Simulate mixing of gas molecules

Cheng Su

19 November 2019

Summary

Problem Description

Purpose of this project is to simulate the effect of collision of molecules in 3D using random walk in 2D. In particular, the random walk can be used to investigate how a quite ordered system, where one gas fills one half of a box, evolves through time to a more disordered system.

This physical process can be simulated by random walk inside a fixed area A = [0, 1] * [0, 1]. Initially, 10000 particles are placed at uniformly distributed random positions in area of [0, 1/2][0, 1]. Then start the random walk and visualize the simulation during a long time and save as an animation.

Bench-marking

Particle class is created to initialized the particle appearing at random position within area [0, 1/2][0, 1]. It also allows particle to move in a random position(up, down, left, right) with the given step distance of 0.05. Total duration of the simulation is 200, and visualize animation is done using matplotlib.FuncAnimation.

Code

Import libraries and initialize constants. Time is the number of steps in the simulation, N is the total number of particles, step is the distance for every step of movement.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib.animation as animation
4
5 # constants
6 time = 200
7 N = 10000
8 step = 0.05
```

Code

Create a particle class. Particles spawns at random place. It can choose to take a move of 0.05 towards random direction(up, down, left or right)

```
1  class Particle:
2      def __init__(self):
3          self.position = [np.random.uniform(0, 0.5), np.random.uniform(0, 1)]
4
5      def move(self):
6          x = self.position[0]
7          y = self.position[1]
8          move_choice = np.random.randint(4)
9          if move_choice == 0:
10             self.position = [x, y + step]
11         elif move_choice == 1:
12             self.position = [x, y - step]
13         elif move_choice == 2:
14             self.position = [x + step, y]
15         elif move_choice == 3:
16             self.position = [x - step, y]
17
18         if self.position[0] > 1 or self.position[0] < 0 or self.position[1] > 1
       or self.position[1] < 0:
19             self.position = [x, y]
20             self.move()
```

Code

Start the simulation by first initialize an array of N particles. Plot the plot with grid, and scatter the points on the plot.

```
1  class Particles:
2      def __init__(self):
3          self.particles = []
4          for i in range(N):
5              self.particles.append(Particle())
6
7      def simulate(self):
8          # set up plot
9          fig = plt.figure()
10         ax = fig.add_subplot(111)
11         ax.grid(True, linestyle='-', color='0.75')
12         ax.set_xlim([0, 1])
13         ax.set_ylim([0, 1])
14         x = []
15         y = []
16         for i in range(N):
17             x.append(self.particles[i].position[0])
18             y.append(self.particles[i].position[1])
19         scat = plt.scatter(x, y, c=x, s=0.3)
```

Code

Move the particles and generate the animation.

```
1
2           def _update_plot(i, fig, scat):
3               pos = []
4               for index in range(N):
5                   self.particles[index].move()
6                   pos.append(self.particles[index].position)
7
8               scat.set_offsets(np.array(pos))
9               return scat,
10
11          anim = animation.FuncAnimation(fig, _update_plot, fargs=(fig, scat),
        frames=time, interval=60)
12          #plt.show()
13          anim.save('disorder.gif', writer='imagemagick')
```

Result

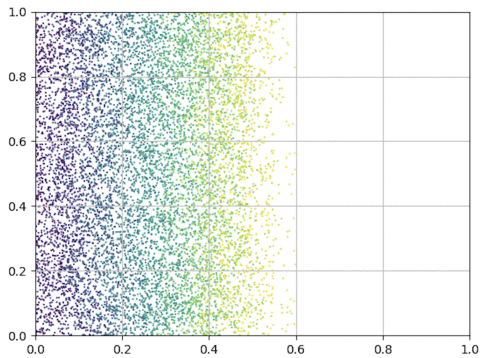

FIGURE – particle positions at time 0

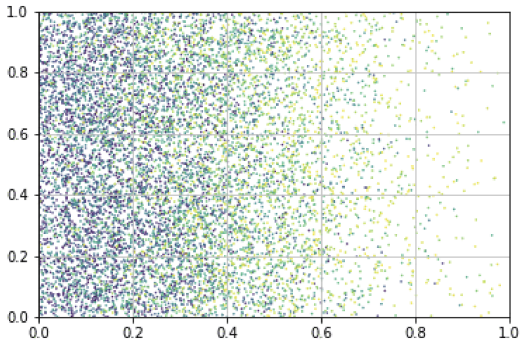

FIGURE – particle positions at time 35

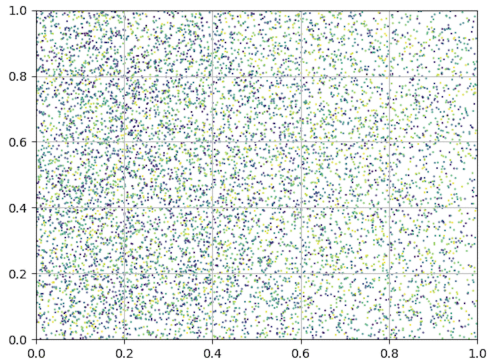

FIGURE – particle positions at time 200

Discussion

Above are the result at the beginning and the end. We can infer that eventually the particles will be uniformly distributed in the whole space.