附录1

```
from tqdm import tqdm
import numpy as np

'''
定义了类lattice,有三个参数表示尺寸size，耦合常数Jconst，磁矩取向direct
'''
class lattice:
    def __init__(self,Jconst,size):
        self.size = int(size)
        self.Jconst = Jconst
        self.direct = np.ones([self.size,self.size],dtype=bool)

    def random_direct(self):
        self.direct = np.random.randint(0,2,(self.size,self.size))

    def negative_direct(self):
        self.direct = np.zeros((self.size,self.size),dtype = bool)

#定义了left,right,up,down,表示某一磁矩的邻近
    def left(self,x,y):
        if x != 0:
            return [x-1,y]
        else:
            return [self.size-1,y]

    def right(self,x,y):
        if x != self.size-1:
            return [x+1,y]
        else:
            return [0,y]

    def up(self,x,y):
        if y != 0:
            return [x,y-1]
        else:
            return [x,self.size-1]

    def down(self,x,y):
        if y != self.size-1:
            return [x,y+1]
        else:
            return [x,0]

#定义了单磁矩的能量oneE,翻转磁矩的能量变化deltaE
    def oneE(self,x,y):
        [leftx,lefty] = self.left(x,y)
        [rightx,righty] = self.right(x,y)
        [upx,upy] = self.up(x,y)
        [downx,downy] = self.down(x,y)
        if self.direct[x,y]*self.direct[leftx,lefty] == 1 or (self.direct[x,y] == 0 and
self.direct[leftx,lefty] == 0):
            leftE = 1
        else:
```

```
            leftE = -1
        if self.direct[x,y]*self.direct[rightx,righty] == 1 or (self.direct[x,y] == 0 and
self.direct[rightx,righty] == 0):
            rightE = 1
        else:
            rightE = -1
        if self.direct[x,y]*self.direct[upx,upy] == 1 or (self.direct[x,y] == 0 and
self.direct[upx,upy] == 0):
            upE = 1
        else:
            upE = -1
        if self.direct[x,y]*self.direct[downx,downy] == 1 or (self.direct[x,y] == 0 and
self.direct[downx,downy] == 0):
            downE = 1
        else:
            downE = -1
        return -self.Jconst*(leftE+rightE+upE+downE)

    def deltaE(self,x,y):
        return -4*self.oneE(x,y)

#定义了总磁化强度totalM,总能量totalE,平均磁化强度aveM,平均能量aveE
    def totalM(self):
        M = 0
        for x in range(self.size):
            for y in range(self.size):
                if self.direct[x,y] == 0 :
                    M -= 1
                else:
                    M += 1
        return M

    def totalE(self):
        E = 0
        for x in range(self.size):
            for y in range(self.size):
                E += self.oneE(x,y)
        return E

    def aveM(self):
        return self.totalM()/(self.size*self.size)

    def aveE(self):
        return self.totalE()/(self.size*self.size)

# 马尔可夫链蒙特卡洛算法（Markov Chain Monte Carlo）定义了单磁针旋转的蒙特卡洛算法singleMC

    def singleMC(self,temperature):

        #随机抽取一格点

        x = np.random.randint(0,self.size)
        y = np.random.randint(0,self.size)

        #计算能量，并以一定概率翻转
        if self.deltaE(x,y) < 0 :
            self.direct[x,y] = not self.direct[x,y]
        else:
```

```
        randomnum = np.random.rand()
        if randomnum < np.exp(-self.deltaE(x,y)/temperature):
            self.direct[x,y] = not self.direct[x,y]


def clustWolff(self,temperature):
    x = np.random.randint(0, self.size)
    y = np.random.randint(0, self.size)
    #定义一个列表种子栈，它代表具有成键可能的晶格，如果列表为空，则这次迭代结束
    possible = []
    possible.append((x,y))
    p = 1 - np.exp(-2*self.Jconst/temperature)
    clust = np.ones((self.size,self.size),dtype=bool)
    clust[x,y] = False
    #如果种子栈非空，则对非空的种子判断周边的晶格能否成键
    while len(possible) != 0:
        seedx,seedy = possible.pop()

        #左邻
        [leftx,lefty] = self.left(seedx,seedy)
        if self.direct[leftx,lefty] == self.direct[seedx,seedy]:
            randomnum = np.random.rand()
            if randomnum < p and clust[leftx,lefty]:
                possible.append((leftx,lefty))
                clust[leftx,lefty] = False

        #右邻
        [rightx,righty] = self.right(seedx,seedy)
        if self.direct[rightx,righty] == self.direct[seedx,seedy]:
            randomnum = np.random.rand()
            if randomnum < p and clust[rightx,righty]:
                possible.append((rightx,righty))
                clust[rightx,righty] = False

        #上邻
        [upx,upy] = self.up(seedx,seedy)
        if self.direct[upx,upy] == self.direct[seedx,seedy]:
            randomnum = np.random.rand()
            if randomnum < p and clust[upx,upy]:
                possible.append((upx,upy))
                clust[upx,upy] = False

        #下邻
        [downx,downy] = self.down(seedx,seedy)
        if self.direct[downx,downy] == self.direct[seedx,seedy]:
            randomnum = np.random.rand()
            if randomnum < p and clust[downx,downy]:
                possible.append((downx,downy))
                clust[downx,downy] = False

        #翻转该种子
        self.direct[seedx,seedy] = not self.direct[seedx,seedy]

def umbrella_sampling(self,M,windows,steps,temperature):
    M_min = int(M - windows/2)
    M_max = int(M + windows/2)
    self.negative_direct()
    dic_totalM = {}
```

```
        for i in range(200 + int(M/2) ):
            x = np.random.randint(0, self.size)
            y = np.random.randint(0, self.size)
            while self.direct[x,y] == True:
                x = np.random.randint(0, self.size)
                y = np.random.randint(0, self.size)
            self.direct[x,y] = not self.direct[x,y]
        print(self.totalM())
        for i in range(M_min + 2,M_max,2):
            dic_totalM[i] = 0
        for k in tqdm(range(steps)):
            flag = 1
            while flag == 1:
                x = np.random.randint(0,self.size)
                y = np.random.randint(0,self.size)
                self.direct[x,y] = not self.direct[x,y]
                if self.totalM() < M_min  or self.totalM() > M_max :
                    self.direct[x,y] = not self.direct[x,y]
                else:
                    self.direct[x,y] = not self.direct[x,y]
                    if self.deltaE(x,y) < 0 :
                        self.direct[x,y] = not self.direct[x,y]
                        if self.totalM() >M_min and self.totalM() < M_max:
                            flag = 0
                            dic_totalM[self.totalM()] += 1
                    else:
                        randomnum = np.random.rand()
                        if randomnum < np.exp(-self.deltaE(x,y)/temperature):

                            self.direct[x,y] = not self.direct[x,y]
                            if self.totalM() >M_min and self.totalM() < M_max:
                                flag = 0
                                dic_totalM[self.totalM()] += 1

        return dic_totalM
```