Computational Physics (Physics.566.01.Sp15)–Group Assignment

Tong Zhu (NetID: tz37)
Mechanical Engineering & Materials Science, Duke University[*]
(Dated: April 2, 2015)

[*] E-mail me:tz37@duke.edu

I. INTRODUCTION

A. 2D random walk

In this part, we would do a problem about random walk in 2 dimensions, taking steps of unit length in ±x or ±y direction on a discrete square lattice.

II. METHOD

A. 2D random walk

B. Gas Mixing

The normal way to do this question can be done by the following ways:

- Find the position of the gas particle randomly
- move the gas particle randomly.

As the definition of our movement, if the positions around the particle has no empty states, then this particle would not be move. This is an waste iteration in the algorithm. According to this, we can add one acceleration method to this problem. That is only selecting the gas particle which its around have at least one empty states randomly.

On the other hand, let's we consider more about this question. The gas particle only move to the empty direction randomly. When this move happened, the coordinate of the gas states and the empty states exchanged. So this question is basic equal to the empty states move randomly.

If we think like this, we can solve this question by the following ways:

- find the position of the empty states randomly
- move the empty states randomly.

The same as the acceleration of gas particle movement, we can still cancel the waste iteration by just choose the empty states randomly for those which its neighbor have at least one gas states.

According to this, our code is like below (the choice function and the move function):

```
def choice5(nx,ny,origin):
    c=0
    while True:
        x=randrange(0,nx)
        y=randrange(0,ny)
        if origin[x,y] == 0 :
            if x < nx-1:
                if origin[x+1,y]!=0:
                    c=1
            if x >0:
                if origin[x-1,y]!=0:
                    c=1
            if y< ny-1:
                if origin[x,y+1]!=0:
                    c=1
            if y>0:
                if origin[x,y-1]!=0:
                    c=1
            if c==1:
                c=0
                break
    return x,y
```

```
def move2(nx,ny,i,j,origin):
    #corresponding to the move of choice 2 and choice 3,
    #after that,return the charge density
    c=random()
    if 0.0  <= c <= 0.25:
        if i<nx-1:
            if origin[i+1,j]!= 0:
                origin[i,j]=origin[i+1,j]
                origin[i+1,j]= 0
    elif 0.25 < c <= 0.5:
        if i>0:
            if origin[i-1,j] != 0:
                origin[i,j] = origin[i-1,j]
                origin[i-1,j] = 0
    elif 0.5 < c <= 0.75:
        if j<ny-1:
            if origin[i,j+1]!= 0:
                origin[i,j]=origin[i,j+1]
                origin[i,j+1] = 0
    else:
        if j>0:
            if origin[i,j-1] != 0:
                origin[i,j] = origin[i,j-1]
                origin[i,j-1] = 0
    return origin
```

FIG. 1. The choice function and the move function of the empty states.

The choice function give you the coordinates of the randomly choice empty states positions, and then move functions according to this coordinates to change the matrix of all gas states and empty states.

As in the choice and the move, we would encounter the problem about the boundary (the movement of the corner or the axis). In this question, we still permit the code to randomly choose that direction to move, but we don't change the matrix at all. You can see that is another waste iteration. By this, we do not need to specify the corner atoms or axis atoms specifically. But just change the matrix if the gas/empty states can move.

The logic is like below:

- if we randomly choose the +y direction to move, we only change the gas/empty states which its y corrdinate is lower than the boundry(ny-1 in the code).
  - The reason for that is if we randomly choosing y coordinate is in the +y boundary, and the randomly choice direction to move is +y, then this move cannot happen, we don't change anything.
- if we randomly choose the -y direction to move, we only change the gas/empty states which its y corrdinate is higher than the boundry(0 in the code).
- if we randomly choose the +x direction to move, we only change the gas/empty states which its x corrdinate is lower than the boundry(0 in the code).
- if we randomly choose the -x direction to move, we only change the gas/empty states which its x corrdinate is higher than the boundry(0 in the code).

The whole solution code is like below:

```
if method==3:
    #random choose empty whose neighbor have gas, then move it
    for i in range (0,niter):
        x,y = choice5(nx,ny,origin_old)
        origin_new = move2(nx,ny,x,y,origin_old)
        origin_old = np.copy(origin_new)
return origin_new
```

FIG. 2. solution iteration of the code.

III. RESULTS

A. 2D random walk

In this part, we plot $< x_n >$ and $< (x_n)^2 >$ up to n=100 by averaging over at least 10^4 different walks. The result are shown in Fig. 3.

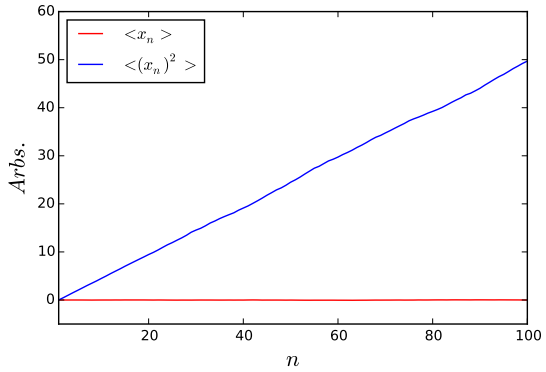

FIG. 3. $< x_n >$ and $< (x_n)^2 >$ up to n=100 by averaging over at least 10^4 different walks.

You can see that the $< x_n >$ is always zero which is agree with our expectation. For the random walk is walking randomly along +x direction and -x direction, so the average of that would be zero after many times of random walks. As you can see that correlation between $< (x_n)^2 >$ and n is linearly, and the ratio is like 0.5. The reason for that is that the random walks in two axis, neither ±x direction or ±y direction, so the probability of walking in ±x direction is almost 0.5 which is also agree with our results.

By see the mean square distance from the starting point $< r^2 >$ up to 100 times by averaging over at least 10^4 different walks, shown in Fig. 4.

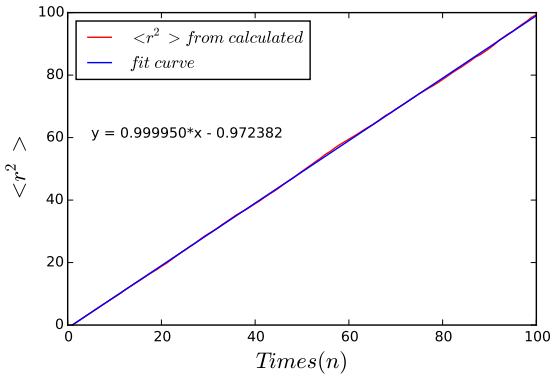

FIG. 4. $< r^2 >$ up to n=100 by averaging over at least 10^4 different walks.

we can see that the motion is diffusive and the value of the diffusion constant after an curve fit is like 0.99995 which is very close to 1. The reason for that the mean square distance from the starting point stands for the walk of two aixs (x and y), so the probability of that is equal to 1 which is agree with our plot.

B. Gas Mixing-problem c)

For the Mixing-problem, when we use the acceleration method, we can get the mixing states more quickly, we use 60×40 parts as a test case, For this case, we only need 10^6 iterations (the normal method needs 10^7 iterations to get this states) to get the whole mixing states like below:

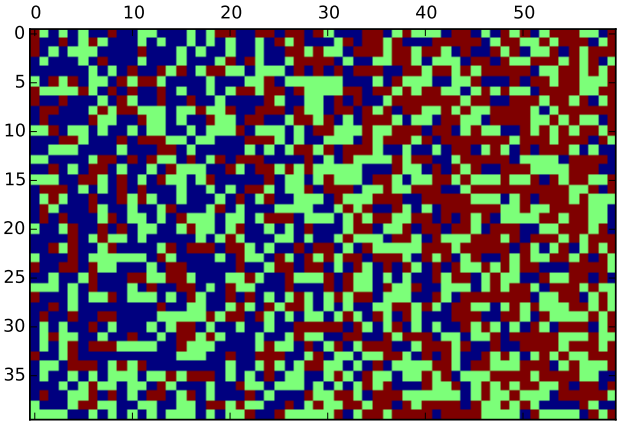

FIG. 5. The gas mixing of 60×40 dimension grids, blue stands for gas A, red stands for gas B, and green stands for the empty.

We difine the density like below, for each x grid point (x coordinate), accumulate all the y coordinate, get the number of gas A or gas B numbers. Then use this number divided by the whole gas A or gas B numbers. The density is show like below:

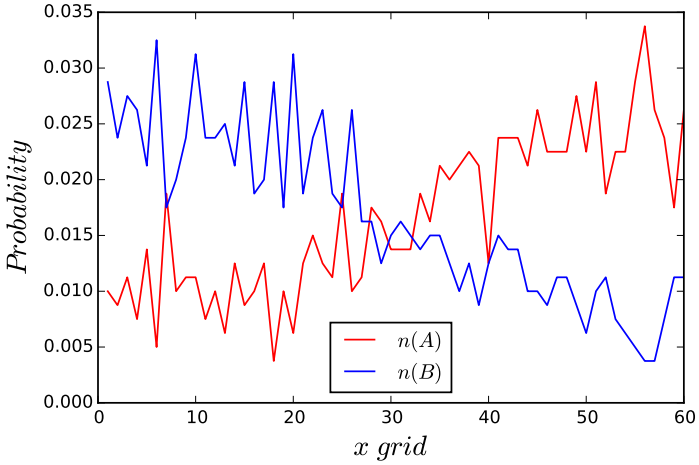

FIG. 6. The density of 60×40 dimension grids when it close to the whole mixing states.

After as the Problem c did, we average of 100 different trials of the whole mixing states, then the average density is like below:

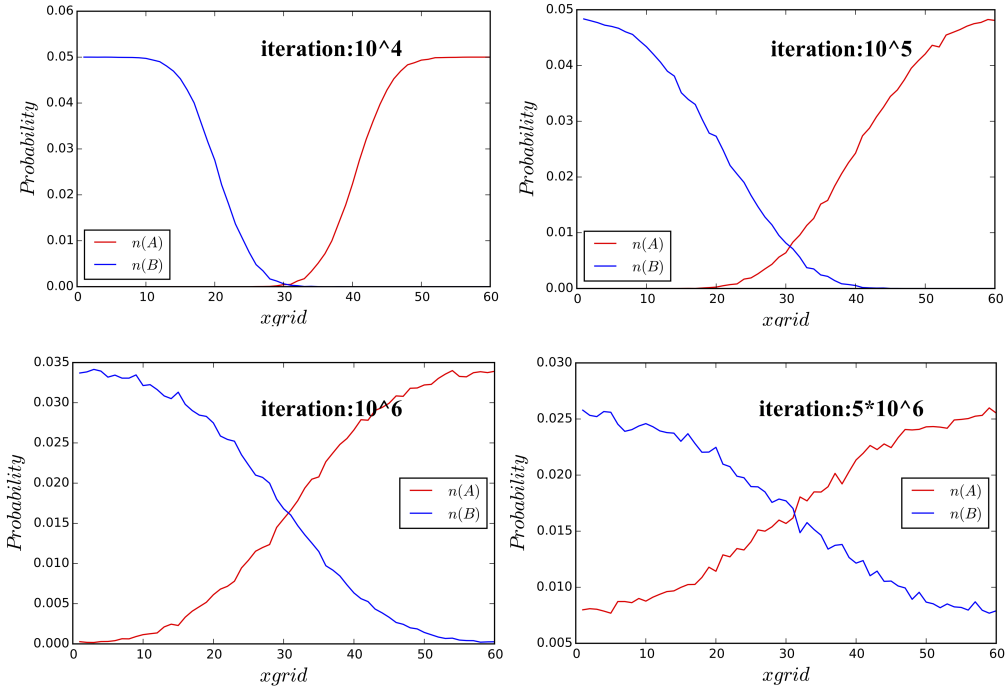

FIG. 7. Average density of 100 trials with different iteration numbers (times)

From that, we can see that, as the time goes longer (iteration number higher), the states would lead to an mixing states, and the states would be more randomly. You can see that even average of 100 trials, it still have some wiggle, not smoothly like the 10^4 iteration (very smooth).

IV. CONCLUSION