🧪 **PYTHON PROJECT REPORT**

Simulation of Brownian Motion Using NumPy and Matplotlib

---

1. Project Title

Simulation of Brownian Motion and Temperature Effect on Gas Molecules Using Python

---

2. Introduction

Physics concepts can be better understood when visualized through simulations. In this project, a Python-based simulation was developed to model the random motion of gas molecules inside a container.

The project demonstrates Brownian motion and the relationship between temperature and kinetic energy using programming tools such as NumPy and Matplotlib.

This simulation connects computer programming with concepts from kinetic theory of gases taught in Class 11 Physics.

---

3. Objectives

- To create a 2D gas particle simulation using Python.
- To visualize random motion of particles.
- To show how temperature affects molecular speed.
- To apply NumPy for numerical computation.
- To use Matplotlib for animation and visualization.

---

4. Technologies Used

1. Python (Programming Language)
2. NumPy – for numerical calculations and arrays
3. Matplotlib – for graphical visualization and animation
4. Seaborn – for temperature-based color mapping

---

5. Scientific Background

Brownian motion was first observed by Robert Brown in 1827. It describes the random movement of particles suspended in a fluid due to collisions with molecules.

According to the kinetic theory of gases:

Average Kinetic Energy ∝ Temperature

$KE = 1/2\ mv^2$

When temperature increases, molecular speed increases. This concept is implemented in the program by increasing velocity magnitude.

---

6. Program Design

Step 1: Creating the Container

A 2D square box was created to represent a gas container.

Step 2: Initializing Particles

- 200 particles were randomly positioned inside the box.
- Random velocities were assigned using NumPy.
- Temperature variable controls velocity magnitude.

Step 3: Updating Motion

At each frame:

- Position = Position + Velocity × time
- Wall collisions reverse velocity direction.

Step 4: Speed Calculation

Speed of each particle was calculated using:

$speed = \sqrt{(vx^2 + vy^2)}$

Step 5: Temperature Visualization

- Speeds were normalized.
- Seaborn color map was applied.
- Blue = Low energy
- Red = High energy

Step 6: Animation

Matplotlib's animation function updates particle positions continuously to simulate motion.

---

7. Algorithm

1. Import required libraries.
2. Define number of particles.
3. Generate random positions.
4. Generate random velocities.
5. Create animation loop.
6. Update positions using velocity.
7. Detect wall collisions.
8. Calculate speed.
9. Assign colors based on speed.
10. Display updated frame.

---

8. Output Description

The output shows:

- A square container with moving particles.
- Random motion of particles.
- Faster motion when temperature variable increases.
- Color variation indicating kinetic energy.

The simulation clearly demonstrates how temperature affects molecular movement.

---

9. Sample Code Structure

Main components of the program include:

- Initialization section
- Update function
- Collision handling
- Speed calculation
- Animation loop

The code is structured for clarity and easy modification of parameters like temperature and particle count.

---

10. Learning Outcomes

Through this project, the following were learned:

- Application of physics concepts in programming.
- Use of NumPy arrays for efficient computation.
- Creating real-time animations in Python.
- Understanding relationship between temperature and molecular motion.
- Visualization of scientific data.

---

11. Advantages of Simulation

- Safe and cost-effective experiment.
- Easy to modify temperature conditions.
- Provides better visual understanding.
- No physical laboratory required.

---

12. Limitations

- The simulation is 2D instead of 3D.
- Real molecular collisions are more complex.
- Assumes ideal gas conditions.

---

13. Conclusion

The Python simulation successfully models Brownian motion and the effect of temperature on molecular speed.

The project combines physics and programming to create an interactive scientific model. It demonstrates how computational tools can enhance conceptual understanding in science education.

This project proves that programming can be effectively used to simulate real-world physical phenomena.