2022 计算物理期末结课论文

祝茗
武汉大学

摘　　要

本文为 2022 计算物理课程的结课论文。内容主要包含了相变模型与元胞自动机。

1　背景介绍

在学习和使用了一个学期的 Python 后，已经可以使用 matplolib 的库来进行基础绘图。

在学习计算物理的过程中，我学到了许多有趣的知识与模型。其中最让我感兴趣的是有关 2D Ising model 的模拟，因为在 Ising model 通过简单的模型与简单的模拟，可以得到并不简单的相变过程，这也可以算是一种”more is difference” [1]。

在本学期的学习过程中，采用了 Monte Carlo 方法来模拟 2D Ising model ，但是在思索期末课程论文选题的过程中，我浏览到了一个同样可以用来描述相变过程的模型——Percolation: a Mathematical Phase Transition，不同于通过多次 Monte Carlo 方法实现的 2D Ising model, Percolation model 只需要一次随机的初始化就可以通过调节 percolation 的概率来获得一个单调递增的模拟结果，比 2D Ising Model 的实现更加的稳定与高效。在学习与实现 Percolation model 的过程中，我还留意到了另外一种模拟相变的模型——元胞自动机 (Cellular Automata)，也进行了简单的学习与实现。

2　Percolation 与 Cellular Automata

通过编写一个 `PercolationModel2D` 的类，来实现模型中网格的存储，详细的代码可以参见附录。由前人的探索，可知在 $p = 0.45$ 附近会发生相变，因此在一定范围内调整 p 的大小可以观察到相变的发生。通过先输出多张随时间演化的图片，后手动筛选演化终止的结点，选择合适的截止位置，然后将多张 png 格式的图片拼接为 gif 作为结果的呈现。

3　结论

在 $p = 0.45$ 附近会发生相变，percolation model 几乎可以占据整个空间。

4　附录

4.1　Class PercolationModel2D

```
import numpy as np
class PercolationModel2D(object):
"""
Object that calculates and displays behaviour of 2D cellular automata
"""

def __init__(self, ni):
"""
Constructe a 2D cellular automaton
input:
ni: number of cells in each direction
"""
self.N = ni                 # Number of cells in each direction
self.Ntot = self.N*self.N   # Total number of cells

self.grid = np.zeros((self.N, self.N))
self.nextgrid = np.zeros((self.N, self.N))
self.tested = np.zeros((self.N, self.N))

self.complete = False  # Boolean to indicate whether the model has completed,
                                          default is False

def getMooreNeighbourhood(self, i, j):
"""
Return a set of indices corresponding to the Moore Neighbourhood, i.e. the cells
                                          immediately adjacent to (i,j) and the
                                          cells diagonally adjacent to (i,j)
input:
`i`: row index
`j`: column index
output:
`indices`: list of indices corresponding to the Moore Neighbourhood
"""
indices = []
for iadd in range(i-1, i+2):
for jadd in range(j-1, j+2):
if (iadd == i and jadd == j):
continue  # exclude the cell itself

if (iadd > self.N-1):
iadd = iadd - self.N  # periodic boundary conditions
if (jadd > self.N-1):
jadd = jadd - self.N  # periodic boundary conditions

indices.append([iadd, jadd])
return indices
```

```
def getVonNeumannNeighbourhood(self, i, j):
"""
Return a set of indices corresponding to the Von Neumann Neighbourhood, i.e. the
                                        cells immediately adjacent to (i,j)
input:
`i`: row index
`j`: column index
output:
`indices`: list of indices corresponding to the Von Neumann Neighbourhood
"""
indices = []
for iadd in range(i-1, i+2):
if (iadd == i):
continue
if (iadd > self.N-1):
iadd = iadd - self.N
indices.append([iadd, j])

for jadd in range(j-1, j+2):
if (jadd == j):
continue
if (jadd > self.N-1):
jadd = jadd - self.N
indices.append([i, jadd])
return indices

def check_complete(self):
"""
Check if all cells have been tested
output:
`complete`: boolean, whether the model has completed
"""
ntested = np.sum(self.tested)   # number of cells that have been tested
if (ntested == self.N*self.N):  # if all cells have been tested
self.complete = True        # indicate the model has completed
return self.complete

def randomise(self):
"""
Place a random selection of zeros and ones into grid
"""
for i in range(self.N):
for j in range(self.N):
self.grid[i, j] = np.rint(np.random.random())

def randomise_with_symmetry(self):
```

```
"""
Place a random selection of zeros and ones into grid, with centual symmetry
"""
for i in range(self.N/2):
for j in range(self.N/2):
self.grid[i, j] = np.rint(np.random.random())
self.grid[i+self.N/2, j] = self.grid[i, j+self.N/2] = self.grid[i+self.N/2, j+
                                        self.N/2] = self.grid[i, j]

def clear(self, icentre, jcentre, extent):
"""
Clear a space on the grid
input:
`icentre`: row index of centre of space to clear
`jcentre`: column index of centre of space to clear
`extent`: extent of space to clear
"""
for i in range(icentre-extent, icentre+extent):
for j in range(jcentre-extent, jcentre+extent):
if (i > 0 and i < self.N and j > 0 and j < self.N):
self.grid[i, j] = 0

def updateGrid(self):
"""
Take the changes queued up on self.nextgrid, and applies them to self.grid
"""
self.grid = np.copy(self.nextgrid)
self.nextgrid = np.zeros((self.N, self.N))

def ApplyPercolationModelRule(self, P):
"""
Construct the self.nextgrid matrix based on the properties of self.grid
Applie the Percolation Model Rules:
1. Cells attempt to colonise their Moore Neighbourhood with probability P
2. Cells do not make the attempt with probability 1-P
"""
for i in range(self.N):
for j in range(self.N):
# If cell has already been tested
if (self.tested[i, j] == 1):
self.nextgrid[i, j] = self.grid[i, j]  # Copy value from self.grid to self.
                                        nextgrid
continue                               # Skip to next cell

# If cell contains a coloniser, then decide whether to colonise
if (self.grid[i, j] == 1 and self.tested[i, j] == 0):
# If colonisation occurs
```

```
if (np.random.rand() < P):
self.nextgrid[i, j] = 1
indices = self.getMooreNeighbourhood(i, j)

for element in indices:
if (self.tested[element[0], element[1]] == 1):
continue
if (self.grid[element[0], element[1]] == 0):
self.nextgrid[element[0], element[1]] = 1

else:
self.nextgrid[i, j] = -1  # If colonisation does not occur, then mark cell as
                                        tested

self.tested[i, j] = 1
```

4.2 Cellular Automata Patterns

```
"""
This file contains functions to add various patterns to the percolation model
"""
from PercolationModel import PercolationModel2D


def add_block(cell: PercolationModel2D, icentre: int, jcentre: int):
"""
Add a 2x2 block into the system, with bottom left corner (icentre, jcentre)
like this:
xx\n
xx\n
"""
extent = 2
cell.clear(icentre, jcentre, extent)                  # clear the space
cell.grid[icentre:icentre+2, jcentre:jcentre+2] = 1   # add the block


def add_beehive(cell: PercolationModel2D, icentre: int, jcentre: int):
"""
Add a beehive into the system, with (icentre, jcentre) being the inner left blank
                                        square
like this:
oxxo\n
xoox\n
oxxo\n
"""
extent = 7
```

```
    cell.clear(icentre, jcentre, extent)
    cell.grid[icentre-1, jcentre:jcentre+2] = 1  # top row
    cell.grid[icentre+1, jcentre:jcentre+2] = 1  # bottom row
    cell.grid[icentre, jcentre-1] = 1            # left dot
    cell.grid[icentre, jcentre+2] = 1            # right dot


def add_blinker(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a horizontal line of 3 blocks, a period 2 oscillator
    like this:
    xxx\n
    """
    extent = 4
    cell.clear(icentre, jcentre, extent)
    cell.grid[icentre, jcentre-1:jcentre+2] = 1


def add_loaf(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a loaf, with (icentre, jcentre) in the bottom left corner (blank)
    like this:
    ooxo\n
    oxox\n
    xoox\n
    oxxo\n
    """
    cell.grid[icentre, jcentre+1:jcentre+3] = 1
    cell.grid[icentre+1:icentre+3, jcentre+3] = 1
    cell.grid[icentre+1, jcentre] = 1
    cell.grid[icentre+2, jcentre+1] = 1
    cell.grid[icentre+3, jcentre+2] = 1


def add_boat(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a boat, with (icentre,jcentre) in the bottom left corner (blank)
    like this:
    xxo\n
    xox\n
    oxo\n
    """
    extent = 4
    cell.clear(icentre, jcentre, extent)

    indices = cell.getVonNeumannNeighbourhood(icentre, jcentre)
    for element in indices:
```

```
    cell.grid[element[0], element[1]] = 1

    cell.grid[icentre+1, jcentre-1] = 1


def add_toad(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a toad, a period 2 oscillator
    like this:
    xo\n
    xx\n
    xx\n
    ox\n
    """
    extent = 3
    cell.clear(icentre, jcentre, extent)
    cell.grid[icentre-1:icentre+2, jcentre] = 1
    cell.grid[icentre:icentre+3, jcentre-1] = 1


def add_beacon(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add two 2x2 blocks, which repeat a pattern of period 2
    like this:
    xxoo\n
    xxoo\n
    ooxx\n
    ooxx\n
    """
    extent = 3
    cell.clear(icentre, jcentre, extent)
    add_block(cell, icentre+2, jcentre)
    add_block(cell, icentre, jcentre+2)


def add_pulsar(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a pulsar, a period 3 oscillator
    """
    extent = 8
    cell.clear(icentre, jcentre, extent)

    # Start with inner cross
    # North
    cell.grid[icentre+2:icentre+5, jcentre+1] = 1
    cell.grid[icentre+2:icentre+5, jcentre-1] = 1
```

```
    # South
    cell.grid[icentre-4:icentre-1, jcentre+1] = 1
    cell.grid[icentre-4:icentre-1, jcentre-1] = 1

    # East
    cell.grid[icentre+1, jcentre+2:jcentre+5] = 1
    cell.grid[icentre-1, jcentre+2:jcentre+5] = 1

    # West
    cell.grid[icentre+1, jcentre-4:jcentre-1] = 1
    cell.grid[icentre-1, jcentre-4:jcentre-1] = 1

    # Now do surrounding bars - quadrant at a time
    cell.grid[icentre+6, jcentre+2:jcentre+5] = 1
    cell.grid[icentre+2:icentre+5, jcentre+6] = 1

    cell.grid[icentre-4:icentre-1, jcentre+6] = 1
    cell.grid[icentre-6, jcentre+2:jcentre+5] = 1

    cell.grid[icentre-6, jcentre-4:jcentre-1] = 1
    cell.grid[icentre-4:icentre-1, jcentre-6] = 1

    cell.grid[icentre+2:icentre+5, jcentre-6] = 1
    cell.grid[icentre+6, jcentre-4:jcentre-1] = 1


    def add_glider(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a glider, with (icentre,jcentre) being the bottom left tile (alive) - period
                                            4 oscillator
    like this:
    oxo\n
    oox\n
    xxx\n
    """
    cell.grid[icentre, jcentre:jcentre+3] = 1
    cell.grid[icentre+1, jcentre+2] = 1
    cell.grid[icentre+2, jcentre+1] = 1


    def add_spaceship(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a lightweight spaceship, with (icentre,jcentre) being the bottom left tile (
                                            alive) - period 4 oscillator
    like this:
    oxoox\n
    xoooo\n
```

```
    xooox\n
    xxxxo\n
    """
    cell.grid[icentre, jcentre:jcentre+4] = 1
    cell.grid[icentre:icentre+3, jcentre] = 1
    cell.grid[icentre+3, jcentre+1] = 1
    cell.grid[icentre+3, jcentre+4] = 1
    cell.grid[icentre+1, jcentre+4] = 1


    def add_glider_gun(cell: PercolationModel2D, icentre: int, jcentre: int):
    """
    Add a Gosper glider gun (period 30 oscillator) with (icentre,jcentre) being the
                                                bottom left tile (alive)
    """
    add_block(cell, icentre+5, jcentre+2)

    # Make first of inner patterns
    cell.grid[icentre+4:icentre+7, jcentre+12] = 1
    cell.grid[icentre+3, jcentre+13] = cell.grid[icentre+7, jcentre+13] = 1

    # cell.grid[icentre+2,jcentre+14] = cell.grid[icentre+8,jcentre+14] = 1
    cell.grid[icentre+2, jcentre+14:jcentre+16] = cell.grid[icentre+8, jcentre+14:
                                                jcentre+16] = 1
    cell.grid[icentre+5, jcentre+16] = 1
    cell.grid[icentre+3, jcentre+17] = cell.grid[icentre+7, jcentre+17] = 1
    cell.grid[icentre+4:icentre+7, jcentre+18] = 1
    cell.grid[icentre+5, jcentre+19] = 1

    # Now second pattern
    cell.grid[icentre+6:icentre+9, jcentre+22:jcentre+24] = 1
    cell.grid[icentre+5, jcentre+24] = cell.grid[icentre+9, jcentre+24] = 1
    cell.grid[icentre+4:icentre+6, jcentre+26] = cell.grid[icentre+9:icentre+11,
                                                jcentre+26] = 1

    add_block(cell, icentre+7, jcentre+36)
```

参考文献

[1] P. W. Anderson. “More Is Different: Broken Symmetry and the Nature of the Hierarchical Structure of Science.” In: *Science* 177.4047 (Aug. 4, 1972), pp. 393–396. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.177.4047.393.