Investigating Thermodynamic Systems with a 2D Simulation of a Gas

Georgios Alevras

***Abstract* — This report presents an investigation of thermodynamic systems by making a 2D simulation of a gas using Object-Oriented Programming (OOP) in Python. The equations of kinetic theory and mechanics for a 2D gas within a container were developed, which were then used to develop code to simulate it. The simulation was run with scripts in order to test it, ensuring it makes physical sense, as well as to verify some equations and conditions of thermodynamic systems; the ideal gas law, the Van der Waal's equation and the Maxwell-Boltzmann distribution. The simulation demonstrated it follows the ideal gas law, and the two parameters of Van Der Waal's equation were found to be: $\boldsymbol{a = 4 \times 10^{-40} \pm 10^{-40}\ Pa\, m^4}$, and $\boldsymbol{b = 1.113 \times 10^{-20} \pm 0.001 \times 10^{-20}\ m^2}$. The average fluctuation in total energy and total momentum of the particles of the gas were $\boldsymbol{\overline{\Delta E} = 4.81 \times 10^{-36}\ J}$ and $\boldsymbol{\overline{\Delta p} = 1.21 \times 10^{-40}\ kgms^{-1}}$, very small, effectively demonstrating the conservation of energy and momentum of the system and the accuracy of OOP in simulating this thermodynamic system.**

I. Introduction

Ever since computers first made their appearance, modelling and simulation have become one of the main ways of exploring, testing and experimenting with physical scenarios. The plethora of advantages that come with simulation: repeating the simulation with different variables, "studying problems at different levels of abstraction" [1], or running more 'experiments' in a smaller time than what would be possible with a real experiment, are what give it the power and practicality that it has.

One of the most practical ways of implementing simulations, especially those that can be thought of as describing some 'thing', is OOP. The ability of OOP to make code more modular, flexible and robust, by hiding unnecessary code and using a higher level of abstraction, makes it a great candidate for simulations [2]. Hence, OOP was used in this investigation for a 2D simulation of a gas, where each 'ball' is treated as an object.

In this investigation I simulated hydrogen atoms in a container, and observed the physical properties that they exhibited. The modularity of OOP enabled me to produce short and easily manipulated scripts, in order to test the sensible limits of the simulation.

II. Theory

A. Ball Collisions in 2D

In this simulation all collisions are treated as elastic, thus, assuming both kinetic energy and momentum conservation. As the balls have velocity components in both the $x-$ and the y$-$ direction, we could compute the change in each component. However, it is more useful to consider the new velocity vectors after a collision for two colliding balls:

$$\overrightarrow{v_1} = \overrightarrow{u_1} - \frac{2m_2}{m_1+m_2} \times \frac{\overrightarrow{u_{rel}}\cdot\overrightarrow{r_{rel}}}{|\overrightarrow{r_{rel}}|^2} \times \overrightarrow{r_{rel}}, \quad (1)[3]$$

$$\overrightarrow{v_2} = \overrightarrow{u_2} + \frac{2m_1}{m_1+m_2} \times \frac{\overrightarrow{-u_{rel}}\cdot\overrightarrow{-r_{rel}}}{|-\overrightarrow{r_{rel}}|^2} \times -\overrightarrow{r_{rel}}, \quad (2)[3]$$

where $\vec{u}_1$ and $\overrightarrow{u_1}$ are the two initial velocity vectors, $\overrightarrow{u_{rel}} = \overrightarrow{u_1} - \overrightarrow{u_2}$, $\overrightarrow{r_{rel}} = \vec{r}_1 - \vec{r}_2$, and m_1 and m_2 are their two masses respectively. The momentum change after a collision is given by $\Delta\vec{p} = m\frac{\Delta\vec{v}}{\Delta t}$, and its magnitude by $\Delta p = m\left|\frac{\Delta\vec{v}}{\Delta t}\right|$, which will be important for calculating pressure, as shown later.

B. Collision Time

The time of collision for ball-ball or ball-container collisions is found by solving the following dynamics eqn:

$$(\overrightarrow{r_{rel}} - \delta t\, \overrightarrow{u_{rel}})^2 = R_1 \pm R_2, \quad (3)$$

where R_1 and R_2 are the respective radii of the two colliding objects. If one of the two objects is the container, then R_2 is subtracted – as the ball is inside it – and if both objects are balls, then R_2 is added, i.e. there is no overlap. Finally, the time of collision, δt is given by re-arranging and solving for:

$$\delta t = \frac{-\overrightarrow{u_{rel}}\cdot\overrightarrow{r_{rel}} \pm \sqrt{(\overrightarrow{u_{rel}}\cdot\overrightarrow{r_{rel}})^2 - |\overrightarrow{u_{rel}}|^2 \times (|\overrightarrow{r_{rel}}|^2 - (R_1-R_2)^2)}}{|\overrightarrow{u_{rel}}|^2}. \quad (4)$$

If δt has no real solutions, the two objects have no crossing paths and won't collide. Negative solutions reflect collisions in negative time – moving backwards – and only positive solutions reflect collisions that will occur. However, this equation disregards the fact that balls can't pass through each other, so only the smallest positive δt solution is useful.

C. Pressure Calculations

We know that pressure can be expressed as:

$$P = \frac{F}{A}. \quad (5)$$

The force is given by the sum of the forces exerted on the container by each ball, given by Newton's II Law:

$$P = \sum_i \frac{F_i}{A} = \sum_i m_i \frac{|\Delta\vec{v}_i|}{A\Delta t_i}, \quad (6)$$

where $m_i, \Delta t_i, \Delta\vec{v}_i$ are the mass, time of collision and velocity change of each ball, and A is the area of the container. As this is a 2D simulation, the area – the contact area between balls and container – is the container circumference.

D. Van der Waal's Equation

Initially it can be expected that the gas will follow the ideal gas law:

$$PV = Nk_BT. \quad (7)$$

However, Eqn. (7) assumes point-like particles, which isn't the case in this simulation. To correct for this, Van der Waal introduced two parameters, shown in Eqn. (8):

$$\left(P + a\frac{N^2}{V^2}\right)(V - Nb) = Nk_BT. \quad (8)$$

The first, $a\frac{N^2}{V^2}$ corrects for inter-particle forces, and the second, Nb corrects for the volume occupied by the particles. a is expected to be zero, as there are no inter-particle forces and the balls are isotropic. However, b is expected to have some finite positive value.

III. OOP Implementation

In this investigation OOP in Python was used for the simulation. Figure 1 highlights the key attributes, methods and relations of the three classes used in the simulation.

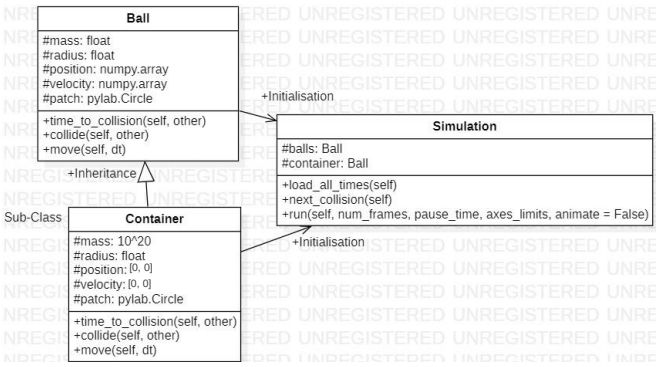

Fig. 1. UML diagram of classes used in the simulation. '#' refers to protected attributes and '+' refers to public methods. Only key methods are shown. Software: [4]

The simulation class is initialised with a list of ball objects and a container object. When the *run()* method is called, it is parametrised with the number of frames, pause time between frames, axes limits for the visualisation and a boolean for the animation. The balls and the container are parametrised at the beginning of each script. All methods in the simulation are completely parametrised, avoiding any hardcoding, thus, making the code more flexible.

The three classes shown in Figure 1 are contained within one file, *Simulation.py*, which is imported in each script. The parameters for the balls and the container are instantiated at the beginning of each script, and then a container and a simulation object are initialised with those parameters, as shown in *TestingSimulation.py*, lines 20-31.

The investigation consists of: *Simulation*.py, containing the simulation classes, *TestingSimulation.py*, testing the simulation for sensible limits and conservation laws, and three scripts used for obtaining pressure laws. In each of the three scripts, the simulation is initialised within a loop with different variables, and the outputs, e.g. pressures for different temperatures, are stored in arrays, avoiding storing in text files, thus, keeping each script self-contained.

Initialising the position of the balls within the container without any overlap quickly became an issue when multiple balls were used, as any nested-loop would perform N^2 calculations at minimum. In order to increase the efficiency of initialising their position, without trading for non-random positions, I created a recursive algorithm which decreases the time-complexity, hence, allowing to generate 100s of balls in less than 1s (see *Simulation.py*, lines 354-397). This method finds the distances between the new randomly generated ball and all other existing balls, and only compares it with the minimum distance. It calls itself again until this minimum distance is bigger than twice the radius of a ball, and then moves onto the next ball, until all balls have a random, non-overlapping position.

Another algorithm that was important to develop was *velocities_at_given_temp()*, which gives all balls a random velocity, so that their sum temperature is equal to the thermal temperature that one wants to set the gas with. It does so, by giving the first ball a temperature, obtained from a normal distribution around $\frac{1}{N}T$, where T is the gas temperature, and N is the number of balls, and then decreases T for the next ball by that calculated temperature. This enabled me to control the variable T in Eqn. (7) and Eqn. (8) as an independent variable and perform tests.

IV. Results and Discussion

Initially, the simulation was checked that it looked 'normal'. Figure 2 shows the simulation when initialised with 100 balls.

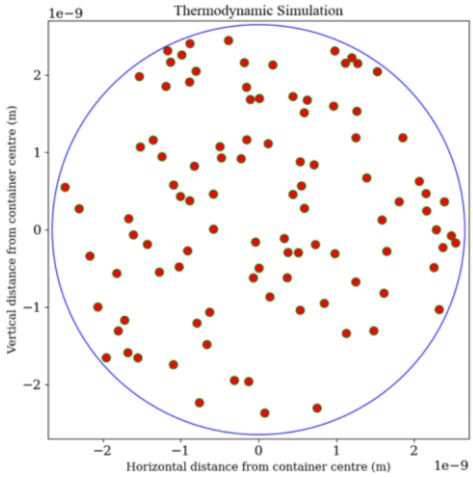

Fig. 2. Simulation initialised with 100 balls, of mass and radius of a hydrogen atom: $m = 1.6736 \times 10^{-27} kg, r = 53\ pm$, at $300K$. Container radius is 50 times larger, $R = 2.65\ nm$.

Figure 2 demonstrates that there are no overlapping balls, and that they are randomly positioned. However, a more quantitative way of evaluating the sensible limits of the simulation is to run it for a few thousand collisions, and plot their motion. Figures 3 and 4 show histograms of the distances of the balls from the centre of the container, and the relative distances of all balls respectively.

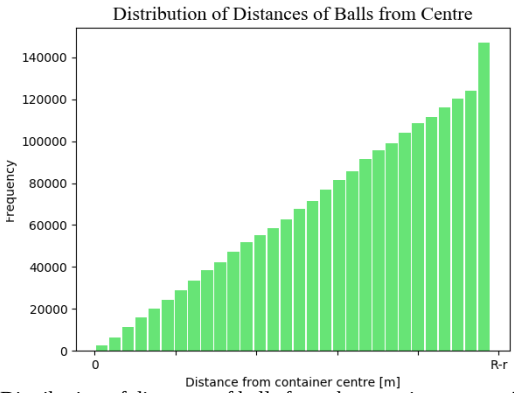

Fig. 3. Distribution of distances of balls from the container centre: 50 balls, at $300K$, for 40,000 frames.

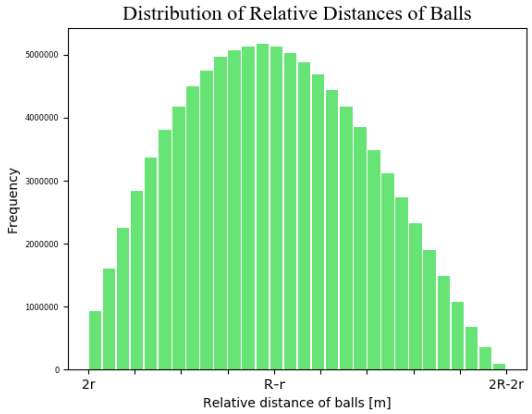

Fig. 4. Distribution of relative distances of balls from the container centre: 50 balls, at $300K$, for 40,000 frames.

Figure 3 shows that no ball has escaped the container during the simulation, as the distance distribution has an upper bound, $R-r$. It also shows that the probability increases linearly, as the rate of increase of the area of the container goes $\propto R$. The increased probability of finding a ball near the container walls shows that balls get more easily trapped there as there are other balls pushing them only from one side, and the container wall stops them from leaving.

Figure 4 demonstrates that no balls overlap during the simulation, as the distribution is bounded by the sum of two ball radii, $2r$ and the difference between container and ball diameters, $2R-2r$. The highest probability for the relative distance is approximately the difference between container and ball radius, $R-r$. The bias towards $2r$ occurs because the relative distances are measured during each consecutive collision. The smaller the balls are compared to their container, the more this distribution will approximate the nearest-neighbour distribution [5].

Another test that can ensure the simulation is running sensibly is to check conservation laws. Figures 5 and 6 show the total kinetic energy and the total momentum of the gas during a run of the simulation.

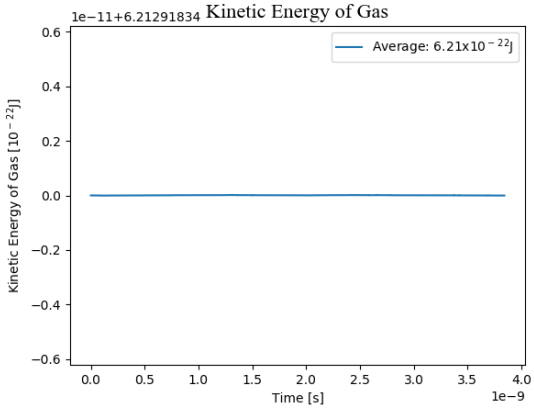

Fig. 5. Total kinetic energy of gas for 50 balls, at $300K$. Average total kinetic energy: $6.21\times10^{-22}J$. Mean fluctuation from the mean: $\overline{\Delta E}=4.81\times10^{-36}\,J$

Figures 5 and 6 show that the total kinetic energy and momentum of the gas are constant; confirming their conservation. The mean fluctuations in total kinetic energy and momentum from their means are $\overline{\Delta E}=4.81\times10^{-36}\,J$ and $\overline{\Delta p}=1.21\times10^{-40}\,kgms^{-1}$ respectively, which are almost zero. It is important to note, that the container does not actually have an infinite mass, but rather a mass of

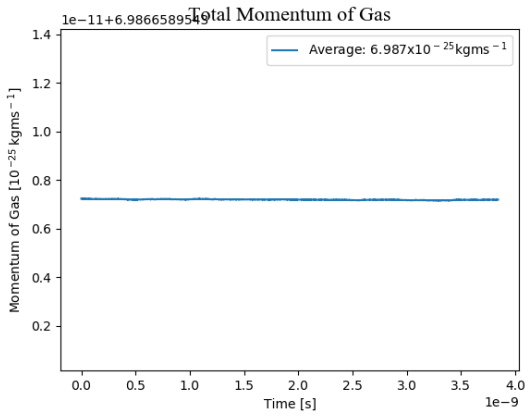

Fig. 6. Total momentum of gas for 50 balls, at $300K$. Average total momentum: $6.987\times10^{-25}kgms^{-1}$. Mean fluctuation from the mean: $\overline{\Delta p}=1.21\times10^{-40}\,kgms^{-1}$.

$M=10^{20}kg$, hence, the system is not perfectly elastic. Considering this and the tiny fluctuations, the simulation can be trusted.

The first thing that we want to observe from the simulation is that the balls, the hydrogen atoms, exhibit a pressure on the container. Figure 7 shows the pressure of the container for 40,000 frames.

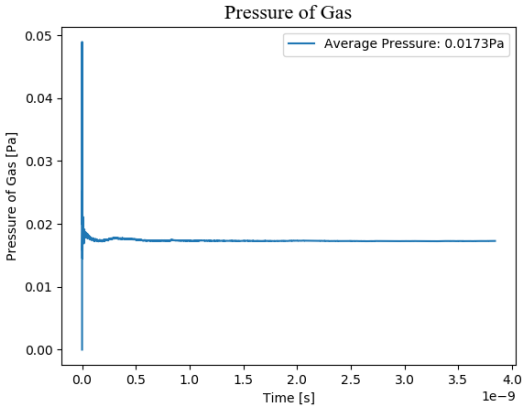

Fig. 7. Pressure of the gas over time, for 100 balls at $300K$ for 40,000 frames.

Initially, the pressure fluctuates, as the changes in time are very small, but it then stabilises to a value. The pressure expected from the ideal gas law, Eqn. (7), for 100 balls, at a temperature of $300K$ in a container of radius $R=2.65\times10^{-9}m$ is $P=0.0188\,Pa$. The obtained pressure is $P=0.0173\,Pa$, deviating by just 7% from the real value. Moreover, the velocity distribution for the hydrogen atoms can be seen in Figure 8.

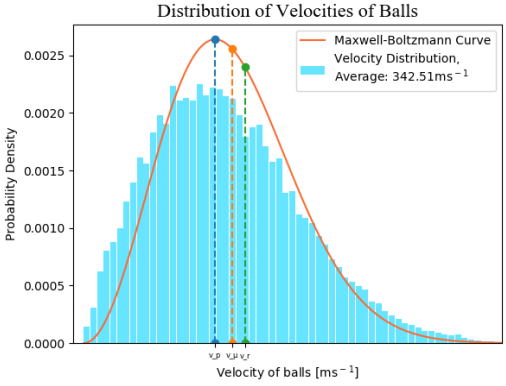

Fig. 8. Velocity distribution (blue histogram), Maxwell-Boltzmann Curve (orange curve), for 50 balls at $300K$ for 40,000 frames.

In Figure 8, v_p is the peak speed, v_μ is the average speed and v_r is the RMS speed. Despite the balls being initially given random velocities, as time passes and they collide multiple times, they follow a Maxwell-Boltzmann distribution. However, the velocity distribution has a small bias towards lower speeds compared to the Maxwell-Boltzmann distribution. This probably occurs because the simulation starts with an uneven distribution, with more balls at smaller speeds, and it hasn't run for a long enough time for them to reach a perfect Maxwell-Boltzmann distribution. The fact that the velocity distribution follows an expected statistical law, and that the pressure and conservation laws are as expected, demonstrates the effectiveness and accuracy of OOP in simulating systems.

Finally, the simulation was run with the same parameters: number of balls, $N = 100$, ball and container radii, $r = 53\ pm, R = 2.65\ nm$, and ball mass, $m = 1.6736 \times 10^{-27} kg$, but different temperatures, ranging from $T = 50K$ to $T = 400K$. The pressure was plotted as a function of temperature, as shown in Figure 9.

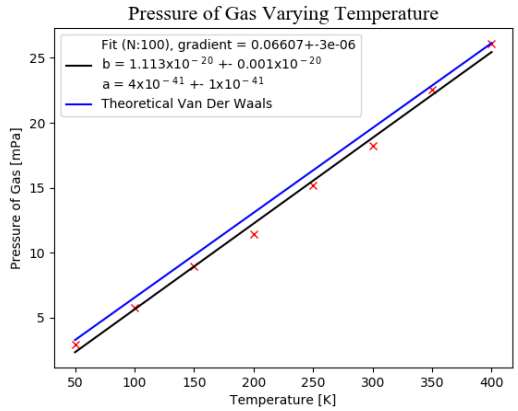

Fig. 9. Pressure of gas varying temperature, for 100 balls, 40,000 frames.

Pressure follows the ideal gas law, as it has a linear relationship with temperature, and the values shown in Figure 9 correspond very well with the expected values, with an average deviation of 5.13%. However, as the hydrogen atoms are not point-like, Eqn. (8) can be re-arranged as such:

$$P = \left(\frac{Nk_B}{(V - Nb)}\right) T - a\frac{N^2}{V^2}, \tag{9}$$

to obtain a from the y-intercept, and b from the gradient, as V, N and k_B remain constant. Using the coefficients obtained from the linear fit, the two constants are found to be: $a = 4 \times 10^{-41} \pm 10^{-41}\ Pa\ m^4$ and $b = 1.113 \times 10^{-20} \pm 0.001 \times 10^{-20}\ m^2$. The obtained value for a is very close to zero, expected as there are no inter-particle forces, while Nb should be equal to the volume occupied by the hydrogen molecules. Given that there are 100, each with a volume of $V_H = 8.82 \times 10^{-21}\ m^2$, then Nb should be $Nb = 8.82 \times 10^{-19}\ m^2$, and b should be $b = 8.82 \times 10^{-21}\ m^2$. The value obtained from the fit deviates by 26.3%. The expected values for a and b are not within the uncertainty given from the fit, however, they are quite accurate. Finally, the linear fit is very close to the theoretical Van der Waal's equation, demonstrating that the thermodynamic simulation has been quite accurate.

V. Conclusions

The aim of this investigation was to develop a 2D simulation of a gas using object-oriented programming in Python and explore computational physics in thermodynamics. To achieve this, I developed the necessary equations of kinetic theory and mechanics, and then developed code to simulate this thermodynamic system. I wrote a few scripts to test the simulation and derive some physical and statistical laws, such as the Maxwell-Boltzmann distribution, the ideal gas law, Van der Waal's Equation and conservation laws.

The conservation laws were found to hold, as both the total kinetic energy and the total momentum were constant, with tiny fluctuations of: $\overline{\Delta E} = 4.81 \times 10^{-36}\ J$ and $\overline{\Delta p} = 1.21 \times 10^{-40}\ kgms^{-1}$, respectively. The hydrogen atoms followed the Maxwell-Boltzmann distribution relatively well, despite their initially random velocities. The constants of Van der Waal's equation were found to be: $a = 4 \times 10^{-41} \pm 10^{-41}\ Pa\ m^4$ and $b = 1.113 \times 10^{-20} \pm 0.001 \times 10^{-20}\ m^2$, which were very close to the expected values.

I was surprised to see how well various physical properties could be shown computationally. OOP was very effective and practical at simulating this system, given its abstraction and flexibility, allowing me to simply import the classes and have small units of code to execute the simulation.

In order to take this investigation further, a 3D simulation of a gas could have been attempted, as well as testing more realistic thermodynamic systems, with different gas molecules, each with their own translational, vibrational and rotational energies, as well as inter-molecule potential energies.

VI. References

[1] Craig D., (1996), "Advantages of Simulation", available at: http://web.cs.mun.ca/~donald/msc/node6.html

[2] Amaral M., (2017), "What are the advantages of OOP?", available at: https://www.quora.com/What-are-the-advantages-of-OOP

[3] Wikipedia (2019), "Elastic Collision", available at: https://en.wikipedia.org/wiki/Elastic_collision

[4] StarUML3, (2019), available at: http://staruml.io/

[5] Chandrasekhar S., (1943), "Stochastic Problems in Physics and Astronomy", Reviews of Modern Physics, 15 (1): 1-89, Accessed on: 12,12,2019, DOI: 10.1103/RevModPhys.15.1