PHY 607 Project 2: Modeling the Motion of Gas Particles with Monte Carlo Simulations

Luis Rufino and Julia Rice

October 22, 2024

1 Introduction

1.1 The Problem - An Non-Ideal Gas in a Finite Volume

In our simulation, we are computing final temperature, pressure, average velocity, and Energy of N identical particles. This system is in a box of volume V, and particles interact via elastic collisions with one another. Given these collisions, we can calculate values of our system using Van Der Waal's equation for a non-ideal gas

$$(P + \frac{aN^2}{V^2})(V - Nb) = RT$$

where a accounts for the force of attraction between two particles and b accounts for the volume of the particles. However, we are neglecting attractive and repulsive forces between our particles, so that term is not needed. Additionally, provided our box volume is sufficiently large and our number of particles is relatively small, we can smimplify Van Der Waal's equation of state to that of an ideal gas:

$$PV = NRT$$

Therefore, we opted to implement ideal gas relationships when calculating our observables, which can then be compared to results of the non-ideal gas equation for analysis and validation purposes.

1.2 Algorithm Overview

The simulation consists of two main classes: **Particle** and **Box**. The **Particle** class models each individual particle in the system, including attributes for mass, position, and velocity. Each particle is initialized with a random velocity, either from a uniform distribution or sampled from the Maxwell-Boltzmann distribution, depending on the user's choice.

The **Box** class manages the simulation volume and the particles within it. It is responsible for initializing the particles, simulating their motion over time, and calculating observables such as temperature, pressure, and total kinetic energy. The key methods are:

- **initialize_particles**: Initializes N particles with initial velocities and positions.
- **step**: Updates the position of each particle for a given time step, handles collisions between particles and the box walls, and measures properties like pressure, energy, and temperature.
- **calculate_temperature, calculate_kinetic_energy, calculate_pressure**: Calculate the temperature, total kinetic energy, and pressure of the system based on particle velocities.

The algorithm evolves in discrete time steps, where the positions and velocities of particles are updated. Collisions are handled elastically, ensuring conservation of momentum and energy. Observables are measured at each time step and stored for analysis.

1.3 Computational Complexity

The computational complexity of the simulation is dominated by the following factors:

- **Particle Initialization**: Initializing N particles with velocities sampled from the Maxwell-Boltzmann distribution using rejection sampling has a time complexity of $O(N)$, as each particle requires a sample and velocity assignment.
- **Time Step Updates**: For each time step, the simulation must update the positions and velocities of N particles. Handling wall collisions requires checking all particles, which is $O(N)$. Particle-particle collisions involve checking pairs of particles, which leads to an $O(N^2)$ complexity.
- **Observable Calculations**: The pressure, energy, and temperature are calculated by summing over all particles, which takes $O(N)$ time.

In total, the time complexity for each time step is $O(N^2)$ due to the pairwise collision checks. This can be further optimized by using spatial partitioning techniques.

1.4 Limiting Parts of the Code

We can assess the most time-consuming parts of the code with a python profile (example provided in package). Our algorithm produces an animation of the simulation which acts as the main bottleneck. However, looking past this, we find that our program spends the most time in the "Step" function of our box module. As stated in the above section, this particular function is responsible for calling other functions to update values at each step and storing them in appropriate arrays. Substantial computational time is added when handling collisions, and specifically collisions between particles. Here, we check the bounds of each individual particle at every step, then calculates the resulting 3D velocity vector for particles that meet the conditions for collision.

2 Validation and Error Assessment

2.1 Expectations and Results

To validate our algorithm's performance, we calculated the expected values of certain observables using the information discussed in the Introduction section, then compared them with the results of our simulation. We selected three levels of detail by varying the number of particles in our system within a reasonable range: $N = 50$, $N = 100$, and $N = 200$. We simulated the system for 200 steps using different velocity initialization methods: uniform distribution and rejection sampling from the Maxwell-Boltzmann distribution.

The following equations were used to compute the expected values, and averages were taken over the total number of steps:

$$E_{\text{avg/particle}} = \frac{1}{2} m v_{\text{avg}}^2$$

$$E_{\text{avgTOT}} = E_{\text{avg/particle}} \times N$$

$$P = \frac{2}{3}\frac{N}{V} \times E_{\text{avg/particle}}$$

$$T = \frac{2}{3}\frac{E_{\text{avg/particle}}}{k_B}$$

Where: - m is the particle mass, - v_{avg} is the average velocity over all particles and time steps, - N is the total number of particles, - V is the volume, - k_B is the Boltzmann constant.

2.2 Velocity Distribution Analysis: Uniform vs. Rejection Sampling

In this section, we describe the methodology used to analyze the particle velocity distributions under different sampling methods (uniform and rejection sampling), fit these distributions using the Maxwell-Boltzmann distribution, and compare the calculated temperature to the measured temperature from the simulation.

Methodology 1. **Velocity Calculation**: For each particle at each time step, the speed was calculated using the velocity components in the x, y, and z directions. The speed v_i for each particle is given by:

$$v_i = \sqrt{v_{x,i}^2 + v_{y,i}^2 + v_{z,i}^2}$$

This calculation was performed using the Python library 'numpy'.

2. **Maxwell-Boltzmann Fit**: To validate the velocity distribution, we fitted the calculated speeds to the Maxwell-Boltzmann distribution using the 'scipy.stats.maxwell.fit()' function. The Maxwell-Boltzmann probability density function (PDF) in three dimensions is given by:

$$P(v) = \sqrt{\left(\frac{2}{\pi}\right)} \frac{v^2}{\sigma^3} \exp\left(-\frac{v^2}{2\sigma^2}\right)$$

where: - v is the speed of a particle, - σ is the scale parameter related to the temperature of the system.

The scale parameter σ is related to the temperature by the following formula:

$$T = \frac{m\sigma^2}{k_B}$$

where m is the particle mass and k_B is the Boltzmann constant. We then compared the temperature calculated from the fitted σ to the temperature measured directly from the simulation.

3. **Python Libraries Used**: The following Python libraries were utilized for the analysis:

- 'numpy' for vectorized calculations and handling numerical data. - 'pandas' for reading and processing the simulation data files. - 'scipy.stats.maxwell' for fitting the particle speed data to the Maxwell-Boltzmann distribution. - 'matplotlib' for generating plots of the velocity distributions.

4. **Error Propagation**: The 'scipy.stats.maxwell.fit()' function provides an estimate of the scale parameter σ and its standard error. This uncertainty in σ is propagated to the calculated temperature T using the following formula:

$$\Delta T = 2\frac{m\sigma}{k_B}\Delta\sigma$$

where $\Delta\sigma$ is the standard error of the fitted scale parameter σ.

Results The calculated temperatures from the fitted Maxwell-Boltzmann distribution for both sampling methods are shown in Table 1, alongside the measured temperatures from the simulation. The results demonstrate that both methods produce consistent temperature values, with the rejection sampling method showing a slightly better agreement with the theoretical distribution.

Sampling Method	**Fitted Temperature (K)**	**Measured Temperature (K)**	**Error (%)**
Uniform [-10,10]	$32.1948 \pm \Delta 0.1918$	32.1951	0.59
Rejection Sampling	$87.2584 \pm \Delta 0.4822$	87.2581	0.55

Table 1: Comparison of fitted and measured temperatures for uniform and rejection sampling.

2.3 Plotting the Velocity Distribution Fits

We generated velocity distribution plots for both sampling methods to visually assess how well the fitted Maxwell-Boltzmann distribution aligns with the simulated velocity data. These plots are shown in Figures 1 and 2.

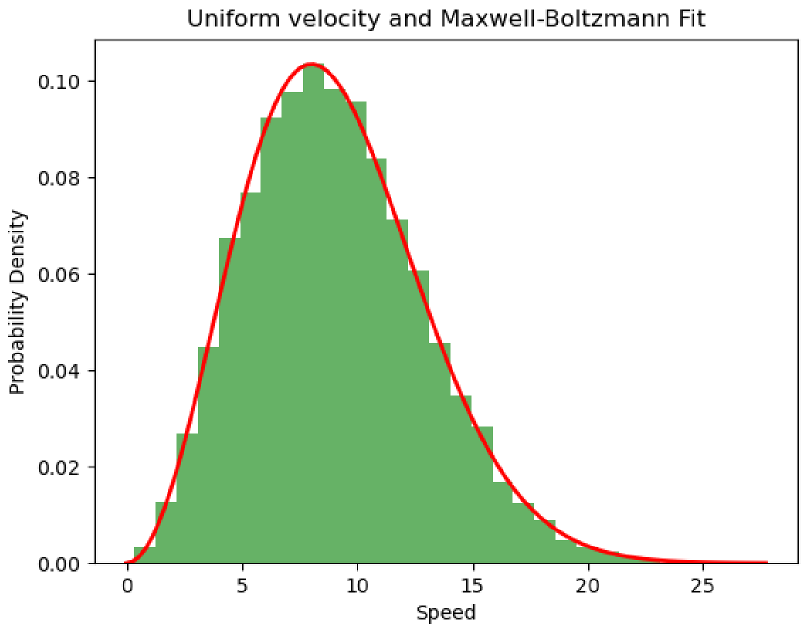


Figure 1: Fitted Maxwell-Boltzmann distribution for uniform sampling in the range [-10,10].

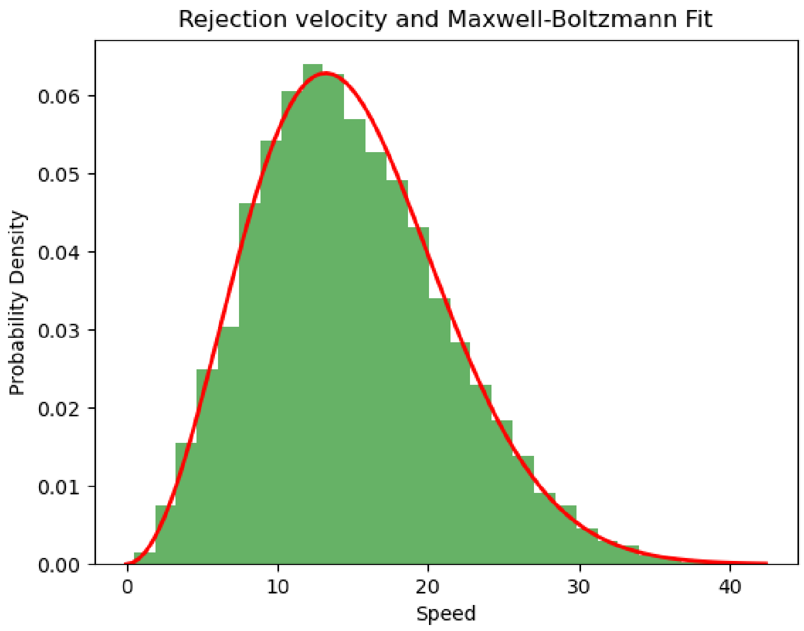


Figure 2: Fitted Maxwell-Boltzmann distribution for rejection sampling from the Maxwell-Boltzmann distribution.

2.4 Error Sources

Fitting Uncertainty The primary source of error in the temperature calculation arises from the uncertainty in the fitted scale parameter σ. The fitting process provides an estimate of this uncertainty through the standard error. This uncertainty is propagated to the calculated temperature T using error propagation, as shown:

$$\Delta T = 2\frac{m\sigma}{k_B}\Delta\sigma$$

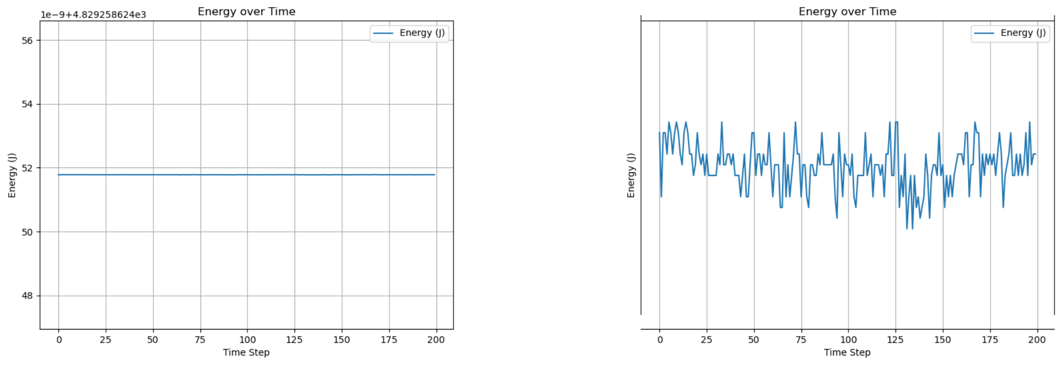
By calculating $\Delta\sigma$ from the fit, we can quantify the uncertainty in the temperature resulting from the fitting process.

Statistical Sampling The large dataset, comprising velocities from all particles over all time steps, minimizes statistical errors. Using a single random seed ensures consistency and reproducibility in the simulation results. Differences observed between the uniform and rejection sampling methods highlight the impact of initial velocity distributions on the system's behavior.

Systematic Errors In our simulation, we carefully controlled for several potential sources of systematic error: time-stepping, boundary conditions (including wall and particle collisions), and numerical precision. These elements are essential for accurately modeling particle dynamics and ensuring that the velocity distribution and temperature calculations are reliable.

1. **Time-Stepping**: The accuracy of the simulation is directly influenced by the choice of time step, Δt. In our system, we used a fixed Δt that was sufficiently small to capture the continuous evolution of particle trajectories while maintaining computational efficiency. If the time step is too large, the approximation of particle motion becomes less accurate, especially when resolving collisions.

To validate the stability, we monitored the total kinetic energy of the system over time for various time steps. As shown in Figure 3, the total energy remained stable with only minimal fluctuations, which can be attributed to floating-point precision. Larger time steps led to noticeable energy drift, but the chosen Δt maintained stability over the full simulation.



(a) Total kinetic energy overtime on a full scale, showing overall stability.

(b) Zoomed-in view of the small energy fluctuations.

Figure 3: Total kinetic energy over time for the system. The left plot shows the full scale, while the right plot zooms in on the energy fluctuations, highlighting their minimal size.

2. **Boundary Conditions**: The simulation uses reflective boundary conditions, where particles elastically collide with the walls of the box. Wall collisions are handled by reversing the particle's velocity component in the direction normal to the wall upon impact. This ensures energy conservation during collisions with the walls and maintains the system's total volume. This is done through `handle_wall_collisions`

3. **Particle Collisions**: Particle-particle collisions are handled elastically, conserving both momentum and energy. The collision detection is based on the distance between two particles, which is compared to the sum of their radii. When a collision occurs, the velocities of both particles are updated according to the elastic collision formula, ensuring that the post-collision velocities reflect the conservation of momentum and energy. This is done through `handle_particle_collisions`.

Impact on Results By carefully controlling time-stepping, boundary conditions, and particle collisions, the potential sources of systematic error were minimized in our simulation. The use of a sufficiently small time step and elastic collision handling ensured that the system's energy remained conserved throughout the simulation. Furthermore, numerical precision did not introduce any noticeable drift in energy, as confirmed by consistent velocity distributions across different simulation runs. Overall, these systematic controls validate the accuracy of the velocity distribution and temperature calculations, which closely match the theoretical Maxwell-Boltzmann distribution.

Consistencies with Theory With a large number of particles in a given volume, their interactions must be taken into account with Van Der Waal's equation. As discussed in the Introduction, we approximated our system with ideal gas laws, assuming we had a sufficiently sized volume and a small number of particles. That being said, we should find in increase in error if we increase the number of particles. We find this to be consistent using a uniform velocity distribution. Table 2 below gives data displaying the increase in error with increased N.

N	Error (%)
5	7.088
50	14.667
100	14.972
200	15.024
300	15.178

Table 2: N vs Error

3 Conclusion

Through implementation of Monte Carlo methods, we were able to successfully model a physical system of N particles to a relatively high degree of accuracy. Though limited by the computational complexity resulting from particle-particle interactions, we were able to optimize the precision of our data with statistical sampling methods and manipulating the size of our time step. It was integral to carefully consider the theoretical foundations of our system to ensure our results were comparable to physical expectations. By doing so, we maintained a low error margin of approximately 0.5%, and found that rejection sampling was marginally closer to theoretical predictions.