Exploring the Algorithmic Instructions Sufficient for Thermalization in Gas Phase

Anton Morgunov

October 6, 2023

1 Motivation

In statistical mechanics, we try to derive macroscopic variables through properties of individual components such as atoms or molecules. Those macroscopic variables, however, are end results (after all, there is no concept of time in thermodynamics). I was interested in the mechanism by which that *end result* is achieved. Particularly, I was interested to see what kind of instructions should I encode for two kinds of particles (potentially with different masses and temperatures), initially placed in different compartments of a box to a) diffuse b) thermalize.

Warning:

Both the simulation code and this report are work in progress

Also, note that because this is a description (or reflection on) of an educational experience, I will include chronological description of my thought process and the steps I took to get to the final result. The code for this simulation is published on Github.

2 Methods

2.1 Initialization Parameters

The `constants.py` file contains all the parameters that are used in the simulation: box dimensions, initial temperature, number of particles, etc.

2.2 Data Structures

I define a class `Particle` which stores particle position (x, y) and velocities v_x, v_y. Coordinates (x, y) are drawn independently from a uniform distribution defined by the compartments of the box. Initially, I modeled random walk in which velocities v_x and v_y are independently drawn from a uniform distribution $[-1, 1]$. Then, I decided to model the behavior of particles in a gas phase. In the gas mode, velocities are initialized as $v_x = v_0 \cos\phi$ and $v_y = v_0 \sin\phi$ where v_0 is the initial velocity (defined in `constants.py`, constant for all particles) and $\phi = 2\pi n$ where n is drawn from a uniform distribution $[0, 1]$. An important method of the Particle class is `move` which updates the position of the particle according to its velocity:

$$x_t = x_{t-1} + v_x \qquad\qquad y_t = y_{t-1} + v_y \tag{1}$$

Right now, `move` method also reflects the velocity components if the particle hits the wall. Another important method is `check_collisions`. This method checks if the particle collides with another particle, and if it does, computes new velocities for both particles by treating the collision as perfectly elastic.

I also define a class `Game` which stores a list of `Particle` objects. Game class has a `timestep` method, which calls the `move` method for each particle, then tries to resolve collisions if there are any by calling the `check_collisions`. There is a trivial $O(n^2)$ algorithm (which results in the simulation being practical only for up to 200 particles) and a smarter check which distributes particles on the `Grid` (defined as separate class) and only checks for collisions between particles that are in the same cell or neighboring cells.

`Game` method has an `export` method which exports the current state of the game into a python dictionary. In addition, there are methods:

1. `update_compartment_fractions`. Computes fraction of particles of second kind in left and right compartments.

2. `compute_velocity_distribution`. Computes the distribution (as a histogram) of v_x^2, v_y^2, and $v_x^2 + v_y^2$.

3. `find_equipartition_temperature`. Computes the temperature according to:

$$\langle KE \rangle = \frac{3}{2} k_B T = \frac{1}{2} m \sum_i v_i^2 \implies T = \frac{m}{3 k_B N} \sum_i v_i^2 \tag{2}$$

2.3 Simulation Visualization

I use a `Flask` library to create a server, which provides several endpoints:

1. `/game_state` which calls the `timestep` method of `Game` class
2. `/game_reset` which reinitializes the game.
3. `/game_analyze` which exports the evolution of temperature and compartment fractions as a function of timestep as Plotly figures.

These endpoints are called by the `index.html` file, which uses `JavaScript` to populate the `HTML` canvas with the current state of the game. In addition, the velocity distributions, compartment fractions, and current temperatures are plotted using `Chart.js`.

3 Results

3.1 Preliminary Exploration with Random Walk

At first, I created $n = 200$ particles with identical mass $m = 1\,\mathrm{amu}$ and launched the random walk. The box was large enough for practically all particles move in a small area around their initial position, never meeting other particles.

Questions for further exploration:

From what distribution should I draw the initial velocities to properly model Brownian Motion? That is, what is the interval and how is the probability distributed within that interval?

At some point, I started wondering: why are we even considering random walk? How can a particle move with one velocity and then at the next step abruptly change it? If the particle changes not just the direction but also the magnitude of velocity, that clearly violates the conservation of momentum. The abrupt change in direction is unrealistic because it implies infinite acceleration. I started reading more about the Brownian motion and, to my embarassment, realized that I confused the appearance of a mechanism for the mechanism itself. In other words, it only seems that particles are moving in a random walk, but there's nothing random about the walk of each particle. The *abrupt* changes in direction are actually the result of collisions with other particles.

That's how I ended up implementing the $v_x = v_0 \cos\phi$ and $v_y = v_0 \sin\phi$ velocity initialization scheme. I also created the collision resolution method, which treats all collisions as perfectly elastic. To my surprise, I started seeing that some particles started to move slower, and some particles started to move faster. After sufficient timesteps, the distribution of speeds started to resemble the Maxwell-Boltzmann distribution. This seems to imply that the mechanism for thermalization is the collisions between particles.

Warning:

As we will see shortly, there are reasons to believe that I have bugs/mistakes in the current implementation of the collision algorithm. If that's the case, it may turn out that the correct implementation of the collision algorithm will not result in the Maxwell-Boltzmann distribution.

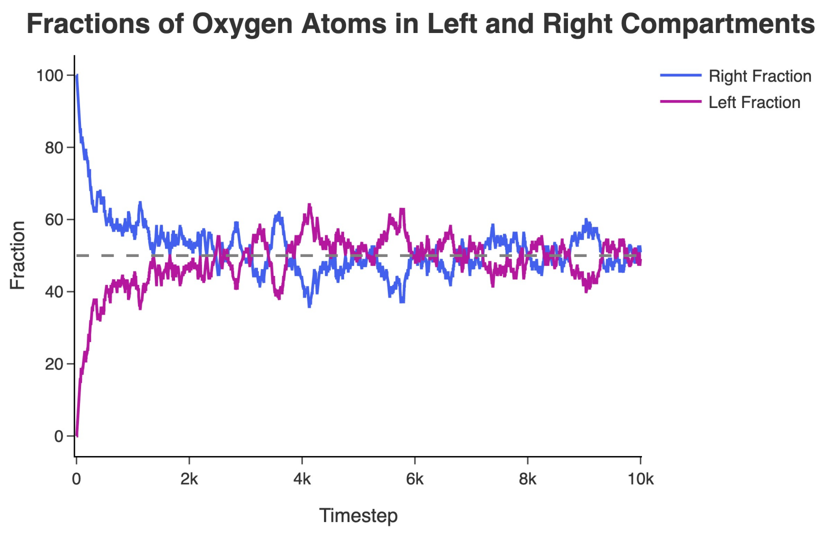

Figure 1: Evolution of compartment fractions

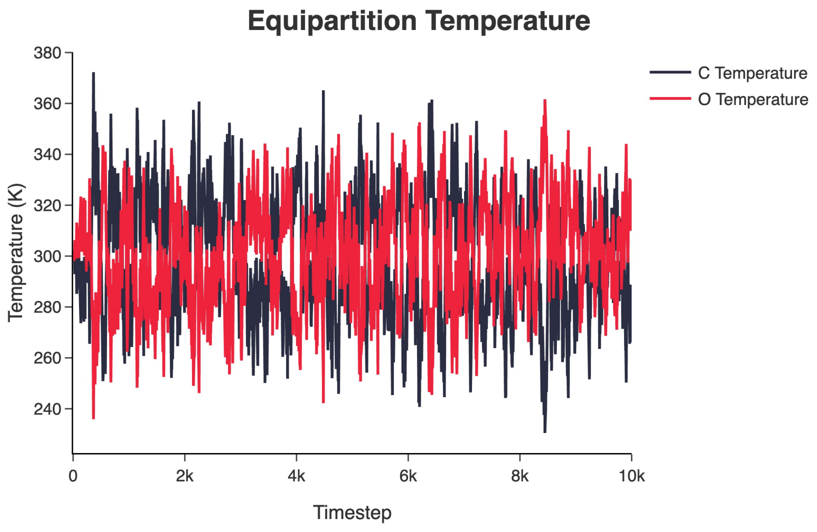

Figure 2: Evolution of temperature

3.2 Identical particles

If we set both kinds of particles to have $m = 1$ and initial temperature $T = 300\,\mathrm{K}$, and create $n = 200$ of those, we can see that particles diffuse and thermalize (Figures 1 and 2). Somewhat not surprisingly, the average temperature of the system is near $T = 300\,\mathrm{K}$, but fluctuates quite significantly.

3.3 Different masses

One of my primary objectives was to fill the left compartment of the box with, say, carbon atoms at T_1 and the right compartment with, say, oxygen atoms at T_2, and see whether/how the two sets of particles thermalize. The choice of elements is superficial, it's suffice to choose two elements with different masses (perhaps H and He would be a better choice). Importantly, $T_1 \neq T_2$. If the final temperature matches the one predicted by the heat transfer equation, that would further solidify the idea that the mechanism for thermalization is the collisions between particles.

Unfortunately, when I fill the compartments even with $T_1 = T_2 = 3000\,\mathrm{K}$ (such high temperature is chosen for particles to move fast enough), after some timesteps the temperature starts to blow up (Figure 3). Obviously, this indicates that there has to be a bug in my code, and I suspect it has to do with the collision resolution algorithm. As much as I'm tempted to keep digging until I resolve this, I have to postpone it.

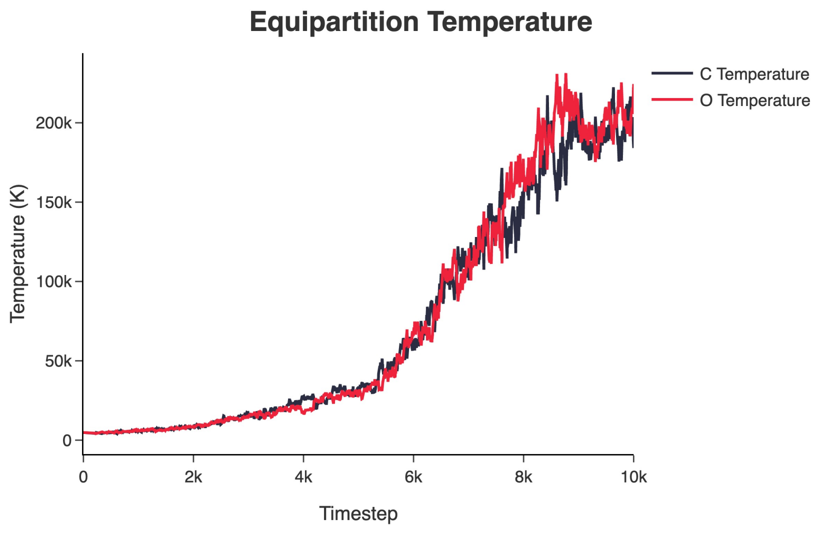

Figure 3: Explosion of Temperature when particles of different masses are introduced

4 Future Work

Here's a list of tasks I could potentially do:

1. Find the mistake in the simulation algorithm
2. Expand the UI of the simulation to allow for more customization: choosing elements, initial temperatures, numbers of atoms.
3. Perform the simulation (once implemented correctly) with different number of particles and see whether the magnitude of fluctuations in temperature scales as $1/\sqrt{N}$.
4. Perform the simulation with different initial temperatures and see whether the final temperature matches the one predicted by the heat transfer equation.

5. Investigate inelastic collisions.
6. Introduce the potential energy term into the simulation. That is, add interatomic interactions.
7. Trace a trajectory of movement of a single particle in the box.
8. Make particles to have non-zero volume. I.e. have particles with different sizes.
9. Explore whether we can model chemical reactions.
10. Introduce unit tests for significant parts of the code.
11. Improve documentation.