Report 2, Lattice model of diffusion in 2D

Alicja Nowakowska, 234857

May 6, 2020

1 Introduction

In this report the lattice model of diffusion in 2D is considered. It's assumed that the lattice objects (atoms) are dispersed randomly at the beginning according to the atom concentration C defined as:

$$C = \frac{\text{number of atoms}}{\text{lattice size}^2}$$

and can move only up, down, right or left with equal probabilities in each simulation iteration. The number of steps that the lattice objects can take is called the Monte Carlo time and denoted as MC. After each simulation performance (when the MC time is over) the final positions are saved in the list. In order to obtain reasonable results simulation is performed K times. At the end of the procedure the diffusion coefficient can be calculated using the following formula (Big Data Analytics Statistical Physics Script prof. Grzegorz Pawlik):

$$D = \frac{< \Delta R^2 >}{2d \cdot MCS}$$

where $< \Delta R^2 >$ denotes the average of the squared displacement over all atoms and all independent realizations and d is the lattice dimension which is 2 in the considered case.

2 Exemplary plot of D against MC time

On the Figure 1 Exemplary plot of D against MC time for the atom concentration $C = 0.5$, number of independent realizations $K = 10$ and MC time from 1 up to 50 and the lattice size $L = 20$.

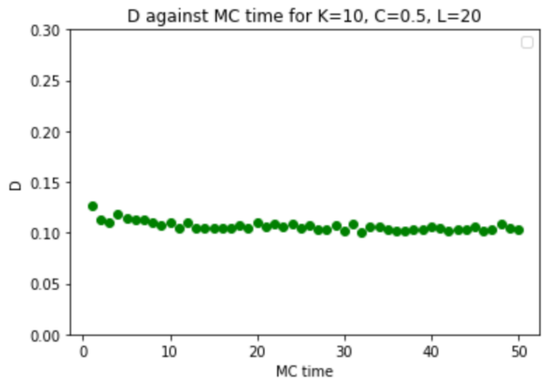

Figure 1: Exemplary plot of D against MC time

It may be noted that the diffusion coefficient is stabilizing already for less than 10 MC time units.

3 The plot of D against concentration C

The next step of the analysis of the lattice model of diffusion in 2D was the performance of the simulation for various C concentration what is shown on the Figure 2. The MC time was 50 in order to get a good accuracy of the diffusion coefficient calculation, the number of independent realizations was $K = 10$ and the lattice size $L = 20$. C varied from 0.01 to 1 with the step 0.01.

It may be noted that the relation between D and C is linear. In order to define the relation formula the regression model was adjusted for the data:

$$D = 0.2503 - 0.1138C$$

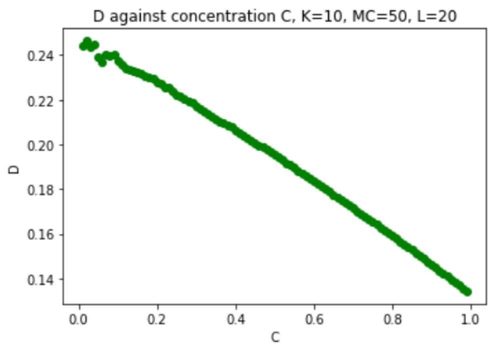

Figure 2: The plot of D against concentration C

4 Conclusion

After the analysis presented in this report it can be concluded that the estimation of the diffusion coefficient can be obtained for already 10 MC time steps and that the relation between C and D is evidently linear.

5 Code

Listing 1: Simulation code

```
import random
import matplotlib.pyplot as plt
import numpy as np
import scipy
import scipy.stats
def run_program(atom_concentration, lattice_size, MC_time):
    #STEP 1 NEIGHEREST NEIGHBOURS
    IN=[]
    IP=[lattice_size -1]
    for i in range(0,lattice_size -1):
        IN.append(i+1)
        IP.append(i)
    IN.append(0)

    #SET THE LATTICE AND X Y POSITIONS
    A=[]
    for i in range(lattice_size):
        A.append([0]*lattice_size)
    #SET THE ATOMS, atom_concentration: n_atoms/lattice_size^2
    n_atoms=int(atom_concentration*lattice_size*lattice_size)
    x=[]
    y=[]

    index=0
    while index<n_atoms: #rzmieszczenie atom w
        X=int(random.random()*lattice_size)
        Y=int(random.random()*lattice_size)
        if A[X][Y]==0:
            A[X][Y]=1
            x.append(X)
            y.append(Y)
            index+=1

    #x_initial=x.copy()
    #y_initial=y.copy()
    #run random walk on the lattice
    deltaR2=[]
    for MC in range(MC_time):
        drx=0
        dry=0
        for k in range(n_atoms):

            a=int(random.random()*4+1) #which movement direction , x[k] y[k] atoms postitions

            if a==1: #move right

                if A[IN[x[k]]][y[k]]==0: #ToIN to jest takie jakby przesuni cie
                    A[x[k]][y[k]]=0
                    A[IN[x[k]]][y[k]]=1
                    x[k]=IN[x[k]]
                    drx+=1

            elif a==2: #move left
                if A[IP[x[k]]][y[k]]==0:
                    A[x[k]][y[k]]=0
                    A[IP[x[k]]][y[k]]=1
                    x[k]=IP[x[k]]
                    drx-=1

            elif a==3: #move up
                if A[x[k]][IN[y[k]]]==0:
                    A[x[k]][y[k]]=0
                    A[x[k]][IN[y[k]]]=1
                    y[k]=IN[y[k]]
                    dry+=1

            elif a==4: #move down
                if A[x[k]][IP[y[k]]]==0:
                    A[x[k]][y[k]]=0
                    A[x[k]][IP[y[k]]]=1
                    y[k]=IP[y[k]]
                    dry-=1

        dr2=(drx**2+dry**2)/n_atoms
        deltaR2.append(dr2)

    raw_d=sum(deltaR2)/MC_time
    return raw_d

#1st plot
d = 2
K = 10
MCS = [] #monte carlo step
dR = []
D =[]
for mc in range(1,50):

    for i in range(0,K):
        drr = run_program(0.5,20,mc) #koncentracja, wielko    tablicy, ile monte carlo
        dR.append(drr)

    MCS.append(mc)
    D.append((sum(dR)/len(dR))/(2*d))

plt.plot(MCS,D,'go')
plt.title("D againt MC_time, K=10, C=0.5, L=20")
plt.xlabel("MC_time")
plt.ylabel("D")
axes = plt.gca()
axes.set_ylim([0,0.3])
plt.show()

#Second plot
d = 2
K = 10
conc = [] #monte carlo step
dR = []
D =[]
d_conc=[]

for c in range(1,100):
    mc=50
    for i in range(0,K):
        drr = run_program(c/100,20,mc) #koncentracja, wielko    tablicy, ile monte carlo
        dR.append(drr)

    conc.append(c/100)
    d_conc.append((sum(dR)/len(dR))/(2*d))

plt.plot(conc,d_conc,'go')
plt.title("D against concentration C, K=10, MC=50, L=20")
plt.xlabel("C")
plt.ylabel("D")
#axes = plt.gca()
#axes.set_ylim([0,0.3])
plt.show()

#regression
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(conc, d_conc)
print("slope",slope)
print("intercept",intercept)
```