Thermodynamics snookered

Roberto A. King

***Abstract*— A hard sphere model can be used to investigate the properties of a thermodynamic system. Code was written in order to simulate collisions between balls in a chamber. By assigning the balls a radius, velocity and mass similar to that of gas molecules, we can investigate many properties of the system of particles. These calculated properties can then be compared to the thermodynamic properties of an equivalent system to gauge the accuracy of the simulated model. The goal of the following experiment was to simulate as accurately as possible an ideal gas.**

I. Introduction

PYTHON was the programming language of choice for this investigation since it is "Object Oriented", meaning a set of objects could be created and then manipulated with ease. It was a case of creating a few classes and then creating objects from these classes in order to simulate a gas. The main goal of my investigation was to quantitively calculate the pressure, temperature, among other properties of a simulated ideal gas.

II. Theory

In this investigation the only force considered is that when two balls collide and a ball and the container collide. It is assumed that the balls do not interact when not in contact, this is the same assumption made as for an ideal gas. This vastly simplifies the simulation and does not affect the later investigations all that much.

A. Time to Collision

First step was to work out the time to collision between two given balls. The following equation (Equation 1) was given to enable this. δt is the time to the collision between the balls, $\mathbf{r}_{1,2}$ is the position vector of each ball and $\mathbf{v}_{1,2}$ is the velocity of each ball. $\mathbf{v}_{1,2}$ is the velocity of each ball. $R_{1,2}$ is the radius of each ball respectively. The ± sign on the right hand side, symbolises a collision between either a ball and a ball (+) and a ball and the container (-). It's a plus for ball on ball as the sum of their radii will be the minimum distance between both balls. A negative sign means that the container radius minus the ball radius is the furthest the ball will ever get from the origin, with the container simply being a big ball centred on the origin. This ± sign can be ignored and simply set as a + sign by assigning the container a negative radius.

$$(\boldsymbol{r}_1 - \boldsymbol{r}_2 + \boldsymbol{v}_1\delta t - \boldsymbol{v}_2\delta t)^2 = (R_1 \pm R_2)^2 \quad \text{Equation 1}$$

By setting the relative positions and relative velocities as $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and $\mathbf{v} = \mathbf{v}_1 - \mathbf{v}_2$ respectively and $R = R_1 + R_2$, the equation is simplified to equation 2:

$$(\boldsymbol{r} + \boldsymbol{v}\delta t)^2 = R^2 \quad \text{Equation 2}$$

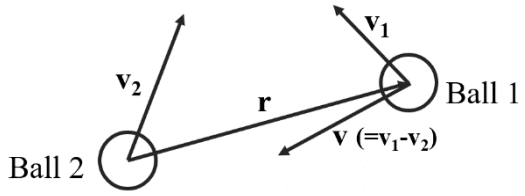

Figure 1: Shows a simplified diagram with the balls' relative velocities and relative positions showing

Equation 2 shows that the relative position from one ball to another plus the time to collision multiplied by the relative velocity between both balls equals the minimum distance between both balls ie. the sum of their radii. Both sides are squared to ensure that no side can be negative. Now solving for δt leads to equation 3:

$$\delta t = \frac{-\boldsymbol{r}.\boldsymbol{v} \pm \sqrt{(\boldsymbol{r}.\boldsymbol{v})^2 - v^2(r^2 - R^2)}}{v^2}$$

Equation 3: An equation to solve for the time to the next collision. Note $r^2 = \mathbf{r}.\mathbf{r}$ and $v^2 = \mathbf{v}.\mathbf{v}$

This δt leads to many solutions; the only solution needed is the lowest, real and positive solution, some if/else statements must be written in the code to find this solution.

B. Collisions

Now in order to collide two balls, expressions must be found that update their velocities, using the balls' masses, their initial positions and their initial velocities. It is assumed that the balls collide elastically and no energy is lost in the collision. This is also true for an ideal gas so is a safe assumption to make. The new velocities after a collision are given by the following:

$$\boldsymbol{v}_1' = \boldsymbol{v}_1 - \frac{2m_2}{m_1 + m_2}\frac{(\boldsymbol{v}_1 - \boldsymbol{v}_2).(\boldsymbol{x}_1 - \boldsymbol{x}_2)}{|\boldsymbol{x}_1 - \boldsymbol{x}_2|^2}(\boldsymbol{x}_1 - \boldsymbol{x}_2)$$
$$\boldsymbol{v}_2' = \boldsymbol{v}_2 - \frac{2m_1}{m_1 + m_2}\frac{(\boldsymbol{v}_1 - \boldsymbol{v}_2).(\boldsymbol{x}_1 - \boldsymbol{x}_2)}{|\boldsymbol{x}_1 - \boldsymbol{x}_2|^2}(\boldsymbol{x}_2 - \boldsymbol{x}_1)$$

Equation 4: shows the new velocities as functions of the old velocities, positions and masses

III. STRUCTURING MY CODE

In order to implement the simulation, 3 classes need to be made: a Ball class that is used every time a new ball is created, a Container class that initialises as a ball with negative radius and a Simulation class that incorporates the two previous classes and runs the simulation every time the run function is called. I made a separate file for my Simulation module and imported BallandContainer into it in order to modularise my code. An overview of the class system is shown by figure 2:

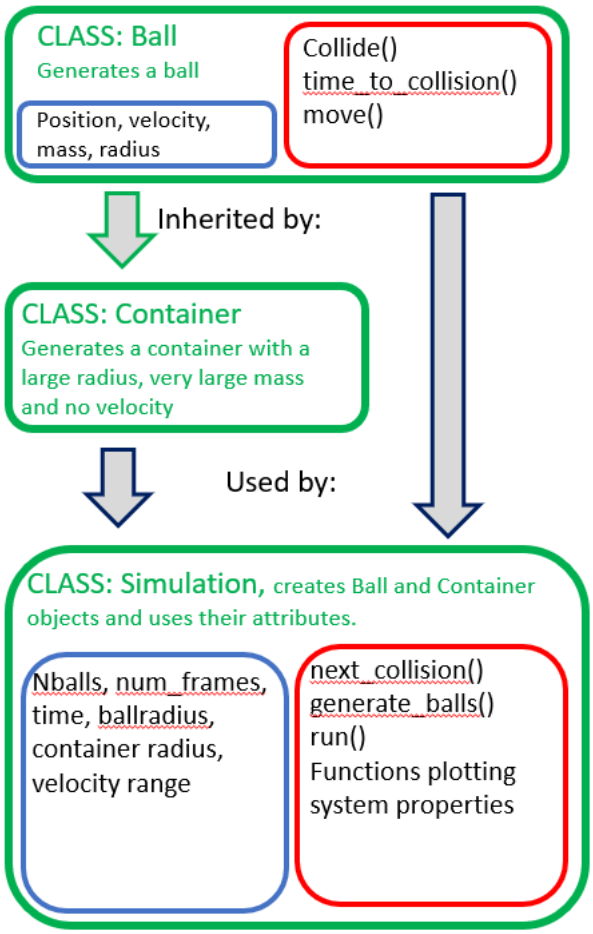

Figure 2: Shows an overview of the class system used, red boxes showing each class' method attributes and blue boxes showing data attributes

The final file that runs the simulation only has to import the Simulation class file and in two lines of code can initialise the simulation object and run it. In order to get plots such as pressure against time or the velocity distribution, separate lines of code must be written.

A. *Ball Class*

The Ball Class is made up of two main functions, the time_to_collision and collide function. The time to collision function uses the method above but before enacting this method, it checks to see that the discriminant is positive. The two answers it returns can be a mix of positive and negative; if statements are used for every combination of positive and negative answers to select the lowest positive answer. The collide function uses the above method to find the new velocities after a collision, it updates the balls' velocity attributes within the function. The Container file simply initialises as a ball with a large (negative) radius, centred on the origin with a very large mass.

B. *Simulation Class*

The Simulation class has a lot more going on, it has numerous important functions. One such function generates the balls by creating N objects of the class ball, positioning them in a square inscribed inside the container. It doesn't matter that their positions are not randomised as the entire system will randomise itself after some time. Initial ball velocities are picked from a uniform distribution with a specified range, the sum of all inital velocities will add up to approximately 0; there is no total momentum of the system in any direction . Another important function is the next collision function, it first makes a list of the time to collision between every possible combination of balls, then picks the lowest time from this list and finds the combination of balls that have this time to collision. The next step is to move every ball in the system by this time, the last step performs the collision between these two balls.

IV. PROBLEMS

I had several problems during the continuous testing of my code. One of the more interesting bugs was that Balls kept escaping from the chamber; I added an exception to stop this bug, since a ball escaping would affect any measurements of thermodynamic properties. In order to find the cause of this bug, I printed out δt for every frame. It was noted that when δt is very small, in the order of 10^{-5}, a ball would escape. A ball would escape because the floating point error on δt could then be bigger than the value of δt. This meant that the ball could then move a negative time, moving it in the opposite direction and out of the container. A solution to this was to create an if statement where if δt was less than a certain number, then it wasn't a real collision and δt was replaced with a big number, δt was then replaced with the next smallest δt. This solution worked most of the time but occasionally a ball still did escape. I decided to also edit the move part of next collision, so that the balls don't move the full amount, if they did move the full amount they could overshoot and move outside of the container.

Another, less interesting, problem I encountered was in my time to collision function. The program was having trouble finding the minimum time in my list of dt. When printing out δt, it was found that some values were returning as none; the min function can only operate on lists of floats and integers. So I took a look at my time to collision function. I found that only 3 of the 4 possible combinations of answers were accounted for, the missing scenario being two negative answers. It was theorised that this scenario should not be possible due to the combination of coefficients in the quadratic from the expanded form of equation 2. However this logic was evidently wrong as adding an if statement to account for this scenario fixed the problem.

V. TAKING DATA

Several functions were written inside the class simulation that are able to take measurements of thermodynamic

properties and also plot functions. One of the hardest graphs to plot was the Pressure against time graph, a sum had to be taken of the momentum change every time a ball hit the container wall. Then a sum had to be taken of all δt s; giving the current time. The plot of pressure is given by equation 5:

$$P(t) = \frac{\Delta mv(t)}{t * 2\pi r}$$

Equation 5: Pressure against time, Δmv(t) is the cumulative momentum change on the wall. The function stabilises after a few hundred frames

Functions were also written to plot histograms of the frequency of different speeds, a distribution of distances from the origin and a distribution of distances between two balls.

VI. ANALYSIS

A. *Conservation Laws*

In a classic thermodynamic system it is observed that several quantities are conserved over time; internal energy of the system and pressure on the container wall; for these two quantities to be conserved the number of balls must be constant, hence the clause to stop the code if a ball escapes. It can be seen from the following graphs that these two quantities are indeed conserved.

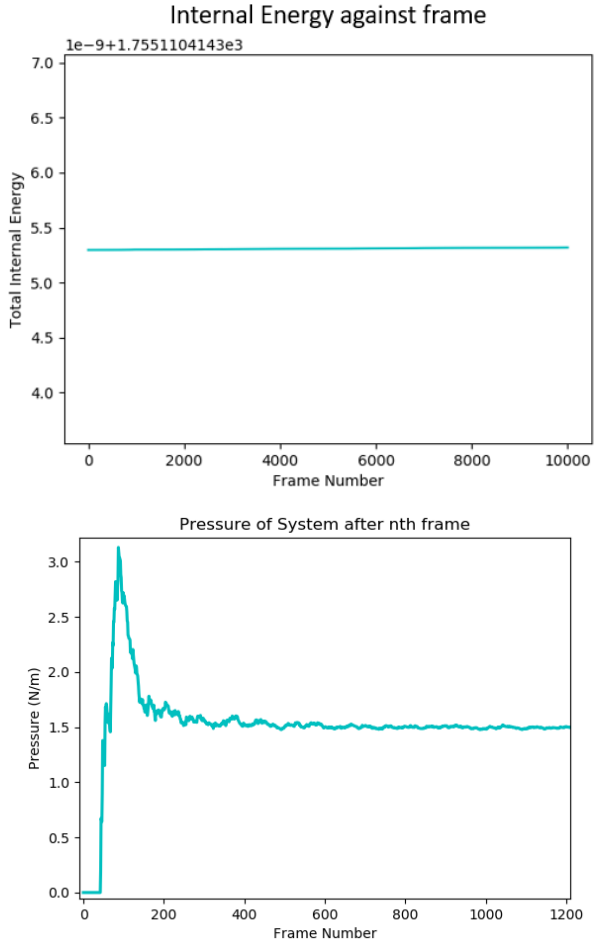

Figure 3: Energy against time, Pressure against time both for N = 50, and an initial velocity range of -10 to 10

These two graphs show that the system conserves energy and pressure. Since the system maintains energy, it will therefore maintain temperature and also average ball velocity.

The initial pressure is very high since within a very small time some balls will hit the top wall since the balls spawn at the top of the container. The pressure then dips and levels out as the balls cross the container and as the collisions between balls and the container wall randomise.

The pressure levels out at 1.48 N/m. This may initially seem like an extremely small value, however as the system only has 50 balls, this makes sense. Using

$$PV = Nk_BT$$

, we can see that as N is a very small number, P is also correspondingly small.

The temperature of the system can be calculated using

$$U = \frac{2}{2}Nk_BT$$

, where U is the internal energy and since there are only 2 degrees of freedom as the system is in 2D the equation has been modified from the original accordingly. The calculated temperature of the above system is T = $2.54*10^{24}$K. This initially seems like an extremely high temperature but we must consider the ball mass and velocity range. The Ball mass is approximately $2.5*10^{25}$ heavier than a nitrogen molecule, so correspondingly U should be $2.5*10^{25}$ times bigger for the simulation. The speed of air molecules is approximately 100 times bigger than that of the balls. Since U goes as v^2, U should therefore be approximately 10^4 times smaller for the simulation. Factoring this in, the U of the simulation should be approximately $2.5*10^{21}$ times bigger. As T is proportional to U, T should also be $2.5*10^{21}$ times larger. Dividing our value of T by this factor gives 1020 K which is in the right order of magnitude.

B. *Velocity Distribution*

My velocity distribution takes the form of a Maxwell-Boltzmann, shown in figure 4.

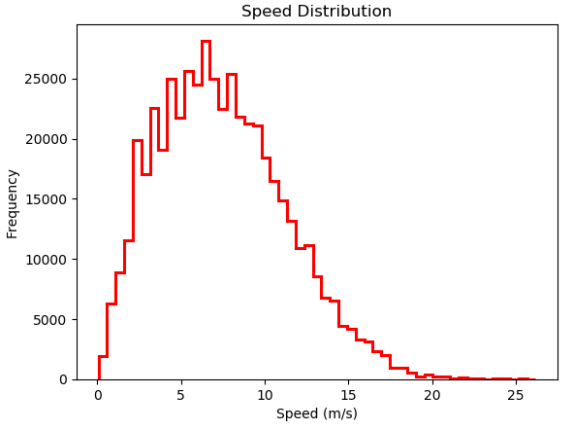

Figure 4: Plotted Histogram showing the distribution of my velocities. Number of balls = 50, over 10,000 frames. Initial velocity range -10 to 10.

The mean velocity is 7.46 m/s and the standard deviation of the distribution is 3.81 m/s. The mean velocity is slightly higher than expected for the velocity range given. Since the initial mean speed in each direction is 5, the initial mean speed should be $\sqrt{5^2 + 5^2} = 7.07$. One possible cause could be that when the balls are generated, the uniform distribution governing the initial velocities has caused there to be more velocities with larger magnitude. Since N is only 50, this is quite likely, especially given the difference is just 5% from the expected velocity.

C. *Pressure vs Temperature*

By varying temperature and measuring temperature, it is possible to obtain Boltzmann constant k_B. A plot of Pressure against Temperature for the simulation is given below.

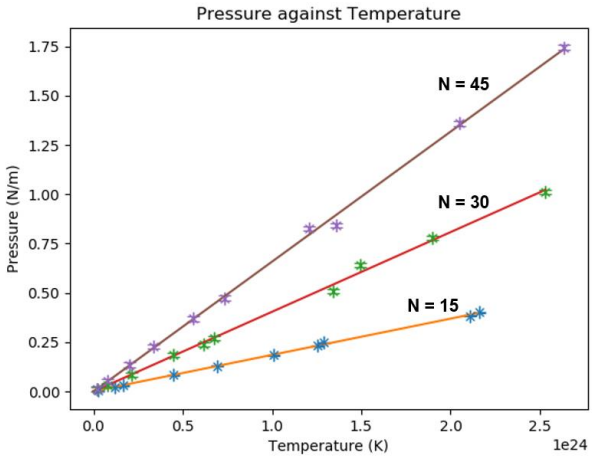

Figure 5: Pressure against Temperature for the simulation with different numbers of balls, over 300 frames. N = 15 gives gradient as $1.84*10^{-25}$
N = 30 gives gradient as $4.22*10^{-25}$ and
N = 45 gives a gradient of $6.51*10^{-25}$

Using:

$$PV = Nk_BT$$

, the gradient of figure 5 is given by $\frac{Nk_B}{V}$. Using the case of N = 45 and taking the area of the container (radius 20) as volume, an estimate of $k_B = 1.82*10^{-23}$ m kg s^{-2} K^{-1} can be obtained. This is 32% off the true value of k_B. This is surprising given the error bars on figure 5 are so small. This large error may be caused by the fact the simulation is not quite of an ideal gas since the particle volume isn't negligible, as in an ideal gas. The ball areas take up 11% of the container area. This is not negligible so the simulation is not ideal.

VII. Conclusion

As can be seen from the plots above, the system can simulate a gas but is some way off simulating an ideal gas and its properties. The simulated gas is a special case as it is an event driven simulation, with a certain number of frames and limited number of balls. A normal computer can only handle so many balls since the time to calculate certain parameters goes as number of balls squared. In order to approximate an ideal gas N will have to be very large and the radii of the balls very small. This is simply not possible on the average computer, vastly limiting the accuracy of the simulation. On the other hand, many properties of a gas were accurately observed, such as P being proportional to T and the velocity distribution forming a Maxwell-Boltzmann, showing that a simulation like this can be a powerful tool in research.

References

(1) https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Map%3A_Physical_Chemistry_for_the_Biosciences_(Chang)/02%3A_Properties_of_Gases/2.8%3A_Molecular_Collisions_and_the_Mean_Free_Path

(2) http://butane.chem.uiuc.edu/pshapley/GenChem1/L14/2.html

(3) https://bb.imperial.ac.uk/bbcswebdav/pid-1678048-dt-content-rid-5350819_1/courses/BJ201910/Y2%20Computing/Scripts/Projects/.build/html/Snooker.html#thermodynamics-investigations

(4) https://bb.imperial.ac.uk/bbcswebdav/pid-1677868-dt-content-rid-5349733_1/courses/BJ201910/Y2%20Computing/Lectures/CompY2_Lecture1_2019.pdf