Monte-Carlo Diffusion Simulation

Author: Partner:

Abstract
Diffusion is driven by the random motions of countless tiny particles and can be found easily on daily life. Python diffusion modeling provides a visual diffusion simulation and offers insights into the underlying random processes. This project presents a Monte-Carlo approach to simulate diffusion by modeling many individual particles' random walks in Python.

Introduction

Diffusion is the process by which particles spread out randomly from areas of high concentration to areas of low concentration[1]. While the motion of individual particles may appear random, the cumulative effect follows rules allowing prediction for macroscopic properties of the system. This phenomenon occurs in countless real-world systems across science and engineering.

In this project, we modeled diffusion using Monte Carlo methods in Python. By simulating the random walks of many particles in 3D over time, we were able to reproduce expected diffusion behavior and gain insight into the underlying random process. Different scenarios were investigated, including free space diffusion and diffusion within confined spaces. Artificial intelligence technique such as the HDBSCAN clustering method is also applied, which can be used for more advanced level predictions.

Diffusion in Physics Theory

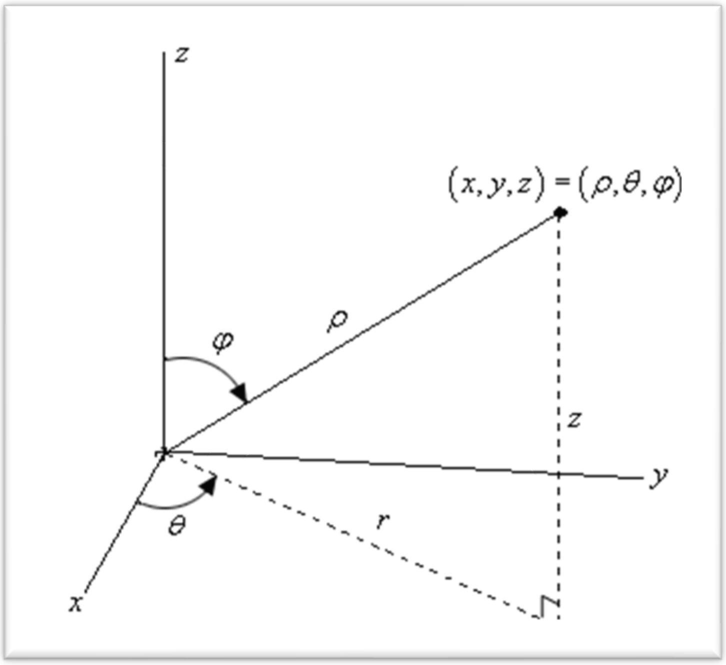
Diffusion in theoretical physics is described as Fick's first law, which postulates the diffusive flux goes from regions of high concentration to regions of low concentration.[2]

$$J = -D\frac{\partial \varphi}{\partial x} \qquad \text{---[2]}$$

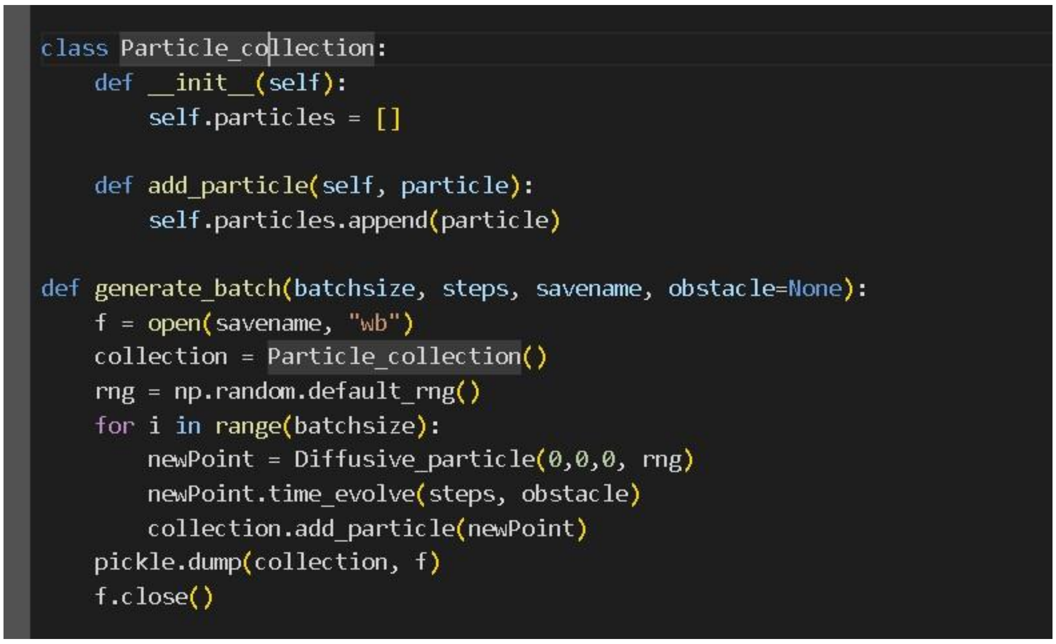
D is diffusion coefficient, which indicates how rapid is the diffusion process with respect to time
φ is the concentration

Simulation setup

The critical component of the simulation is the Diffusive_Particle class, which tracks the position history of each simulated particle as it undergoes a random walk in 3D space. At each time step, every particle randomly chooses a direction defined by azimuthal and polar angles and moves a fixed distance in that 3D direction.



Collections of particles are generated by creating many objects of the Diffusive_Particle class with initial positions at the origin region. Particles then diffuse freely, or their motion can be confined by virtual obstacles like walls or boxes that reflect their original movement upon collision. By simulating many timesteps, particle positions are analyzed to study diffusion behavior.

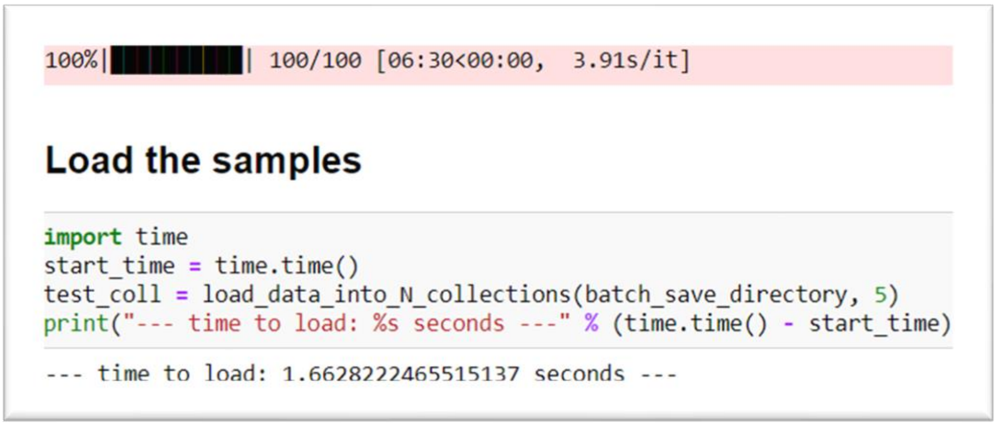

```
class Particle_collection:
    def __init__(self):
        self.particles = []

    def add_particle(self, particle):
        self.particles.append(particle)

def generate_batch(batchsize, steps, savename, obstacle=None):
    f = open(savename, "wb")
    collection = Particle_collection()
    rng = np.random.default_rng()
    for i in range(batchsize):
        newPoint = Diffusive_particle(0,0,0, rng)
        newPoint.time_evolve(steps, obstacle)
        collection.add_particle(newPoint)
    pickle.dump(collection, f)
    f.close()
```

Simulation performance Optimization

To optimize performance, particles are pre-simulated once and saved to folder rather than regenerated each time on runtime. Whenever necessary, particles are loaded into variables for analysis; this method improves runtime by about 230 times than regenerating simulation.



```
100%|██████████| 100/100 [06:30<00:00,  3.91s/it]
```

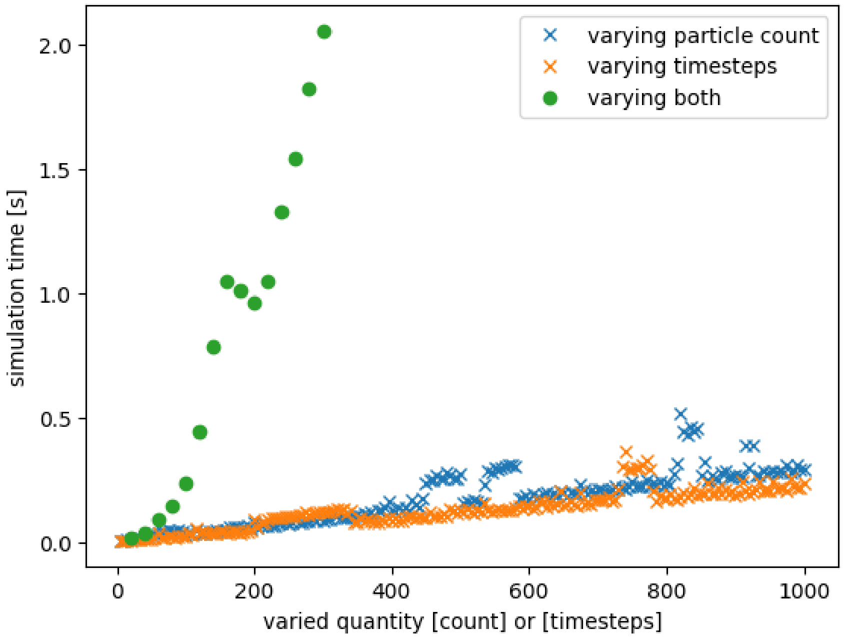
Load the samples

```
import time
start_time = time.time()
test_coll = load_data_into_N_collections(batch_save_directory, 5)
print("--- time to load: %s seconds ---" % (time.time() - start_time))
```

```
--- time to load: 1.6628222465515137 seconds ---
```

(The red bar indicates time consumed for generating, i.e., 6 minutes 30 seconds.)

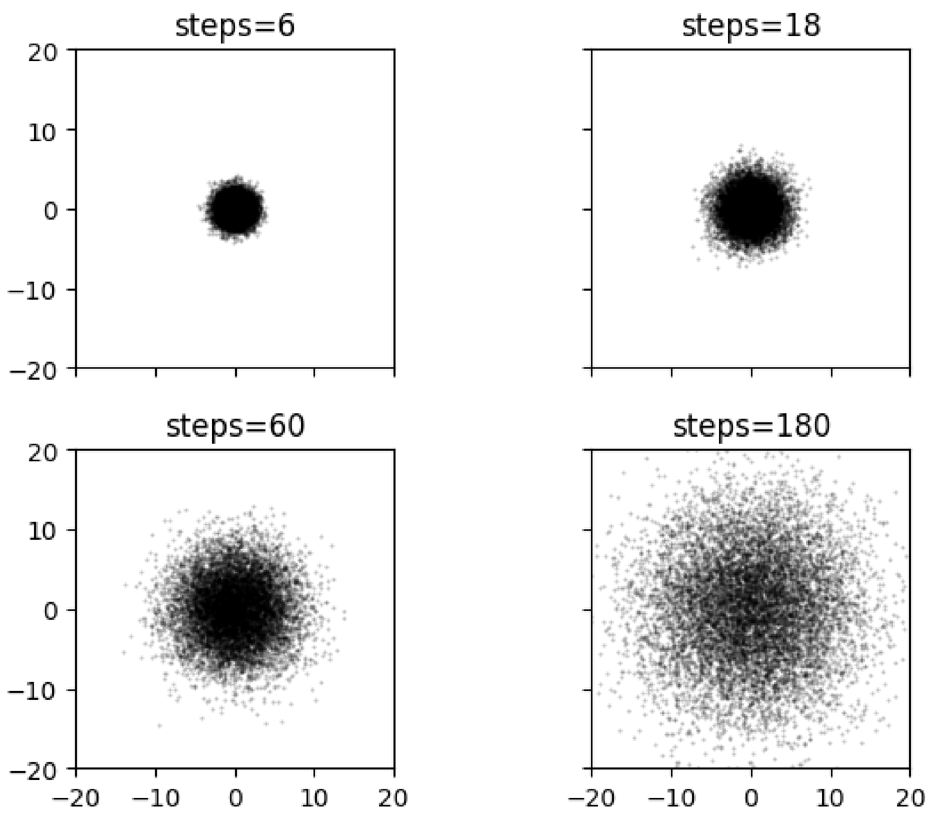
To analyze the time complexity for generating new particles simulation, it is found that the time complexity scales as $O(n^2)$ for n particles over n timesteps.



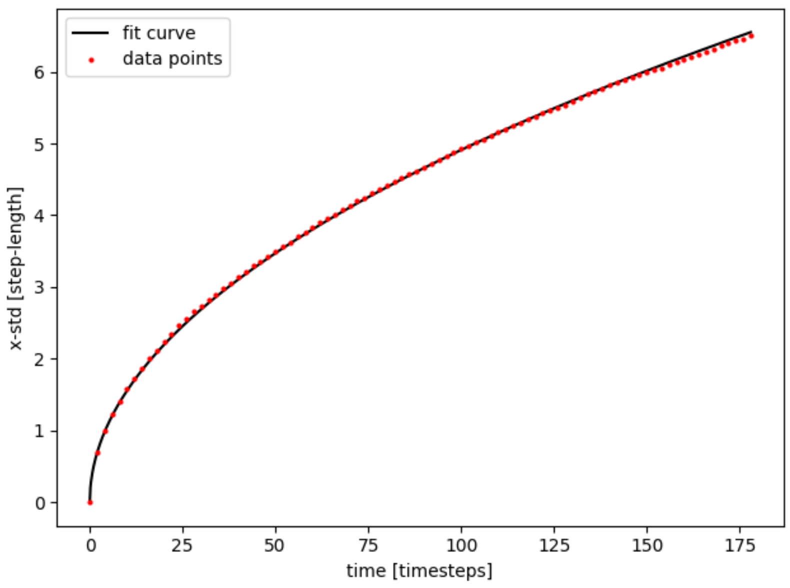
Therefore, the generate-load strategy is applied to the simulation.

Free Diffusion

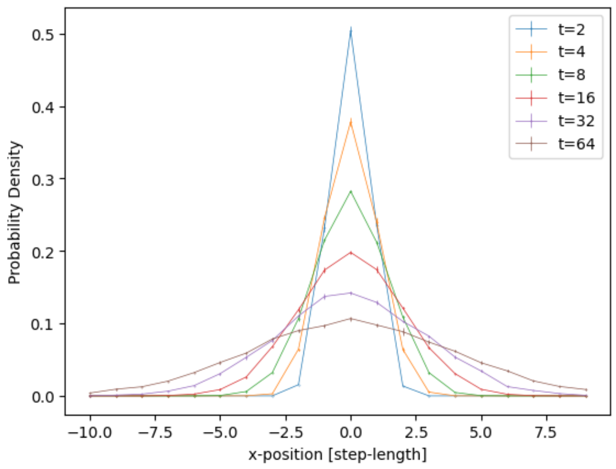
On a free diffusion simulation, particles were initialized at the origin and then allowed to undergo 3D random walks by choosing random 3D space directions and moving a fixed step length of 1 at each timestep. Since showing them in 2D graphs cause the separation distance between particles to decrease, graphs with about 3x timesteps are therefore shown.

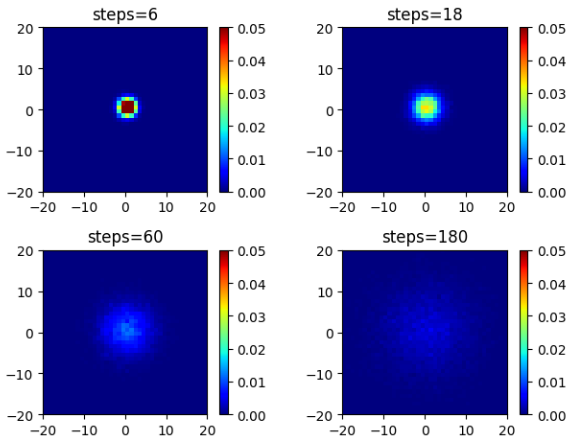


As particles spread out to larger areas, the standard deviation of particle distribution keeps increasing as the square root of time.

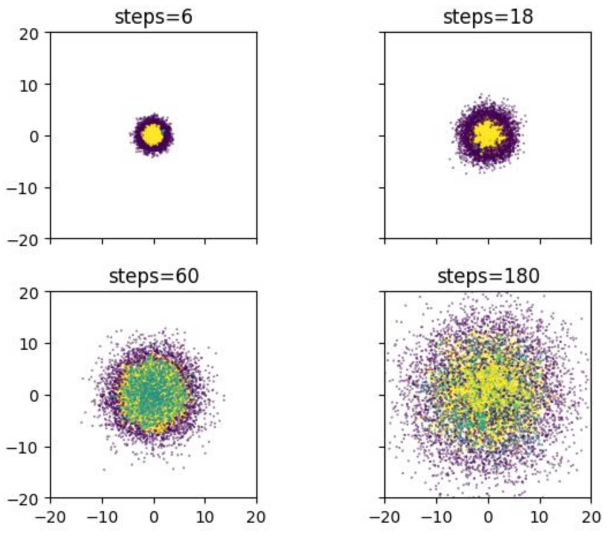


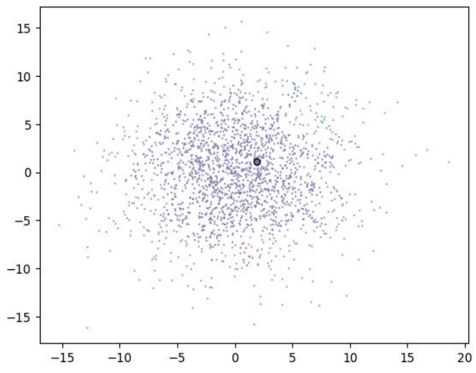
The probability density distribution should flatten as the particles diffused over a wider space as time goes by. To analyze the results, we generated histograms of the particle positions, both 1D histograms of the x-position distribution and 2D histograms of the full x-y position distribution. Errors were estimated by breaking the total sample into batches and calculating the standard error on the mean bin counts.



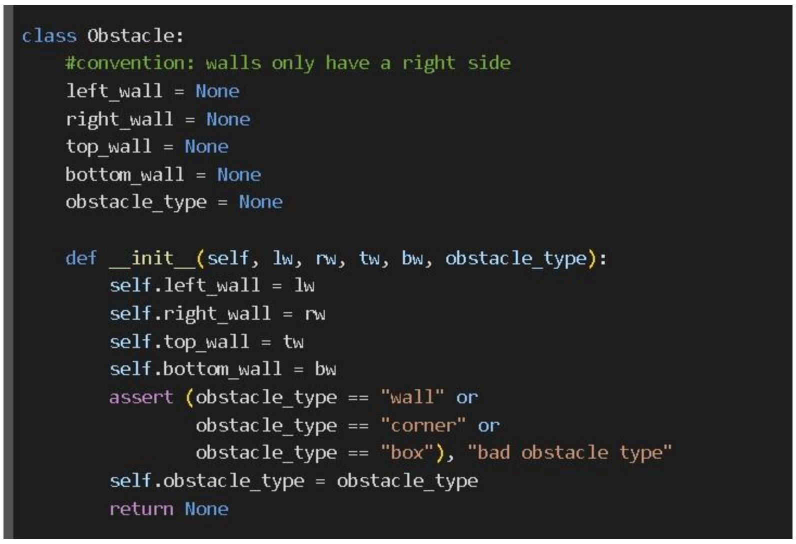


By using one clustering tool of machine learning – HDBSCAN, we can consider particles as different clusters and carry out future analysis[3][4]. For example, in a water molecule diffusion study for evaporation, the outer-most layer has the highest probability of escaping space and becoming the gas phase. We can also cluster a particular particle at a certain timestep to predict its future movement.





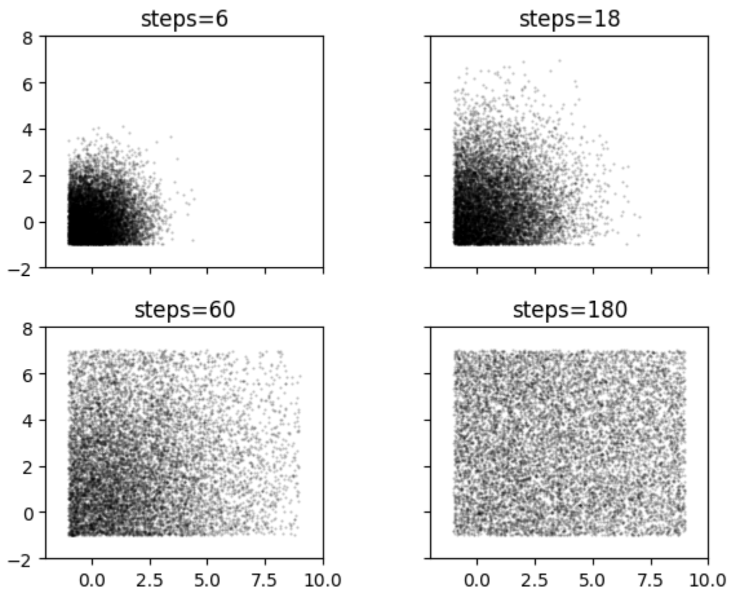
Confined Diffusion
We add the Obstacle class to the original model to model diffusion within a bounded space. Moreover, we also need to add functions like checking legal moves and reflecting the particle to the original model so that particles can move in the way we want.



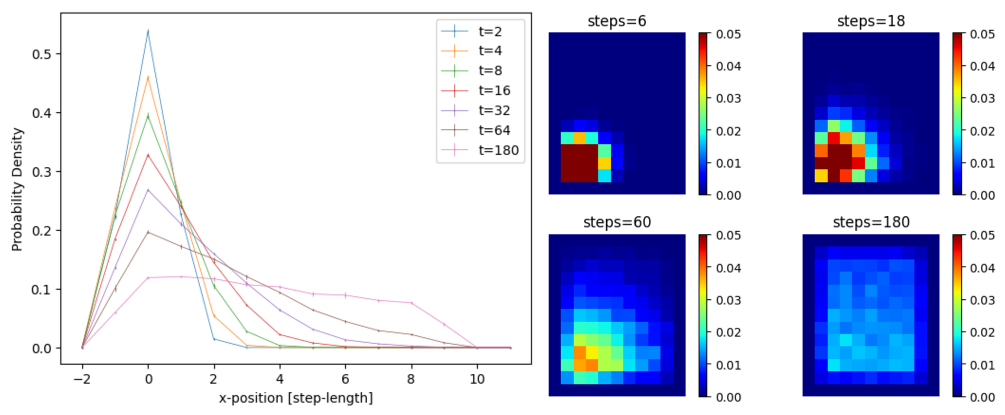
```python
class Obstacle:
    #convention: walls only have a right side
    left_wall = None
    right_wall = None
    top_wall = None
    bottom_wall = None
    obstacle_type = None

    def __init__(self, lw, rw, tw, bw, obstacle_type):
        self.left_wall = lw
        self.right_wall = rw
        self.top_wall = tw
        self.bottom_wall = bw
        assert (obstacle_type == "wall" or
                obstacle_type == "corner" or
                obstacle_type == "box"), "bad obstacle type"
        self.obstacle_type = obstacle_type
        return None
```

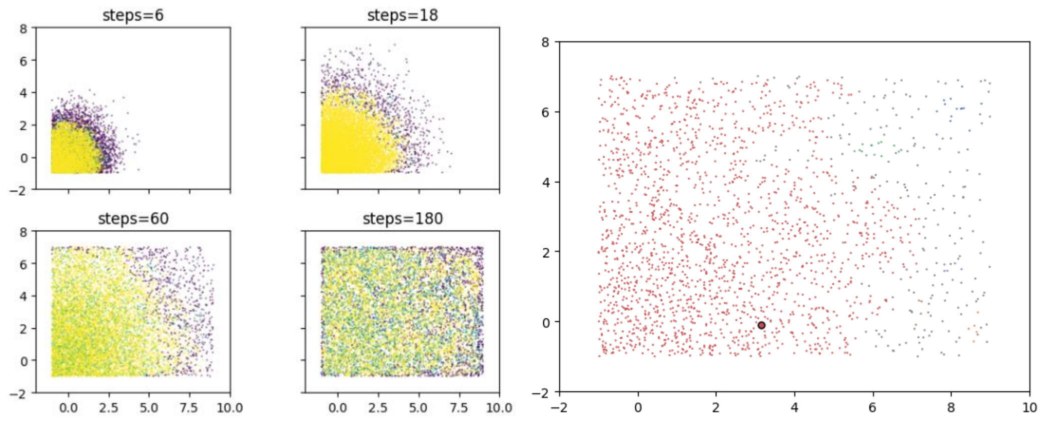
We apply box boundaries and elastic particle collisions with the walls. The box is located slightly to the upper-right position. Elastic collision indicates that upon collision, particles reflect and continue their movement for that timestep.



Initially, particles spread, and some reach the boundaries. After bouncing off the walls, their distribution begins to flatten as they fill up the entire box area. The probability distribution shifts from an initially Gaussian distribution to a flat, uniform distribution.



Machine learning HDBSCAN technique can also be applied, and particle clustering prediction can also be carried out.[3][4]



Success/Limitation

We succeeded in building a simulation starting from scratch, then we keep adding more and more functions and features, and eventually finished the project. We used OOP, libraries, nested functions, and simple machine learning to simulate and analyze a physics process. However, there are also some limitations. We did not consider the factors like a collision between particles, energy loss due to collisions, external forces like gravity, etc. More advanced features and functions can be carried out to apply the simulation model to more situations.

Conclusion

We successfully applied Monte Carlo methods to simulate and analyze free and confined diffusion systems. Our Python code uses OOP to model numerous random particles. Their movements demonstrate the expected diffusive behavior. We also apply code optimization method, that allows us to handle large datasets in a short time. Moreover, analysis and data visualization allow us to validate physics diffusion theory. This project provides a foundation for future investigations into more complex diffusion models and phenomena.

Acknowledgements
I would like to acknowledge my project partner, , for his valuable contributions to this project. He spent so much energy building and debugging different parts of the model, and he explained to me some advanced diffusion theory to help me understand better. I am very thankful for 's help and partnership. In addition, the advice given by Professor and GSI provides us with deeper insight into the simulation, their suggestions helped us save the computing time, and let us know more about the factors determining computer efficiency is the particle number and timesteps for this model.

Reference

[1] “Russell Kightley scientific animations diffusion (rights managed stock footage),” Russell Kightley Scientific Illustrator &amp; Animator, https://www.scientific.pictures/-/galleries/physics/-/medias/fe446b0b-45b4-436e-bc05-c97eecca5558-diffusion (accessed Aug. 11, 2023).
[2] “Fick’s laws of diffusion,” Wikipedia, https://en.wikipedia.org/wiki/Fick%27s_laws_of_diffusion (accessed Aug. 11, 2023).
[3] “Sklearn.cluster.HDBSCAN,” scikit, https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html#sklearn.cluster.HDBSCAN (accessed Aug. 11, 2023).
[4] “Predicting clusters for new points,” Predicting clusters for new points - hdbscan 0.8.1 documentation, https://hdbscan.readthedocs.io/en/latest/prediction_tutorial.html (accessed Aug. 11, 2023).