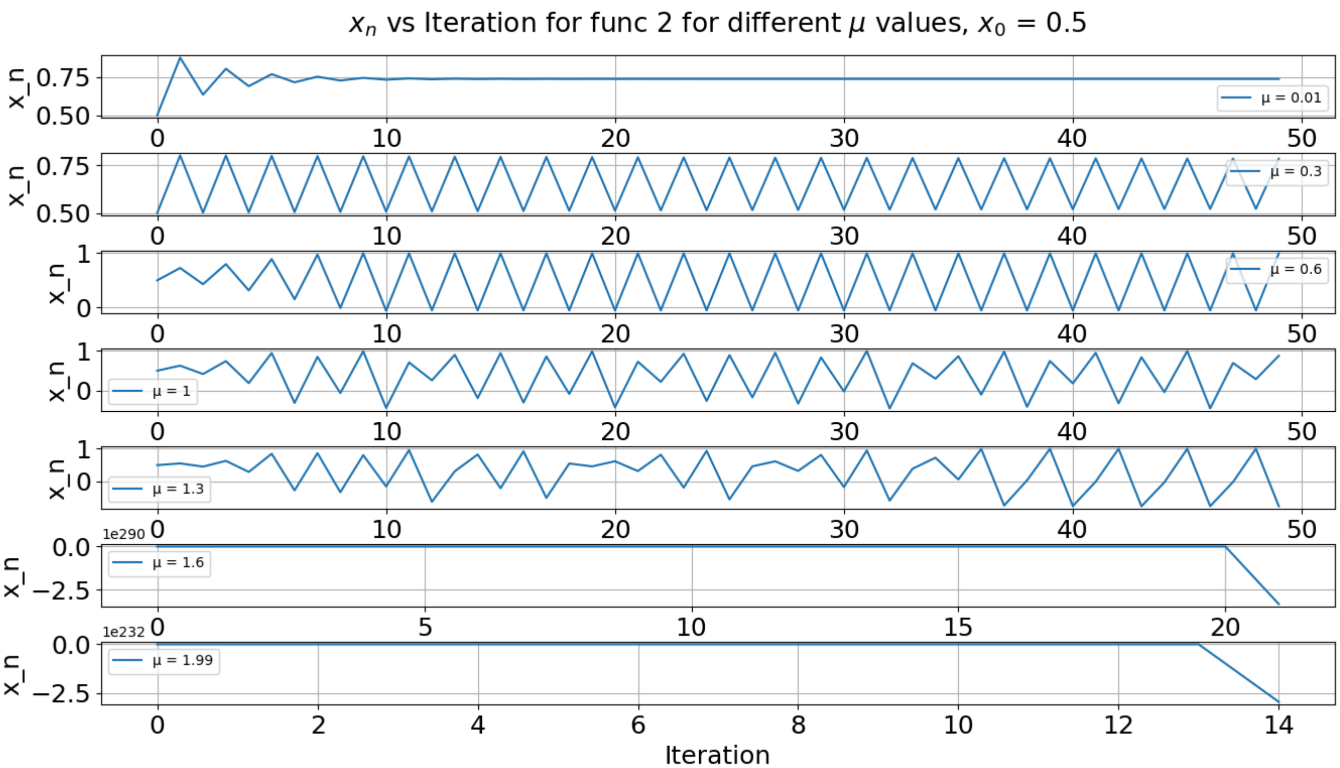
$x_0 = -0.5$:



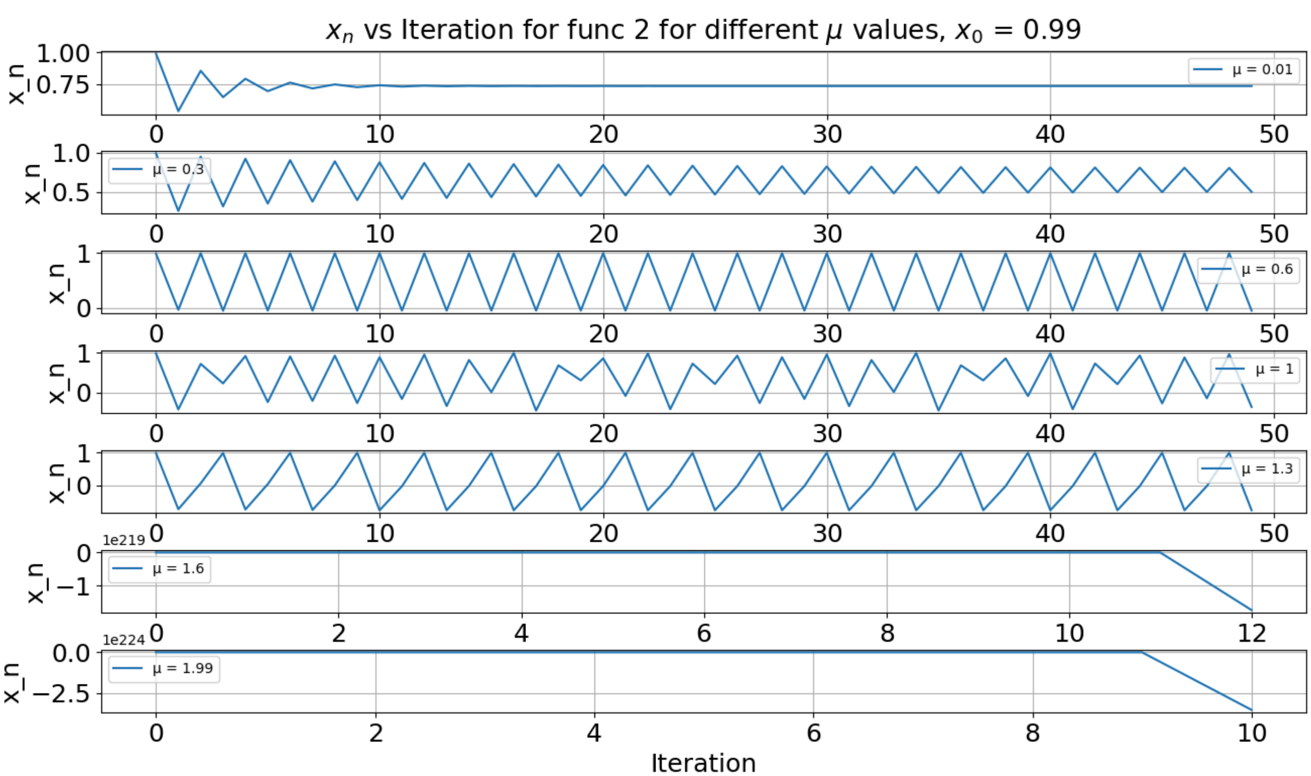
$x_0 = 0$:



$x_0 = 0.5$:



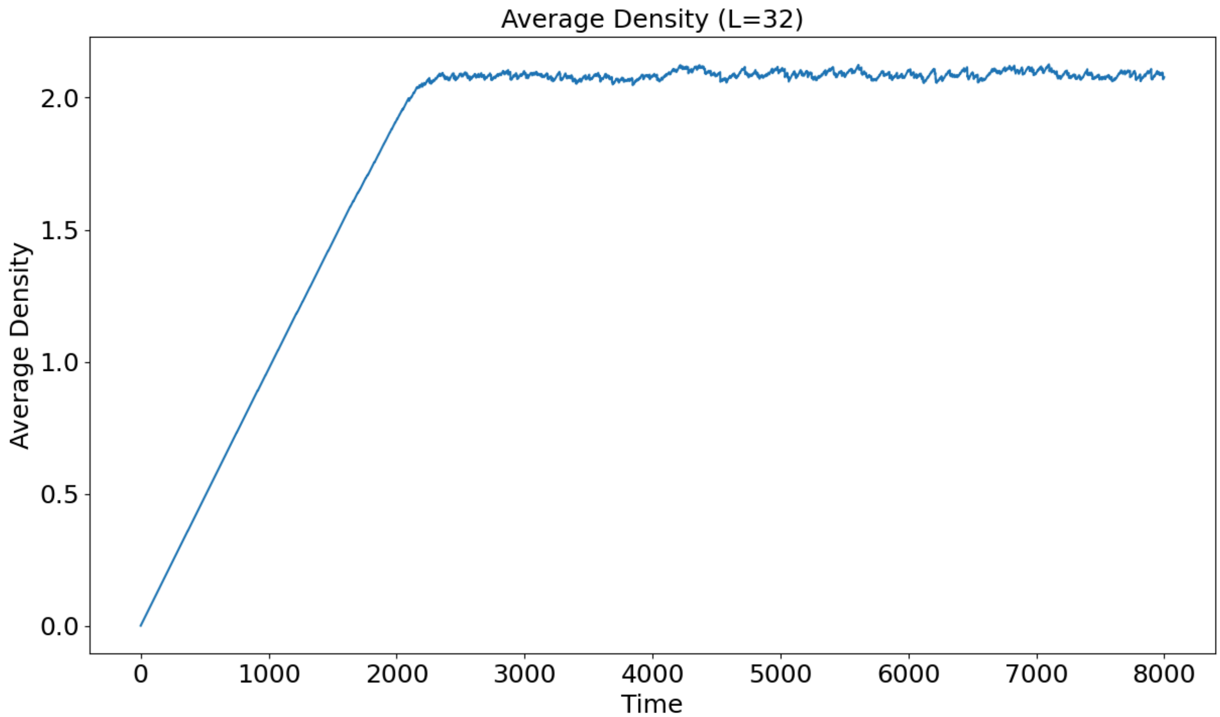
$x_0 = 0.99$:



综合以上结果，我们发现对于不同的$x-0$取值，以上得到的基本现象是一致的。

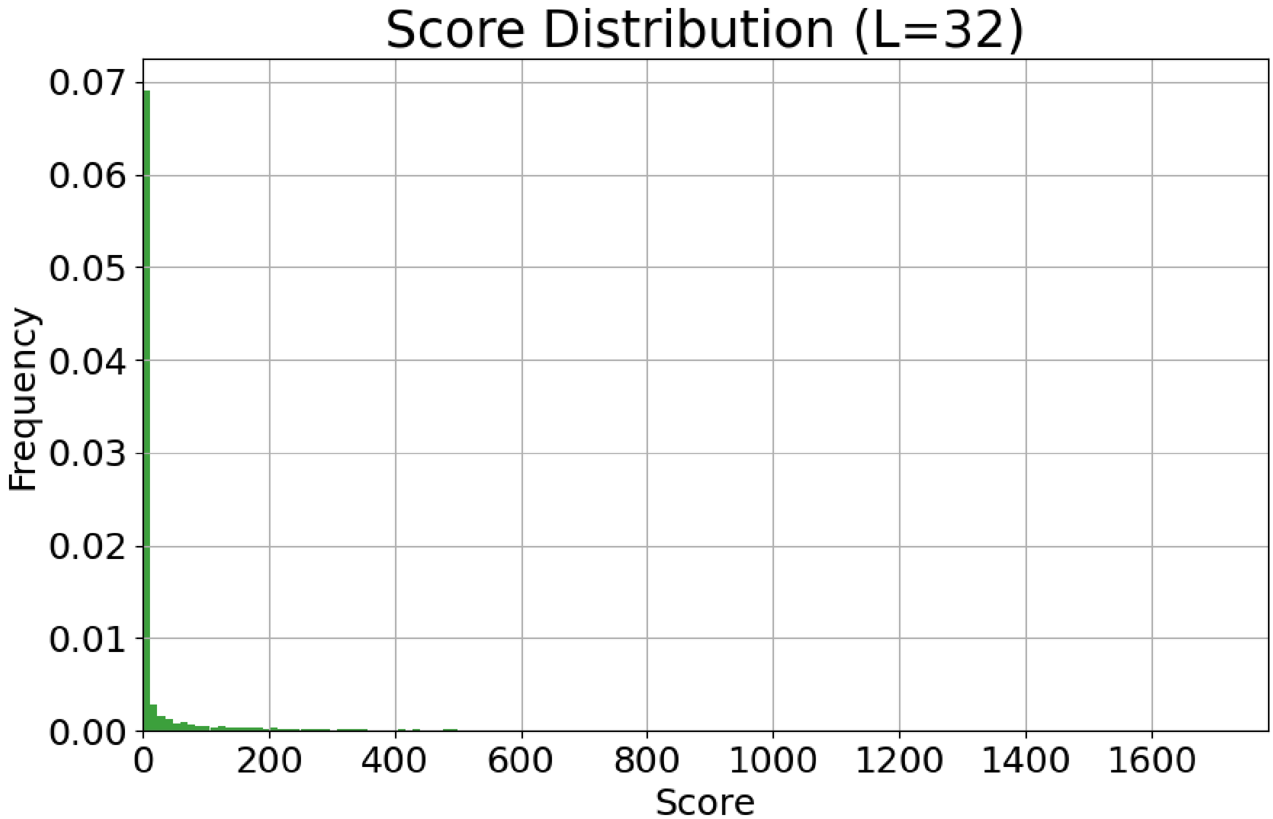
B.开心消消乐

1. 运行代码，得到网格平均方块密度 n随着演化时间 t 的关系：



由上图可以看出，初始网格密度线性增长，这说明初始时刻在边缘“溢出”的格子较少。随着时间的推移，网格密度逐渐维持在一个稳定的值，这说明随机增加效应和边缘溢出效应达到了某种程度上的平衡。

2. 统计长时间演化时，每步得分的频率分布 $P(s)$ 。代码运行结果如下：



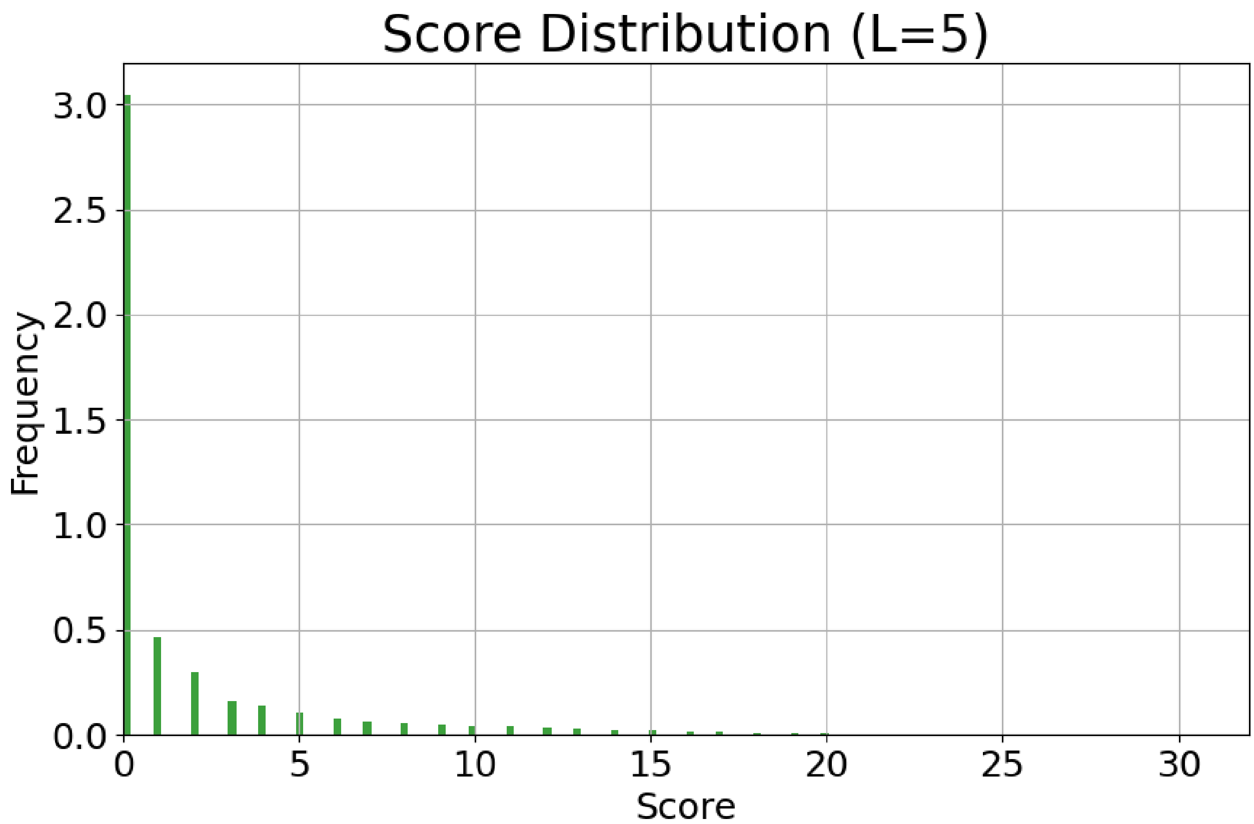
由上图可以看出，得分频率在分数低的时候概率高，随着分数的增加概率迅速降低，呈现指数衰减的趋势。

而这也可以理解。想要得到较高的得分需要随机游走增加的格子周围有不少的3，这对于系统是有一定要求的。

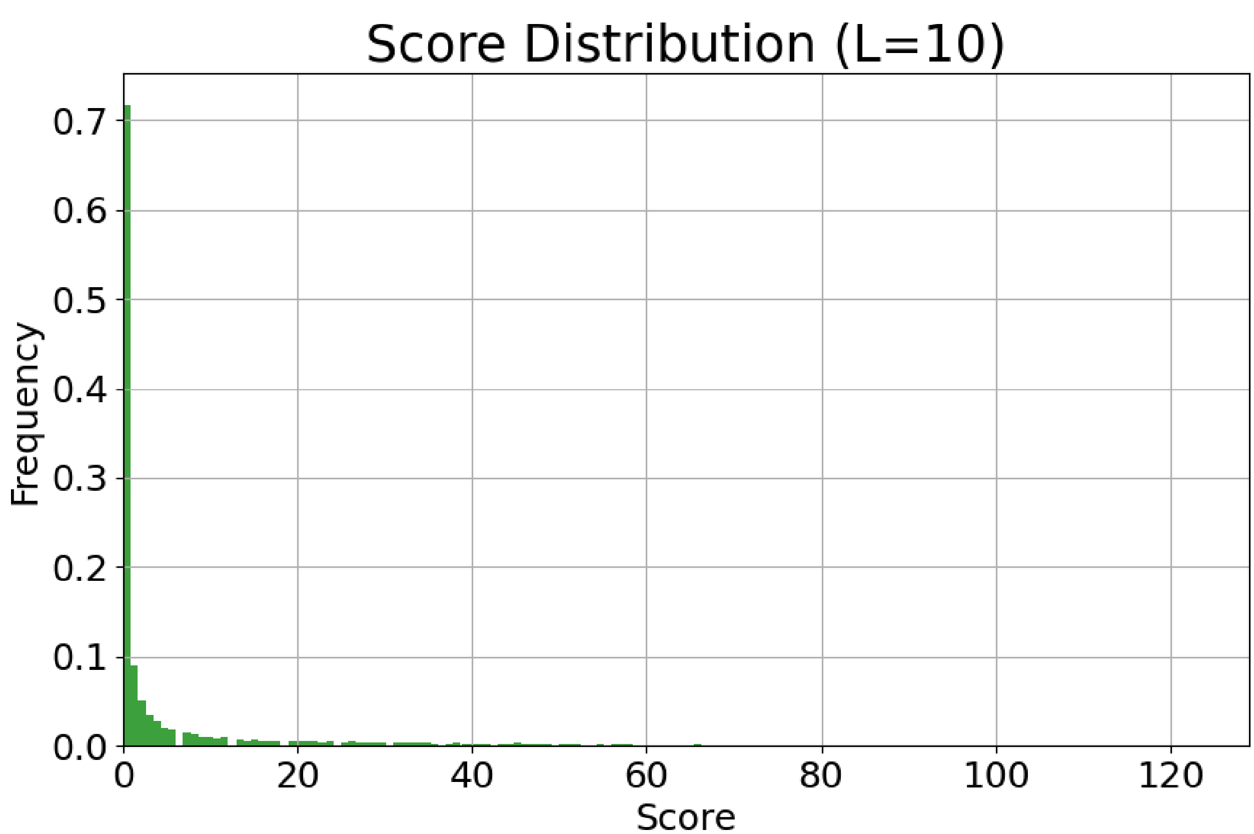
3. 选取多个不同的系统尺寸 L 重复（2）的过程

不妨选取$L=[5,10,1520,25,30,35,40]$，运行代码，得到得分的概率分布随着系统尺寸的变化：

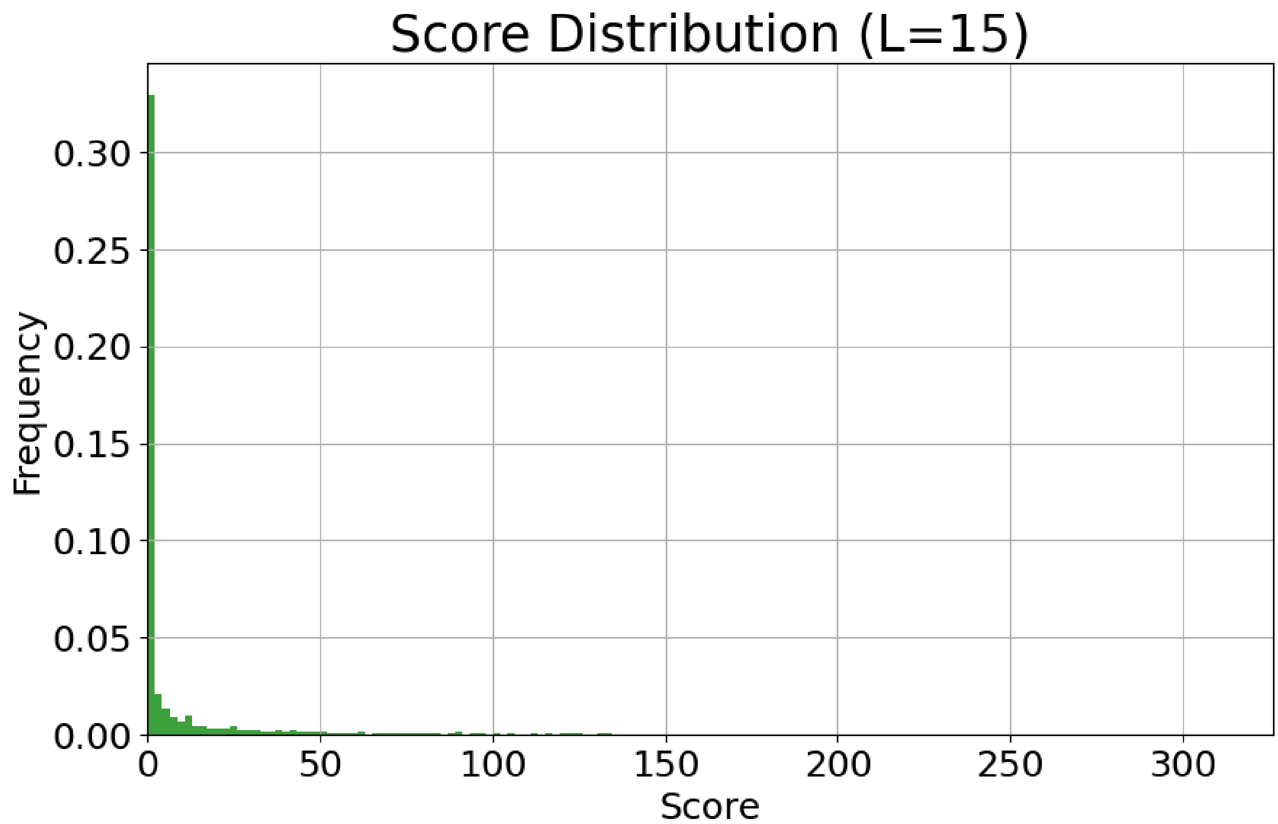
$L=5$:


Score Distribution (L=5)
Frequency
Score
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
5
10
15
20
25
30

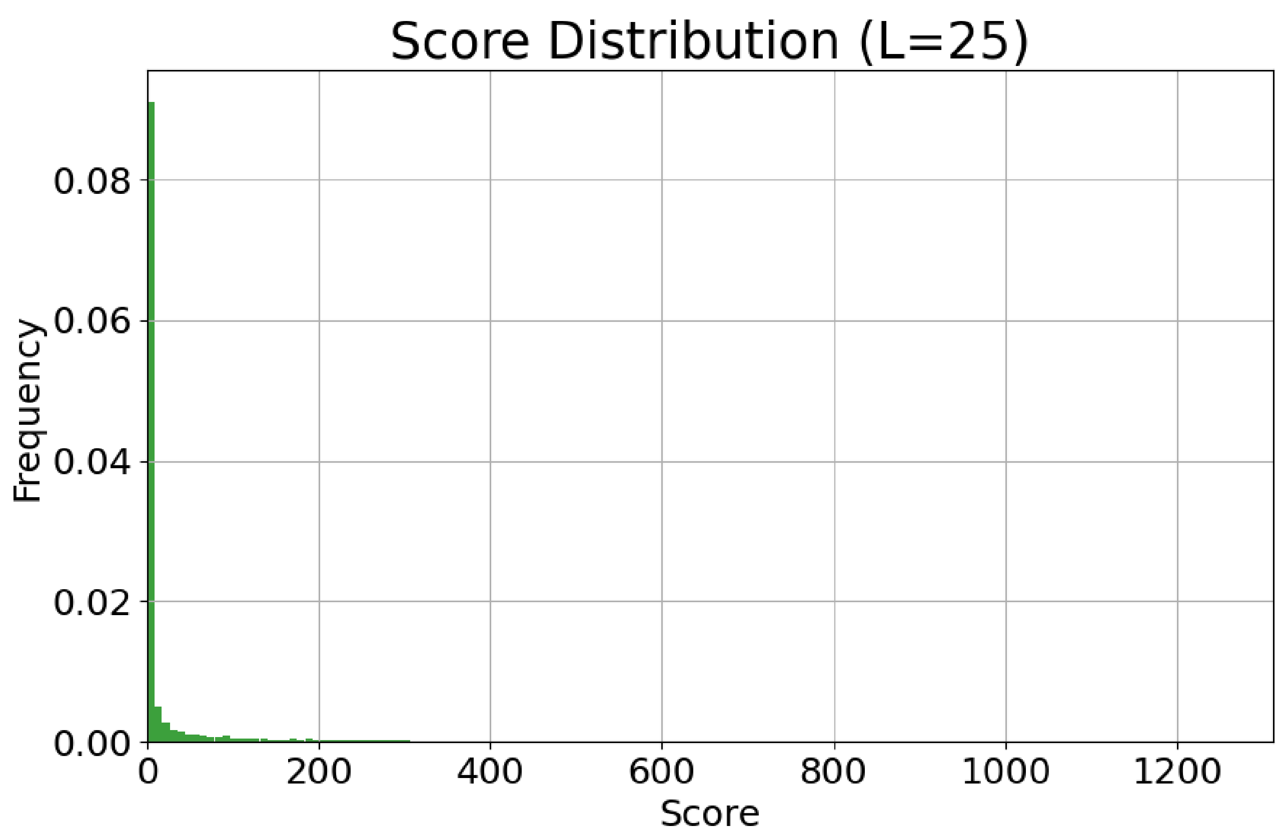
$L = 10$:


Score Distribution (L=10)
Frequency
Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
20
40
60
80
100
120

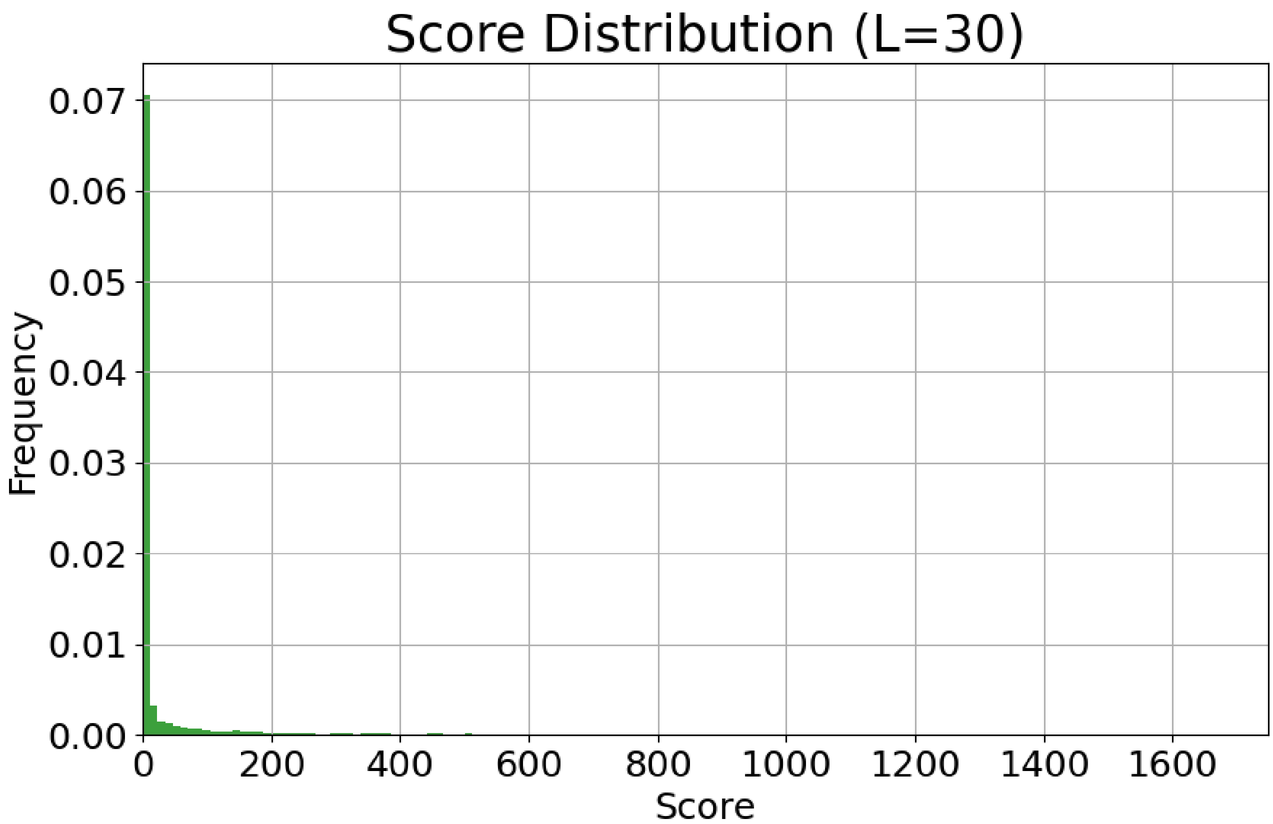
$L = 15$:


Score Distribution (L=15)
Frequency
Score
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0
50
100
150
200
250
300

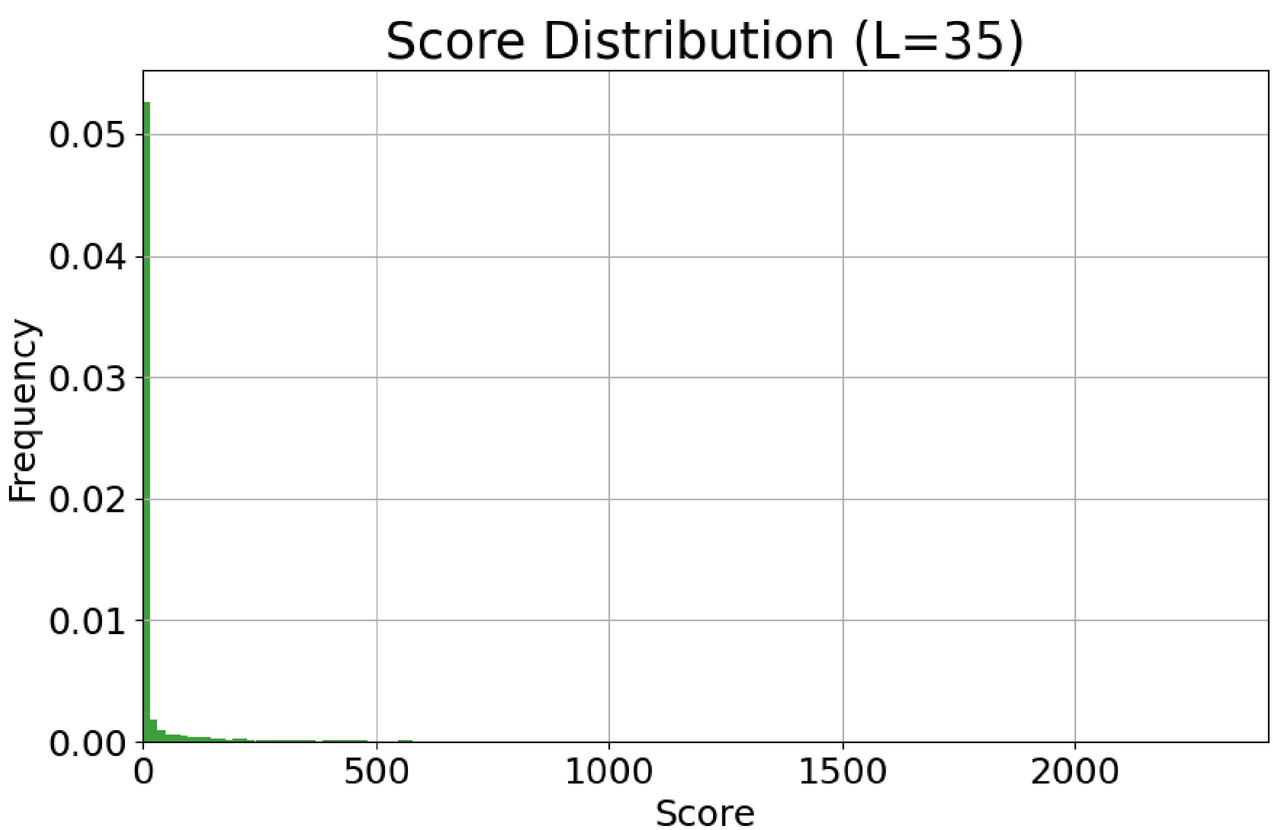
$L = 25$:


Score Distribution (L=25)
Frequency
Score
0.00
0.02
0.04
0.06
0.08
0
200
400
600
800
1000
1200

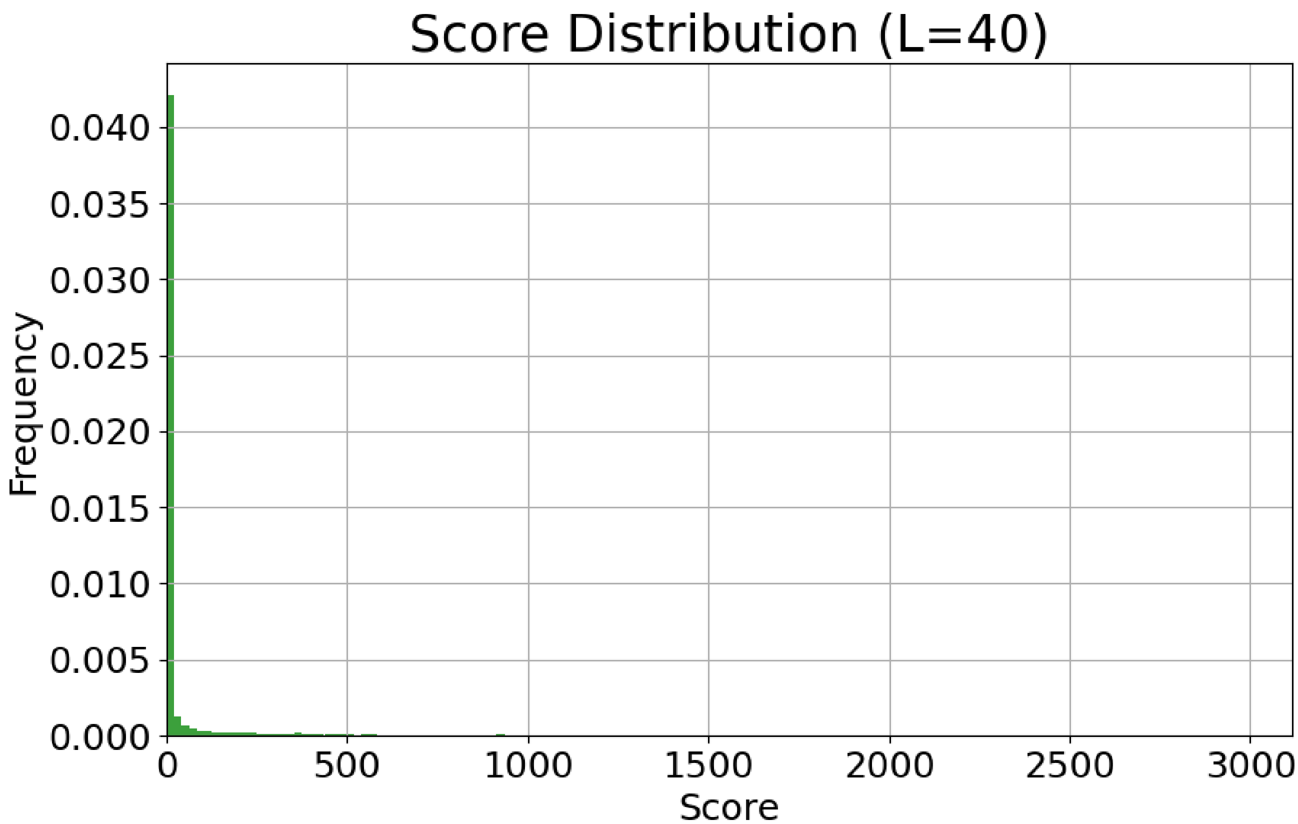
$L = 30$:


Score Distribution (L=30)
Frequency
Score
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
200
400
600
800
1000
1200
1400
1600

$L = 35$:


Score Distribution (L=35)
Frequency
Score
0.05
0.04
0.03
0.02
0.01
0.00
0
500
1000
1500
2000

$L = 40$:



此外，我统计了不同尺寸的网格在“消消乐”的过程中的最大得分，结果如下：

```
[Running] python -u "e:\大二下\computational physics\homework\hw_1\code\B_tree.py"
L=32,the maximum score is: 2090
L=5,the maximum score is: 32
L=10,the maximum score is: 129
L=15,the maximum score is: 326
L=20,the maximum score is: 534
L=25,the maximum score is: 1312
L=30,the maximum score is: 1748
L=35,the maximum score is: 2414
L=40,the maximum score is: 3115

[Done] exited with code=0 in 32.099 seconds
```

由上图可以看出，随着网格尺寸的增大，“消消乐”的最大得分越来越高，尤其是当网格尺寸达到40时，“消消乐”的最大得分达到3115。

附录：源码

A_1.py

```python
import numpy as np
import matplotlib.pyplot as plt

def f_1(mu,x):
    """
    定义第一问中的迭代方程
    """
    return 1-mu*x**2

def logistic_map(x0, mu, n_iter)->np.ndarray:
    """
    表示映射的迭代函数。
    输入三个参数，输出各代的x值。

    参数设置:
        x0: 初值，(0, 1) 区间内的浮点数。
        mu: 参数，(0, 4) 区间内的浮点数。
        n_iter: 迭代次数。

    Returns:
        一个包含迭代结果的 NumPy 数组。
    """
    x = np.zeros(n_iter)
    x[0] = x0
    for i in range(n_iter - 1):
        x[i+1] = f_1(mu, x[i])
    return x

# def plot_logistic_map(mu_values, x0, n_iter):
#     """
#     绘制不同 mu 值下的 Logistic 映射结果。

#     Args:
#         mu_values: 一个包含多个 mu 值的列表或 NumPy 数组。
#         x0: 初值。
#         n_iter: 迭代次数。
#     """
```

```python
#     plt.figure(figsize=(12, 6))
#     for mu in mu_values:
#         print
#         x = logistic_map(x0, mu, n_iter)
#         plt.plot(x, label=f"μ = {mu}")

#     plt.xlabel("Iteration")
#     plt.ylabel("x_n")
#     plt.title("Logistic Map")
#     plt.legend()
#     plt.grid(True)
#     plt.show()

def plot_logistic_map(mu_values, x0, n_iter):
    """
    绘制不同 mu 值下的 Logistic 映射结果。
    将不同mu值的结果画到不同的子图中
    """
    num_plots = len(mu_values)
    fig, axes = plt.subplots(num_plots, 1, figsize=(12, 6 * num_plots))

    print("debug")

    for i, mu in enumerate(mu_values):
        x = logistic_map(x0, mu, n_iter)
        axes[i].plot(x, label=f"μ = {mu}")
        axes[i].xaxis.set_tick_params(labelsize=18)
        axes[i].yaxis.set_tick_params(labelsize=18)
        axes[i].set_xlabel("Iteration",fontsize=18)
        axes[i].set_ylabel("x_n",fontsize=18)
        axes[i].grid(True)
        axes[i].legend()

    plt.title(f"$x_n$ vs Iteration for func 1 for different $\mu$ values, $x_0$ = {x0}",
              fontsize=19, y=10.5)
    plt.tight_layout()  # Adjust subplot parameters for a tight layout.
    plt.show()

mu_values = [0.01,0.3,0.6,1,1.3,1.6,1.99]
mu_values = [0.01,0.5,1,1.5,1.99]
x0 = 0.99
n_iter = 50
```

```python
plot_logistic_map(mu_values, x0, n_iter)


def plot_initial_condition_sensitivity(mu, x0_1, x0_2, n_iter):
    """
    绘制 Logistic 映射的初值敏感性。
    """
    x1 = logistic_map(x0_1, mu, n_iter)
    x2 = logistic_map(x0_2, mu, n_iter)

    plt.figure(figsize=(12, 6))
    plt.plot(x1, label=f"x0 = {x0_1}")
    plt.plot(x2, label=f"x0 = {x0_2}")

    plt.xlabel("Iteration",fontsize=18)
    plt.ylabel("x_n",fontsize=18)
    plt.title(f"Logistic Map - Initial Condition Sensitivity (μ = {mu})")
    plt.legend()
    plt.grid(True)
    plt.show()


mu = 3.9
x0_1 = 0.2
x0_2 = 0.2001
n_iter = 100

# plot_initial_condition_sensitivity(mu, x0_1, x0_2, n_iter)
```

A_2.py

```python
import numpy as np
import matplotlib.pyplot as plt

def f_2(mu,x):
    """
    定义第二问中的迭代方程
    """
    return np.cos(x) - mu*x**2

def logistic_map(x0, mu, n_iter)->np.ndarray:
    """
    表示映射的迭代函数。
    输入三个参数，输出各代的x值。

    参数设置:
        x0: 初值，(0, 1) 区间内的浮点数。
        mu: 参数，(0, 4) 区间内的浮点数。
        n_iter: 迭代次数。

    Returns:
        一个包含迭代结果的 NumPy 数组。
    """
    x = np.zeros(n_iter)
    x[0] = x0
    for i in range(n_iter - 1):
        x[i+1] = f_2(mu, x[i])
    return x

# def plot_logistic_map(mu_values, x0, n_iter):
#     """
#     绘制不同 mu 值下的 Logistic 映射结果。

#     Args:
#         mu_values: 一个包含多个 mu 值的列表或 NumPy 数组。
#         x0: 初值。
#         n_iter: 迭代次数。
#     """
#     plt.figure(figsize=(12, 6))
#     for mu in mu_values:
#         print
#         x = logistic_map(x0, mu, n_iter)
```

```python
#         plt.plot(x, label=f"μ = {mu}")

#     plt.xlabel("Iteration")
#     plt.ylabel("x_n")
#     plt.title("Logistic Map")
#     plt.legend()
#     plt.grid(True)
#     plt.show()

def plot_logistic_map(mu_values, x0, n_iter):
    """
    绘制不同 mu 值下的 Logistic 映射结果。
    将不同mu值的结果画到不同的子图中
    """
    num_plots = len(mu_values)
    fig, axes = plt.subplots(num_plots, 1, figsize=(12, 6 * num_plots))

    print("debug")

    for i, mu in enumerate(mu_values):
        x = logistic_map(x0, mu, n_iter)
        axes[i].plot(x, label=f"μ = {mu}")
        axes[i].xaxis.set_tick_params(labelsize=18)
        axes[i].yaxis.set_tick_params(labelsize=18)
        axes[i].set_xlabel("Iteration",fontsize=18)
        axes[i].set_ylabel("x_n",fontsize=18)

        axes[i].grid(True)
        axes[i].legend()

    plt.title(f"$x_n$ vs Iteration for func 2 for different $\mu$ values, $x_0$ = {x0}",
              fontsize=19, y=10.5)
    plt.tight_layout()  # Adjust subplot parameters for a tight layout.
    plt.show()

mu_values_1 = [0.01,0.3,0.6,1,1.3,1.6,1.99]
mu_values_2 = [0.01,0.5,1,1.5,1.99]
x0 = -0.99
n_iter = 50

plot_logistic_map(mu_values_1, x0, n_iter)
```

```python
def plot_initial_condition_sensitivity(mu, x0_1, x0_2, n_iter):
    """
    绘制 Logistic 映射的初值敏感性。
    """
    x1 = logistic_map(x0_1, mu, n_iter)
    x2 = logistic_map(x0_2, mu, n_iter)

    plt.figure(figsize=(12, 6))
    plt.plot(x1, label=f"x0 = {x0_1}")
    plt.plot(x2, label=f"x0 = {x0_2}")

    plt.xlabel("Iteration",fontsize=18)
    plt.ylabel("x_n",fontsize=18)
    plt.title(f"Logistic Map - Initial Condition Sensitivity (μ = {mu})")
    plt.legend()
    plt.grid(True)
    plt.show()


mu = 3.9
x0_1 = 0.2
x0_2 = 0.2001
n_iter = 100

# plot_initial_condition_sensitivity(mu, x0_1, x0_2, n_iter)
```

B.py

```python
import numpy as np
import matplotlib.pyplot as plt

def random_block(L,gridll):
    """
    该函数随机挑选一个格子，放上去一个方块
    """
    x = np.random.randint(0,L)
    y = np.random.randint(0,L)
    gridll[x][y] += 1
    return gridll



def delete_block(L,gridll,x,y):
    """
    该函数执行“消消乐”的步骤，消去当前格子上的方块
    并记录一次消去
    这四个方块会移动累加到上下左右的4个格点（它们各自方块数+1）
    """
    gridll[x][y] -= 4
    if x >=1:
        gridll[x-1][y] += 1
    if x < L-1:
        gridll[x+1][y] += 1
    if y >=1:
        gridll[x][y-1] += 1
    if y < L-1:
        gridll[x][y+1] += 1
    return gridll




def exist_4_or_more(L,gridll):
    """
    这个函数判断当前场上是否存在大于4的方块
    """

    for i in range(L):
        for j in range(L):
            if gridll[i][j] >= 4:
                return [True,i,j]
```

```python
    return [False,0,0]


def main(total_time,L) :
    """
    这个函数模拟整个游戏

    """
    gridll = np.zeros((L,L))
    t = 0
    score_list = []
    average_density = []
    while t < total_time:
        random_block(L,gridll)
        s = 0
        while exist_4_or_more(L,gridll)[0]:
            x = exist_4_or_more(L,gridll)[1]
            y = exist_4_or_more(L,gridll)[2]
            delete_block(L,gridll,x,y)
            s += 1
            print(s)
        score_list.append(s)
        t += 1
        average_density.append(np.sum(gridll)/(L*L))
    return gridll,score_list,average_density

def plot_score_distribution(score_list, L):
    """
    绘制得分的频率分布直方图，并进行归一化
    """
    plt.figure(figsize=(10, 6))
    n, bins, patches = plt.hist(score_list, bins=100,
                                density=True, facecolor='g', alpha=0.75)



    plt.xlabel('Score')
    plt.ylabel('Frequency')
    plt.xlim(0)
    plt.title(f'Score Distribution (L={L})')
    plt.grid(True)

    plt.show()
```

```python
def main_1(average_densityL):
    """
    这个函数画出平均密度随时间的变化图
    """
    plt.plot(average_density)
    plt.xlabel("Time")
    plt.ylabel("Average Density")
    plt.title(f"Average Density (L={L})")
    plt.show()

if __name__ == '__main__':
    L = 32
    total_time = 8000
    gridll,score_list,average_density = main(total_time,L)
    main_1(average_density)
    plot_score_distribution(score_list, L)
```

B_tree.py

```python
import numpy as np
import matplotlib.pyplot as plt
import os

def random_block(L,gridll):
    """
    该函数随机挑选一个格子，放上去一个方块
    """
    x = np.random.randint(0,L)
    y = np.random.randint(0,L)
    gridll[x][y] += 1
    return gridll

def random_block_xy(L,gridll):
    """
    该函数随机挑选一个格子，放上去两个方块
    返回girdll和方块的xy坐标
    """
    x = np.random.randint(0,L)
    y = np.random.randint(0,L)
    gridll[x][y] += 1
    return gridll,x,y



def delete_block(L,gridll,x,y):
    """
    该函数执行“消消乐”的步骤，消去当前格子上的方块
    并记录一次消去
    这四个方块会移动累加到上下左右的4个格点（它们各自方块数+1）
    """
    gridll[x][y] -= 4
    if x >=1:
        gridll[x-1][y] += 1
    if x < L-1:
        gridll[x+1][y] += 1
    if y >=1:
        gridll[x][y-1] += 1
    if y < L-1:
        gridll[x][y+1] += 1
    return gridll
```

```python
def exist_4_or_more(L,gridll):
    """
    这个函数判断当前场上是否存在大于4的方块
    """

    for i in range(L):
        for j in range(L):
            if gridll[i][j] >= 4:
                return [True,i,j]
    return [False,0,0]



def find_4_or_more(L,gridll,x,y,s):
    if gridll[x][y] < 4:
        return
    if gridll[x][y] >=4 :
        delete_block(L,gridll,x,y)
        s[0] += 1
        if x >=1:
            find_4_or_more(L,gridll,x-1,y,s)
        if x < L-1:
            find_4_or_more(L,gridll,x+1,y,s)
        if y >=1:
            find_4_or_more(L,gridll,x,y-1,s)
        if y < L-1:
            find_4_or_more(L,gridll,x,y+1,s)




def main(total_time,L) :
    """
    这个函数模拟整个游戏

    """
    gridll = np.zeros((L,L))
    t = 0
    score_list = []
```

```python
    average_density = []
    while t < total_time:
        gridll,x,y = random_block_xy(L,gridll)
        s = [0]
        find_4_or_more(L,gridll,x,y,s)

        score_list.append(s[0])
        t += 1
        average_density.append(np.sum(gridll)/(L*L))
    return gridll,score_list,average_density




def plot_score_distribution(score_list, L):
    """
    绘制得分的频率分布直方图，并进行归一化
    """
    plt.figure(figsize=(10, 6))
    n, bins, patches = plt.hist(score_list, bins=150,
                                density=True, facecolor='g', alpha=0.75)



    plt.xlabel('Score',fontsize=18)
    plt.ylabel('Frequency', fontsize=18)
    plt.xlim(0,max(score_list))
    plt.title(f'Score Distribution (L={L})',fontsize=25)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)
    plt.grid(True)

    plt.show()

def plot_score_distribution_and_save(score_list, L):
    """
    绘制得分的频率分布直方图，并进行归一化
    """
    plt.figure(figsize=(10, 6))
    n, bins, patches = plt.hist(score_list, bins=150,
                                density=True, facecolor='g', alpha=0.75)



    plt.xlabel('Score',fontsize=18)
```

```python
    plt.ylabel('Frequency', fontsize=18)
    plt.xlim(0,max(score_list))
    plt.title(f'Score Distribution (L={L})',fontsize=25)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)
    plt.grid(True)
    save_path = f"./figure/B_2_{L}.png"
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    plt.savefig(save_path)
    plt.show()


def main_1(average_density):
    """
    这个函数画出平均密度随时间的变化图
    """
    plt.plot(average_density)
    plt.xlabel("Time",fontsize=18)
    plt.ylabel("Average Density",fontsize=18)
    plt.title(f"Average Density (L={L})",fontsize=18)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)
    plt.show()

if __name__ == '__main__':
    L = 32
    total_time = 8000
    gridll,score_list,average_density = main(total_time,L)
    # main_1(average_density)
    plot_score_distribution(score_list, L)
    L_list = [5,10,15,20,25,30,35,40]
    print(f"L={L},the maximum score is:",max(score_list))
    for L in L_list:
        total_time = 8000
        gridll,score_list,average_density = main(total_time,L)
        print(f"L={L},the maximum score is:",max(score_list))
        plot_score_distribution_and_save(score_list, L)
```