```python
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 16 09:13:39 2019

@author: Alexm
"""
#CODE 1: ISING MODEL WITH NO EXTERNAL MAGNETIC FIELD
#USED TO DEMONSTRATE THE ROLE OF EXCHANGE ENERGY, J


#importing the the necessary functions
from __future__ import division
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

print('Alexandra Mulholland 17336557')
print('Part 1: Varying the value of J with either no external magnetic field or a constant external mf applied')
J=-1
print('NO MF, J=%d'%J)

nt = 60       #number of steps in the temperature range
N= 10       #size of the lattice is NxN
stepsequil = 1000    #number of Monte Carlo sweeps to equilibriate
stepsmoncar = 1000     #number of Monte Carlo sweeps used for calculation of the different parameters-- energy, heat capacity etc
T = np.linspace(1.0, 10.0, nt); #defining the temperature range and in how many steps, nt

E= np.zeros(nt)
M= np.zeros(nt)
C= np.zeros(nt)
X = np.zeros(nt) #initialising each variable to zero array

n1 =  1.0/(stepsmoncar*N*N)
#later, we will divide energy and magenetization by the number of equil. steps
#and by the number of lattice points, NxN
n2 = 1.0/(stepsmoncar*stepsmoncar*N*N)
#we will also divide the energy and magnetisation squared by NxN and the number of steps squared
#in order to find specific heat and susceptibility
#explained in the report


#Firstly, generate the NxN lattice of randomly orientated spins
def randomstate(N):
    state = 2*np.random.randint(2, size=(N,N))-1
    return state
#np.random.randint: the second 2 here says you have 2 choices- these can be 1 or 0
#like binary of a computer
#multiplying this by 2 and taking away 1-- if it picks 1 you get 1 as product
#if it picks 0, you get -1 as a product. Hence all the lattice points are
#either 1 or -1 spin

#setting the external magnetic field to be zero and exchange energy to be 1
#J>0 means it is a ferromagentic material (explained in the report)
magf=0.0
J=-1
#defining the Monte Carlo algorithm
def moncar(config, beta):
    for i in range(N):
        for j in range(N):
                a = np.random.randint(0, N)
                b = np.random.randint(0, N) #selecting a random coordinate in lattice (a,b)
                s =  config[a, b] #defining the spin of this coordinate
                spinmag=s*magf #second term of energy equation- spin multiplied by h
                naybor = config[(a+1)%N,b] + config[(a-1)%N,b] + config[a,(b+1)%N] + config[a,(b-1)%N]
                #modulo function ensures each edge is neighbours with the opposite edge- lattice becomes a donut
                #as modulo divides and returns the value of the remainder
                #eg N=4 so (a+1)%N for a=0,1,2,3 the result is 1,2,3,0- makes the lattice periodic
                EC = 2*J*s*naybor+2*spinmag #from energy equation, this is the energy change value (2 times the spin-product)
                if EC < 0: #if energy change is less than zero, flip the spin
                    s *= -1
                elif rand() < np.exp(-EC*beta): #if not, the rand() generates a random value between 0 and 1
                    s *= -1 #if the rand() value is less than the boltzmann prob, the spin if flipped
                config[a, b] = s #the final spin value of this coordinate is produced and the MC is looped over again
    return config
    #after the MC has been interated through, the final, optimum configuration is produced
#s *= -1 is s=s*(-1)
#want the final system to be of lowest energy value
#2*spinmag-- multiply by 2 for the same reason you multiply the spin product by 2:
#the menergy value after minus the energy value beforeis just 2 times the energy value before the flip

#calculating the total energy for a given configuration (ie summing the energies of each lattice point)
#len(config) returns the number of items within the rows and columns of configuration
def TotEnergy(config):
    energy = 0 #initialising the energy value
    for i in range(len(config)):
        for j in range(len(config)):
            S = config[i,j] #again selecting a random coordinate and finding its spin
            spinmag=S*magf #external magnetic field, see equation in writeup- same as above
            naybor = config[(i+1)%N, j] + config[(i-1)%N,j] + config[i, (j+1)%N] + config[i,(j-1)%N] #spin-products with neighbours
            #neighbour to the right, neighbour to the left, neighbour above, neighbour below
            energy += -naybor*J*S/4-spinmag #summing over the enture configuration
    return energy #can see we divide by 4, must recognise that the spin-product means that each point is accounted for a total of 4 times
#we only want each to be counted once, hence divide by 4
#energy += ... is energy = energy + ...

#now calculating the sum of the magnetisation over the whole lattice
#this is the sum of the spin products- magnetisation is this divided by the number of points
#therefore, this is not the magnetisation, just the sum of the spins
#but we will use it to find the magnetisation
def TotSpin(config):
    mag = np.sum(config) #sum of spins in configuration, eg for a 5 by 5 lattice..
    #this will be either 25 or -25 for a ferromagnetic material
    return mag



for tt in range(nt):
    avE = avM = avE2 = avM2 = 0 #initialising each value to zero
    config = randomstate(N) #again, starting with a random config, the code will
    #loop through the monte carlo
    BT=1.0/T[tt]
    BT2=BT*BT
#BT is beta, which is 1/kT where k is the Boltzmann and is one in this case

    for i in range(stepsequil): #cofiguring to optimal state
        moncar(config, BT)

    for i in range(stepsmoncar):
        moncar(config, BT)
        Ene = TotEnergy(config) #we are calculating the average energy over the last
        #stepsmoncar configurations
        Mag = TotSpin(config) #we are calculating the average magnetisation for the
        #last stepsmoncar configurations (see below)
        #and plotting both against temperature
        #we must average over the last stepsmoncar in order to get error values
        #which will subsequently be used to find susceptibility and heat capacity

        avE = avE + Ene #now total energy averaged over stepsmoncar configurations
        avM = avM + Mag #total mag averaged over stepsmoncar configs
        avM2 = avM2 + Mag*Mag
        avE2 = avE2 + Ene*Ene

    E[tt] = n1*avE  #average energy over each temperature
    M[tt] = n1*avM  #average magnetisation over temperature range
    C[tt] = (n1*avE2 - n2*avE*avE)*BT2 #average heat capacity over temperature range
    X[tt] = (n1*avM2 - n2*avM*avM)*BT #average susceptibility over temperature range
#specific heat capacity is the (error in the energy) squared over temperature squared
#susceptibility is the (error in the magnetisation) squared over temperature

```

```python
#plotting the variables against temperature range

plt.figure(1)
plt.plot(T, E,'d', color='k',
    markersize=7,
    markerfacecolor='white',markeredgecolor='k',
    markeredgewidth=1)
plt.suptitle('Plot of the average energy versus temperature', fontsize=13)
plt.title('MF=0T and J=%d'%J, fontsize=10)
plt.xlabel("Temperature, T / K", fontsize=12);
plt.ylabel("Energy/J ", fontsize=12); plt.axis('tight');



plt.figure(2)
#plotting the absolute value of the magnetisation as this can be negative (1 or -1)
plt.plot(T, M,'d', color='winter',
    markersize=7,
    markerfacecolor='white',markeredgecolor='grey',
    markeredgewidth=1)
plt.suptitle('Plot of the average magnetisation versus temperature', fontsize=13)
plt.title('MF=0T and J=%d'%J, fontsize=10)
plt.xlabel("Temperature, T / K", fontsize=12);
plt.ylabel("Magnetisation / Am^-1", fontsize=12); plt.axis('tight');


#specific heat
plt.figure(3)
plt.plot(T, C,'d', color='c',
    markersize=7,
    markerfacecolor='white',markeredgecolor='c',
    markeredgewidth=1)
plt.suptitle('Plot of the specific heat versus temperature', fontsize=13)
plt.title('MF=0T and J=%d'%J, fontsize=10)
#max_y = max(C)  # Find the maximum y value
#max_x = T[C.av.index(max_y)]  # Find the x value corresponding to the maximum y value
#print(max_x, max_y)
plt.xlabel("Temperature, T / K", fontsize=12);
plt.ylabel("Heat capacity C / JK^-1 ", fontsize=12); plt.axis('tight');


#suscpetibility- measure of the error in the magnetisation
plt.figure(4)
plt.plot(T, X, 'd',color='blue',
    markersize=7,
    markerfacecolor='white',markeredgecolor='blue',
    markeredgewidth=1)
plt.suptitle('Plot of the Susceptibility versus Temperature', fontsize=13)
plt.title('MF=0T and J=%d'%J, fontsize=10)
plt.xlabel("Temperature, T / K", fontsize=12);
plt.ylabel("Susceptibility", fontsize=12); plt.axis('tight');
```