用 Monte Carlo 方法模拟二维 Ising 模型

一、Ising 模型简介

Ising 模型是描述物质相变的一种模型。物质经过相变，要出现新的结构和物性。发生相变的系统一般是在分子之间有较强相互作用的系统，又称合作系统。它是一个简单且被广泛使用的物理模型，下面就对从二维 Ising 模型的基本结构进行简单介绍。

“Ising 模型是一个非常简单的模型,在一维、二维或三维的每个格点上占据一个自旋。自旋是电子的一个内禀的性质,每个自旋在空间有两个量子化的方向,即其指向可以向上或向下。” 如上所说，一个 n 维的 Ising 模型，在每个格点上占据一个自旋，指向向上或向下，与周围的自旋相互作用，Ising 模型只考虑最近邻自旋的作用，因而每个自旋与周围自旋相互作用的强弱与周围最近邻子自旋数有关，即随着维度的增加,每个自旋的最近邻自旋数增加,与周围自旋的相互作用也在增强。下面简要介绍正方 Ising 模型模型的基本结构。如图 1，一个正方格子，每个格点代表一个自旋(向上(+1)或向下(-1))，仅与周围上下左右最近邻的四个自旋有相互作用，这就是，一个简单的正方二维伊辛模型。在求解一些物理量或者解决一些问题时，就以这个模型为基础，根据需要确定格子大小，初始自旋指向，然后以仅考虑最近邻相互作用为基础，根据需要，设定条件，处理问题。这就是最基本的思想。

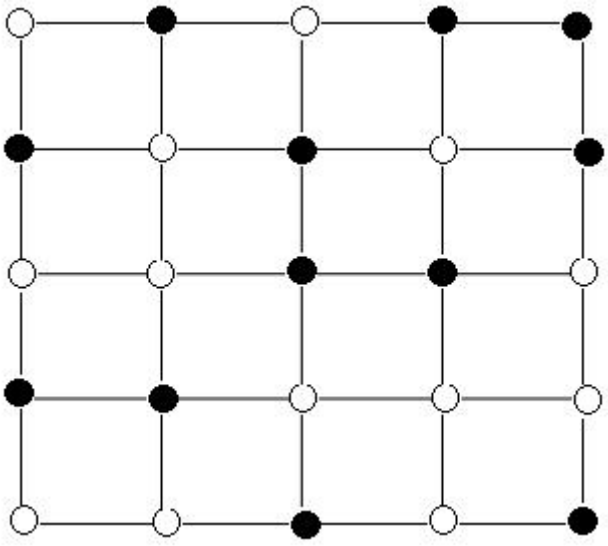


图 1

其中空心圆点代表自旋向上，实心圆点代表自旋向下。

二、Monte Carlo 方法

Monte-Carlo 模拟法,是一种随机模拟方法，以概率和统计理论方法为基础的一种计算方法，是使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。将所求解的问题同一定的概率模型相联系，用电子计算机实现统计模拟或抽样，以获得问题的近似解。为象征性地表明这一方法的概率统计特征，故借用赌城蒙特卡罗命名。下面介绍 Monte-Carlo 方法的其基本思想以及文中所涉及到的相关理论。

Monte-Carlo 方法的基本思想是当所求解问题是某种随机事件出现的概率，或者是某个

随机变量的期望值时，通过某种“实验”的方法，以这种事件出现的频率估计这一随机事件的概率，或者得到这个随机变量的某些数字特征，并将其作为问题的解。

由概率定义知，某事件的概率可以用大量试验中该事件发生的频率来估算，当样本容量足够大时，可以认为该事件的发生频率即为其概率。因此，可以先对影响其可靠度的随机变量进行大量的随机抽样，然后把这些抽样值一组一组地代入功能函数式，确定结构是否失效，最后从中求得结构的失效概率。Monte-Carlo 方法正是基于此思路进行分析的。

Monte-Carlo 方法回避了结构可靠度分析中的数学困难，不管状态函数是否非线性、随机变量是否非正态，只要模拟的次数足够多，就可得到一个比较精确的失效概率和可靠度指标，于是蒙特卡罗方法为很多以前难以解决和分析的问题提供了一种思想，使他们得到解决和进展，是 Monte-Carlo 方法称为一种基础而重要的思想。

三、计算理论

如图 2[5],是某一线度为 N 的方形二维伊辛模型的某一格点的自旋（S0）及其周围最近邻的四个自旋（S1,S2,S3,S4）,格子上伊辛自旋模型自旋取向分布的一个状态， 或叫自旋组态，Si 取值为+1 或-1，即自旋向上或向下，称为伊辛变量。

由于只考虑每个自旋周围最近邻的四个自旋，那么 Si 与其周围最近邻自旋 Sj 相互作用能为 $E=-J\sum_{j}S_iS_j$ ，那么伊辛模型系统的哈密顿量为：

$$H(S)=-J\sum_{<ij>}S_iS_j$$

其中 J 为耦合常数，与物质有关。

同时系统的磁化强度为： $M=\frac{1}{N}\sum_{i}S_i$ 。

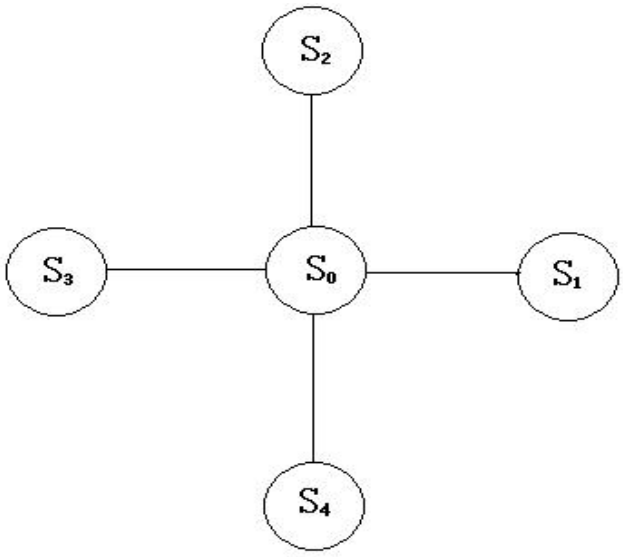


图 2　二维方形伊辛模型的某一格点的自旋及其最近邻自旋

四、程序展示

本程序利用 Python 语言的 Tkinter GUI 库模拟了 Ising 模型二维动态分布。为了模拟该分布，我们借助所谓的 Metropolis 算法，遵循以下步骤来生成近似于 Boltzmann 分布的分布：

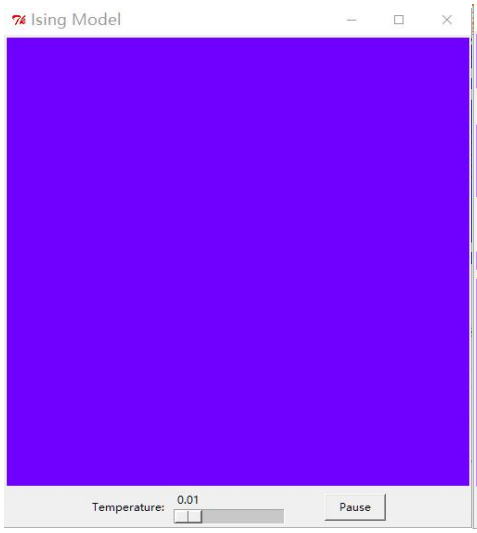
初始化系统磁矩取向，设定温度 T、磁场 H 等的值。

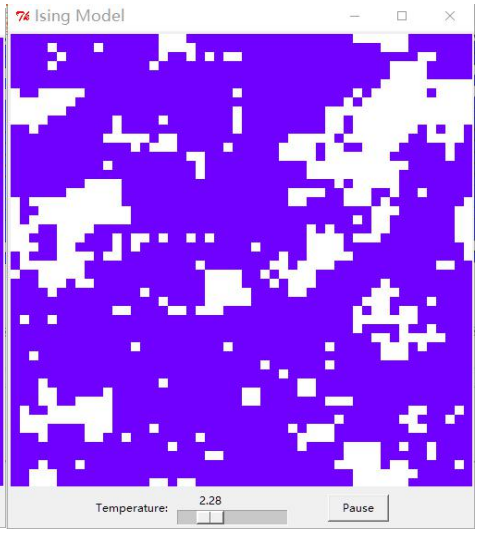
随机让一个磁矩翻转，计算系统总能量的改变 ΔE 。

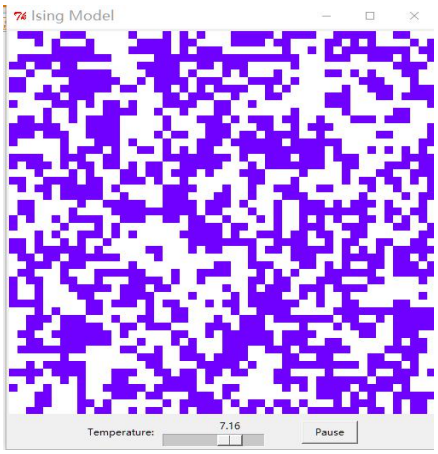
如果 $\Delta E > 0$ ，接受此次翻转；若 $\Delta E < 0$, 令 Boltzman 因子为 $P = \exp(-\Delta E / kT)$ 这样的取值，并生成一个在 $[0,1]$ 均匀分布的伪随机数 ξ ，若 $\xi < P$ 则接受翻转，否则拒绝翻转。

下图为程序默认界面：

运行程序：







以上三幅图片展示了 50×50 格子的二维 Ising 模型的顺磁相变过程，程序中取 Boltzmann 常数为 1 ，磁矩之间的耦合强度 $J=1$，外磁场 $H=0$。将 2500 个磁矩画在平面上，深色表示磁矩 $s=1$，浅色则相反。分析可得临界温度 $T_C=2.27$，第一幅图展示当温度低于临界温度时，磁矩绝大部分是取向相同的，显示出铁磁性。第二幅图显示当温度略超过临界温度后，磁矩无序性开始增加，有很多磁矩翻转了过来。第三幅图显示当温度进一步升高后，磁矩取向的随机性更高， 磁矩正反向的数目平分秋色，系统宏观不显磁性，即发生了顺磁相变。

参考文献

Nicholas J. Goordano,Hisao Nakanishi.《computational physics》.清华大学出版社
Wesley J. Chun.《Python Core Programming》.人民邮电出版社
《Ising model-Wikipedia, the free encyclopedia》:https://en.wikipedia.org/wiki/Ising_model

Source code:

```
"""
@author: thy
"""

# Simulates the two-dimensional Ising model using the Metropolis algorithm

import Tkinter, numpy, random, math

size = 50                                  # number of sites in a lattice row
squareWidth = 10                           # width of one site in pixels
canvasWidth = size * squareWidth       # full width of canvas in pixels
s = numpy.ones((size, size), int)      # 2D array of dipoles (1=up, -1=down)
running = False                            # will be true when simulation is running
M = []

theWindow = Tkinter.Tk()                   # create the GUI window
theWindow.title("Ising Model")
theWindow.geometry('+50+50')

theCanvas = Tkinter.Canvas(theWindow, width=canvasWidth, height=canvasWidth)
theCanvas.pack()
theImage = Tkinter.PhotoImage(width=canvasWidth, height=canvasWidth)
theCanvas.create_image((3, 3), image=theImage, anchor="nw", state="normal")
# The coordinates (3, 3) are a kludge to eliminate a mysterious offset that occurs otherwise.

# Function called when Start/Stop button is pressed:
def startStop():
```

```
    global running
    running = not running
    if running:
        goButton.config(text="Pause")
    else:
        goButton.config(text="Resume")

# Create the GUI controls:
controlFrame = Tkinter.Frame(theWindow)
controlFrame.pack()
tLabel = Tkinter.Label(controlFrame, text="Temperature: ")
tLabel.pack(side="left")
tSlider = Tkinter.Scale(controlFrame, from_=0.01, to=10.0, resolution=0.01, length=120,
orient="horizontal")
tSlider.pack(side="left")
tSlider.set(2.27)                                   # set to critical temperature initially
spacer = Tkinter.Frame(controlFrame, width=40)
spacer.pack(side="left")
goButton = Tkinter.Button(controlFrame, text="Start", width=8, command=startStop)
goButton.pack(side="left")

# Function to color the square representing site (i,j):
def colorSquare(i, j):
    theColor = "#7000ff" if s[i,j]==1 else "#ffffff"      # purple and white
    theImage.put(theColor,
to=(i*squareWidth,j*squareWidth,(i+1)*squareWidth,(j+1)*squareWidth))

# Function to calculate energy change upon hypothetical flip (with pbc):
def deltaE(i,j):
    leftS = s[size-1,j] if i==0 else s[i-1,j]
    rightS = s[0,j] if i==size-1 else s[i+1,j]
    topS = s[i,size-1] if j==0 else s[i,j-1]
    bottomS = s[i,0] if j==size-1 else s[i,j+1]
    return 2.0 * s[i,j] * (leftS + rightS + topS + bottomS)

# Main simulation "loop" schedules a call to itself upon completion:
def simulate():
    if running:
        T = tSlider.get()                           # get the current temperature
        for step in range(1000):
            i = int(random.random()*size)       # choose a random row and column
            j = int(random.random()*size)
            eDiff = deltaE(i,j)
            if eDiff <= 0 or random.random() < math.exp(-eDiff/T):       # Metropolis!
```

```
                    s[i,j] = -s[i,j]
                    alls =0 + s[i,j]
                    colorSquare(i, j)
                    M.append(alls /250)
    theWindow.after(1,simulate)                    # come back in one millisecond

# Initialize to a random array, and draw it as we go:
for i in range(size):
    for j in range(size):
        s[i,j] = 1 if random.random()<0.5 else -1
        colorSquare(i,j)

simulate()
theWindow.mainloop()
```