Computing Project Y2: Thermodynamics Snookered

Vid Homšak
Blackett Laboratory
Imperial College London
SW7 2AZ, UK

I. Introduction

The aim of this project was to produce a working simulation of a container filled with colliding balls. Furthermore, the thermodynamic properties of the system were studied and the arrangement was used to be compared with the properties of an ideal gas. However, since the number and the size of the particles (balls) in the simulation are tens orders of magnitudes different than those of a gas in the container, Van der Waals equation was used to model the "gas" properties, as a direct approach with the equation of state would not suffice.

II. The code structure - user friendliness

The project is quite broad in the sense of the variety of outputs required depending on the user's needs - task(s). The code has thus been put into a layered structure to allow for a fully customizable user-friendly experience.

1) Front Layer - User Interface: To be able to easily obtain various outputs, one only needs to use the *Thermodynamics Snookered - main.py*, where with five functions, all possible outputs can be obtained. Functions' argument inputs allow controlling system's properties relevant for the given output.

2) Middle Layer - Further Modifications: These functions are defined in the *Simulation_types.py*, where they can be further customized i.e their optional parameters can be changed, which were inaccessible from *main.py*, but they will not greatly change the output (thus they were optional). Furthermore, the classes *Simulation* and *Ball* are initialised in these functions, hence their properties can be adjusted by changing their input arguments. However, they've been set in the manner that a user need not change them.

3) Back Layer - Definitions: PositionGenerator.py, Ball.py, Simulation.py, function_fits.py are the files where classes and other functions used in the Middle Layer are defined. They are fully adjustable from the *Simulation_types.py* where they are initialised or called, respectively. Most importantly, the set of arguments used in the initialisation of *Simulation* allows changing the output and determining which calculations run during the simulation(s), based on the function that was called in *main.py*, to make the code as efficient as possible.

III. Results

The output of the project can be separated into three sections, and is obtained by calling the desired functions in *Thermodynamics Snookered - main.py*.

A. Graphical Animation of the Simulation

The animation of the multiple balls colliding in the container is meant to be obtained by calling the function *simulation_animation_without_physics* and is drawn using *matplotlib.pylab* library. It is optimized for speed, hence no physical quantities are being stored during the simulation and no plots will show up at the end (energy/momentum conservation is checked regardless). Other functions also allow displaying the animation by adjusting arguments in the Front (or Middle Layer code), but by default they do not, as it is not their primary purpose and would slow them down significantly.

The function's arguments allow adjusting for the number of frames and the time between each frame of the animation, the number of the balls in the container and its radius R, as well as the balls' initial speed distribution $Norm(\mu, \sigma^2)$, while their directions are determined by a random polar angle.

The balls are initialised in the cells of a square grid of size $N_{balls} \times N_{balls}$ inside the container (see Fig. 1) This enables the user to set the argument N_{balls} as an arbitrary integer, which will then create N_{balls}^2 balls. Hence, balls' diameter d_{ball} is automatically constrained to the size of a lattice cell $l = \sqrt{2}R_{container}/N_{balls}$. In fact, in the two functions (governing section III.A, III.B); $d_{ball} = \alpha l$, where $\alpha < 1$ is the input argument in the latter functions, while the functions mentioned in section III.C allow the user to set the ball's radius as an input argument.

B. Variables as Functions of Time and Distributions Plots

Function *time_dependances_and_distributions* takes the same arguments as the previous one, but also allows the user to set the N_{data_points} number of data points.

This separates the total number of frames into N_{data_points} time periods, over which pressure, temperature, speed and velocity component values are averaged and after which the data for histograms are being collected.

Firstly, to check if conservation laws hold, we plot $KE(t)$ and $p_{x,y}(t)$, which are constant over the time of the simulation (see Fig. 2a, 2b). $\sum_{\mathbf{p}\ total} \neq 0$, as despite random initial directions, N_{balls} is too small for randomization to be sufficient. However, as mass of the container is set very high, the system seemingly does not move.

Moreover, histogram of ball distance d shows a linear relation between d and the time ball spends on the ring with radius d. This makes sense as the area of the ring increases

$\propto d^2$ and so does the time spent on a 2D surface (see Fig. 3). Histogram of inter-ball separations shows an average separation of about, but a bit smaller than $R_{container}$. This is because there are more collisions due to balls ' sizes and because the system has probably not yet fully reached the randomised equilibrium i.e too few frames used (see Fig. 4).

This simulation serves as great means to study kinetic theory of gases. The average speed remains constant with slight fluctuations, as you would expect for a random system. The balls' initial velocities have been randomised, but tests had shown that even with uniform $\mathbf{v}_{initial}$, the system quickly randomises. Furthermore, the average value for $v_{x,y} \approx 0$ with slight fluctuations (see Fig. 5a, 5.b). Similarly, velocity components' distribution (see Fig. 7) indicates a normal distribution centered around 0 ms^{-1} with a very small uncertainty, as expected for random collisions.

The speed distribution can be nicely fitted with a 2D Maxwell-Boltzmann curve $p(v) \propto ve^{-\frac{mv^2}{2k_BT}}$ (see Fig. 6). Moreover, comparing distributions for various speeds we can see that the peak of the distribution shifts right as $< v >$ increases. The peak's height also effectively decreases, since bins get wider as the number of bins is kept constant (see Fig. 8). Regardless, it shows that this simulation seems to model kinematic physics quite well.

C. Equation of State and Van der Waals equation

An important aim of the project was to compare the physics of the simulated system with an actual gas in a container, and check how its thermal properties vary based on various factors. The gas laws are addressed by the three functions in *Thermodynamics Snookered - main.py*, that perform the needed calculations and display P(T), P(V) and P(N) for arbitrary number of different radii of balls, respectively.

Functions' arguments allow the control of balls' mass, radius, initial speed distribution, container size, number of frames in the simulations, number of balls and number of data points to be taken for the plot, as well as the range of different numbers of the balls, used as an independent variable for P(N).

The Van der Waals equation

$$P + a(\frac{N^2}{V^2}))(V - Nb) = Nk_BT$$

is used to (fit the data) model the properties of the system for each radius, since it can be expected that the simulated "gas" would not behave like an ideal gas due to large radii of balls. Furthermore, the numerical data is compared with analytical solution from the equation of state $PV = Nk_BT$.

Since balls are non-interacting particles, constant $a = 0$, while b is to be determined from the plots as a parameter determined by the *scipy.optimize.curve_fit* function. In fact, b represents the excluded volume, which for a "ball" in two dimensions possesses an upper bound of $b < 4\pi r_{ball}^2$, since the minimum distance between two balls is $2r_{ball}$. Thus, let $\epsilon = b/V_{2D} < 4$ [1].

Therefore, plots displaying P(T), P(V) and P(N) were constructed for 10 different radii (see Fig. 9-11). For each radius, the data was fit with a Van der Waals equation, whose fit parameter b was determined with a standard deviation error calculated by np.sqrt(np.diag(covariance_matrix)), where covariance_matrix is one of the outputs from the curve_fit.

Averaging ϵ over each plot for various radii yields values $\epsilon_{P(T)} = (2.1344 \pm 0.0001)$, $\epsilon_{P(V)} = (2.1131 \pm 0.0004)$, $\epsilon_{P(N)} = (1.72577 \pm 0.00005)$, where the smallest radius was left out, since it produces weird ϵ values, but that is probably just due to to the fact that number of frames was insufficient, since for the smallest r the simulation behaves differently, as almost no inter-ball collisions occur for such small N_{balls} that was used in these simulations. Hence, the obtained ϵ (b) values are very feasible as empirically ϵ is usually less than 4 [1].

Furthermore, the data was compared with analytical solution from the equation of state $PV = Nk_BT$, which is represented by a thick red line on the plots (see Fig. 9-11).

One can see, how on all three figures, best fit lines converge towards the equation of state line as radii get smaller, which make sense, since the ideal gas treats balls as point particles.

IV. Tests and error handling

Multiple checks were implemented to ensure that the simulation represents physical reality.

ValueError is raised if kinetic energy and momentum conservation are violated to finite precision. Moreover, if any balls (or the container) overlap, *ValueError* is raised, noting the user by how much they've crossed each other's boundaries. This turned out to be very useful for solving a bug, where due to finite numerical precision, the balls would sometimes overlap when colliding and become entangled for a while or even escape the container.

Similarly, balls must not overlap i.e must fit in the lattice base cell when they are created inside the square grid in the container (see Fig. 1). Hence, *ValueError* is displayed, if parameter $\alpha > 1$ (see section III.A). Similarly, the user must not choose r_{ball} too big, when running gas law checks (section III.C) *ValueError* reminds the user to reduce the radius in the case its value was set too high, since the balls are still created utilizing the grid structure and so their diameter must not exceed the cell's size.

Lastly, for the ease of debugging, setting the global variable DEBUG to 1 in *Simulation.py* and *Ball.py*, causes printing out multiple useful statements about the collision times, pairs collided etc. in every frame, in order for the user to obtain a deeper insight about the mechanisms of the code and allows further checking of the correctness of the simulation.

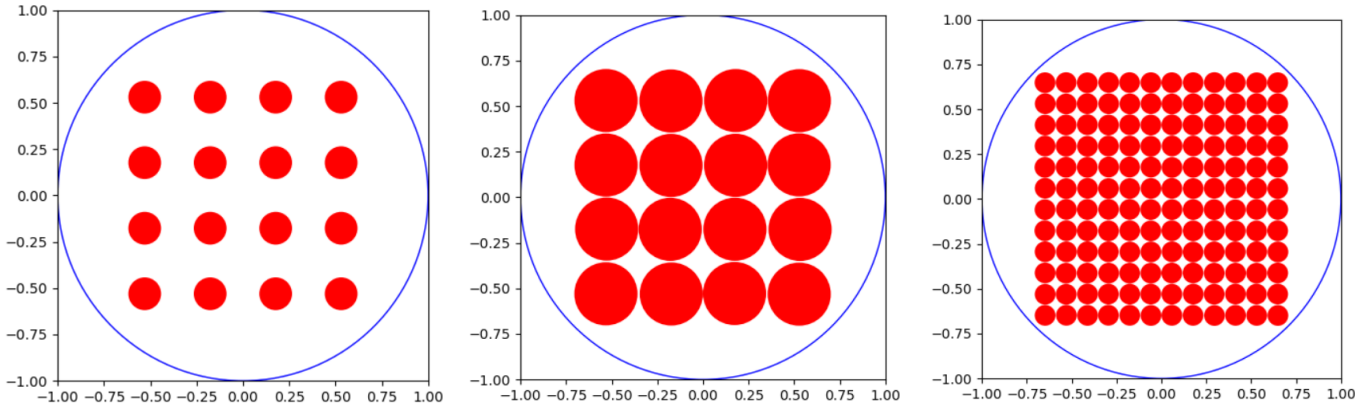
V. Conclusion and improvements

The simulation yielded good results and seemingly models reality well, however the biggest problem was computational power; $N_{balls} << N_A$ and the simulation should run for $t >> 1$s in the simulation reference frame. Calculations implemented functions such as *itertools.combinations* to efficiently find all combinations, tweakable output and optional parameters allowed not to waste calculations on unwanted output... but at the end, multiprocessing or some other programming language should be used to improve the simulation.

References

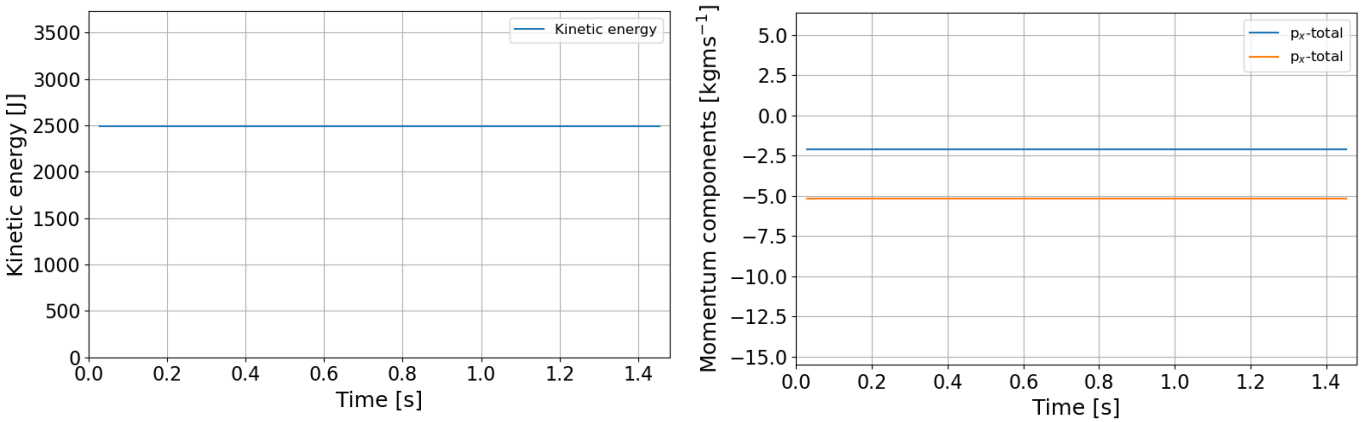
[1] Van der Waals equation. [Online] Wikipedia. Wikimedia Foundation; Available from: Link [Accessed: 5th November2020]

VI. FIGURES



(a) **16 balls,** $\alpha = 0.5$ (b) **16 balls,** $\alpha = 0.98$ (c) **144 balls,** $\alpha = 0.98$

Fig. 1: **Automatic Generation of the Balls**
Balls are initialised in a square lattice with size $N_{balls} \times N_{balls}$. Varying parameter α allows adjusting the balls' radius r_{ball}. In the physics investigation i.e functions that are meant for gas law investigations (see section III.C), however, the user can set the radius as an input value [by the author].



(a) **Kinetic Energy as a function of time.** (b) **Momentum components as a function of time**

Fig. 2: **Conservation of Energy and Momentum.**
49 balls with radius $r = 0.02\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)\text{ms}^{-1}$in a container with $R = 1\text{m}$. 50 data points acquired through 10 000 frames i.e data points averaged over each 200 frames. *ValueError* is raised if any of the conservations are broken [by the author].

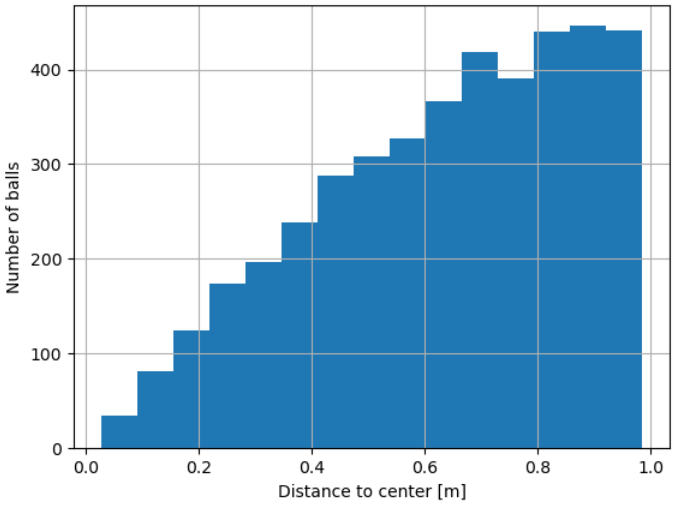


Fig. 3: **Histogram of ball distance from container centre.** 225 balls with radius $r = 0.014\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)$ ms^{-1} in a container with radius $R = 1\text{m}$. 20 data points acquired through 2000 frames [by the author].

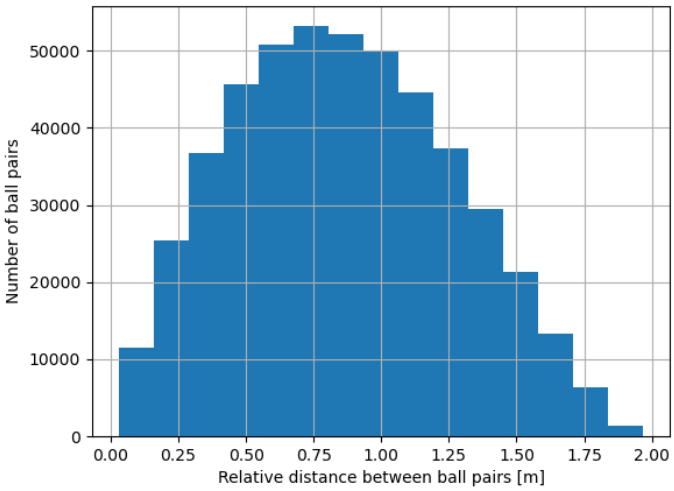
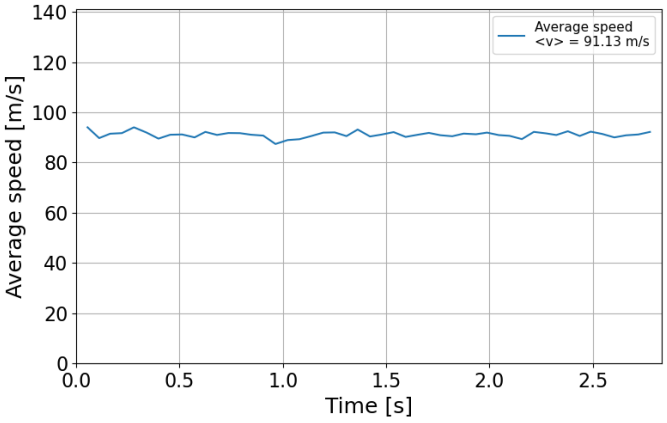
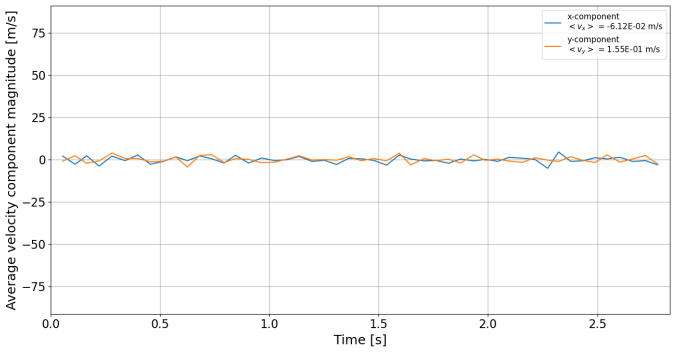


Fig. 4: **Histogram of inter-ball separation.** 225 balls with radius $r = 0.014\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)$ ms^{-1} in a container with radius $R = 1\text{m}$. 20 data points acquired through 2000 frames. Function *Itertools.combinations* was used to efficiently find all possible ball pairs [by the author].



(a) **Average Speed as a function of time.**



(b) **Average Velocity components as a function of time**

Fig. 5: **Speed and Velocity components through the Simulation.** 49 balls with radius $r = 0.02\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)$ ms^{-1} in a container with radius $R = 1\text{m}$. 50 data points acquired through 10 000 frames i.e each data point was averaged over previous 200 frames. Initial direction of particles was randomly determined [by the author].

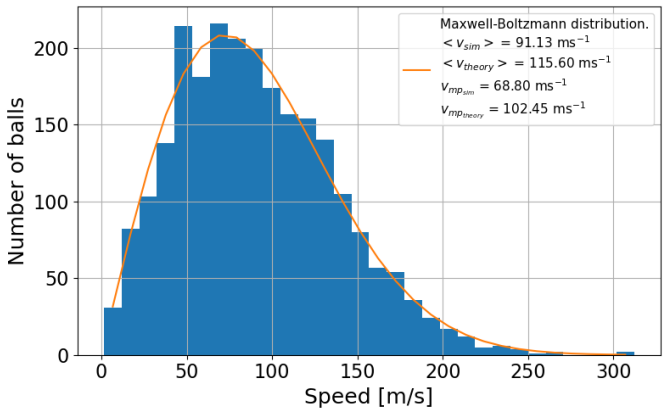


Fig. 6: **Maxwell-Boltzmann Speed Distribution.** 49 balls with radius $r = 0.02\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)$ ms^{-1} in a container with radius $R = 1\text{m}$. 50 data points acquired through 20 000 frames i.e data was collected every 400 frames. Initial direction of particles was randomly determined [by the author].

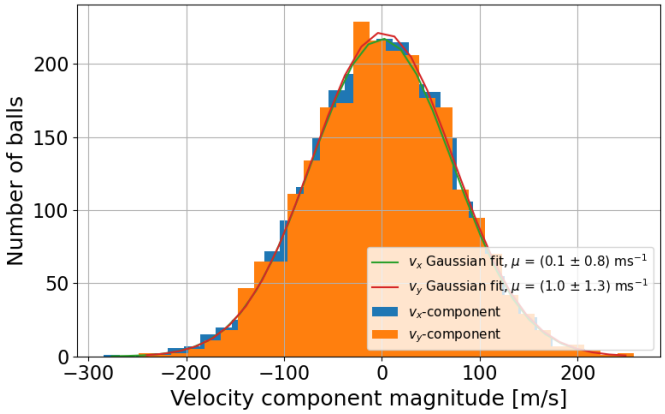
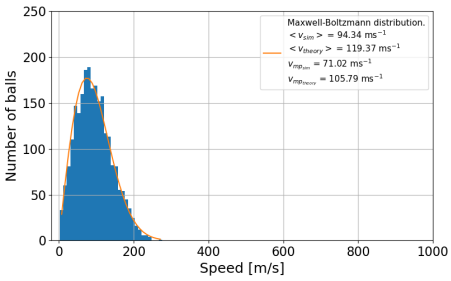
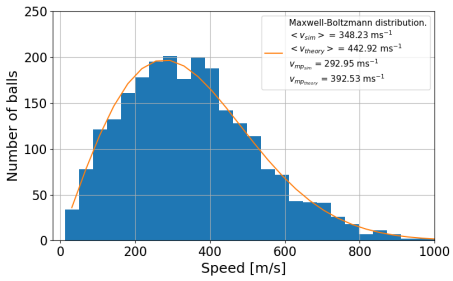


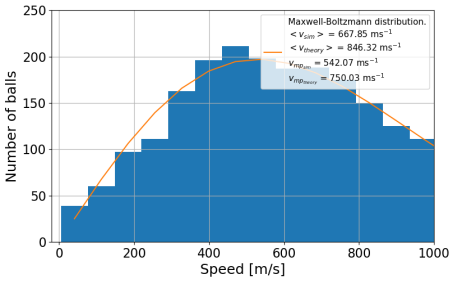
Fig. 7: **Velocity component distributions.** 49 balls with radius $r = 0.02\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)$ ms^{-1} in a container with radius $R = 1\text{m}$. 50 data points acquired through 20 000 frames i.e data was collected every 400 frames. Initial direction of particles was randomly determined [by the author].



(a) **Initial speed distribution:** $Norm(\mu = 100, \sigma^2 = 30)$ ms^{-1}



(b) **Initial speed distribution:** $Norm(\mu = 400, \sigma^2 = 30)$ ms^{-1}



(c) **Initial speed distribution:** $Norm(\mu = 750, \sigma^2 = 30)$ ms^{-1}

Fig. 8: **Maxwell-Boltzmann distribution for different $< v >$.** 49 balls with radius $r = 0.02\text{m}$, mass $m = 0.01\text{kg}$ and initial velocity distribution v = Norm$(100, 30)\text{ms}^{-1}$ in a container with $R = 1\text{m}$. 50 data points acquired through 20 000 frames. Bins get wider as $< v >$ increases, hence effectively decreasing the height of the peak of the distribution. The distributions for higher temperature become wider with a longer ”tail” on the right side of the most probable speed, which also increases [by the author].

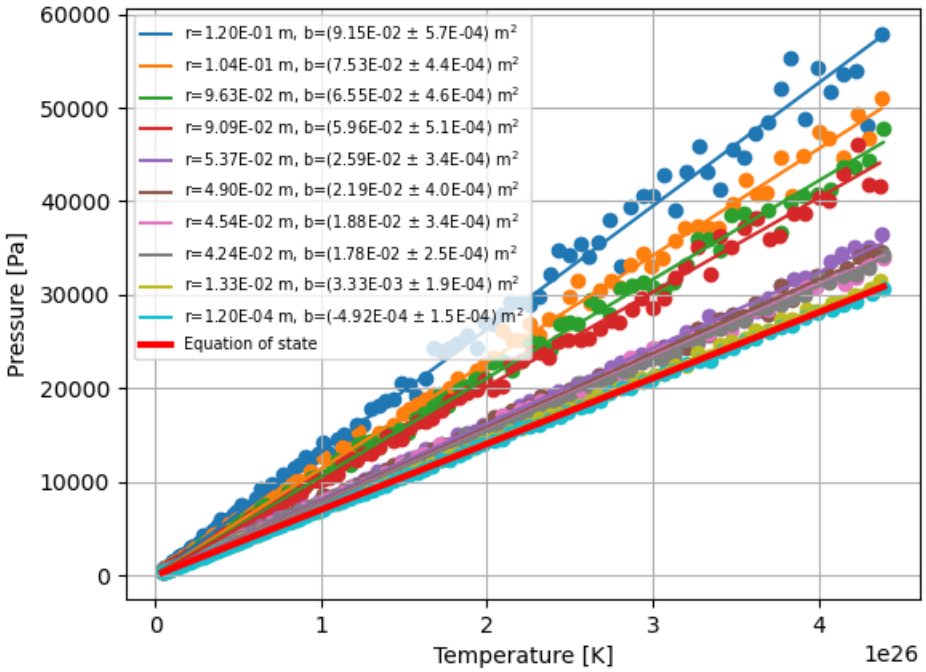


Fig. 9: **Pressure vs Temperature.**
System configuration: 16 balls with initial speed distribution v=Norm($\mu = 100 - 1000\mathrm{ms}^{-1}, \sigma^2 = 3\mathrm{m}^2\mathrm{s}^{-2}$), mass $m = 0.01\mathrm{kg}$ in a container with $R = 1\mathrm{m}$. 10 different radii, 100 variations per radius. Data taken from 5 data points on 200 frames.
Best Fit Lines: For each radius, Van der Waals eq. is used to fit the data points. The lines of best fit approach the thick red line representing the ideal gas assumption, as radii get smaller. $< b/V_{ball} >= 2.1344 \pm 0.0001$ based on 9 biggest radii points [by the author].

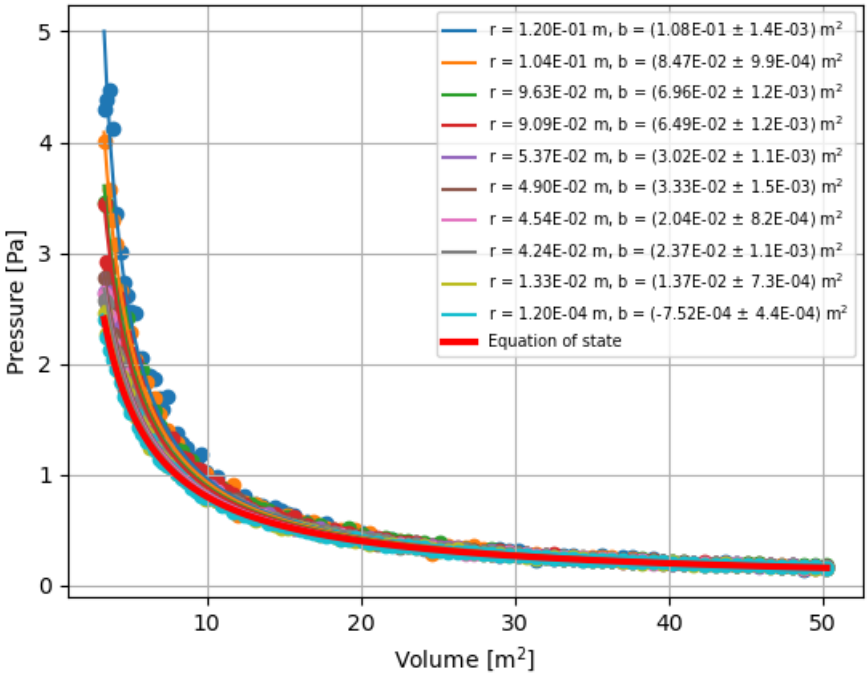


Fig. 10: **Pressure vs Volume**
System configuration: 16 balls with initial speed distribution v=Norm($\mu = 100\mathrm{ms}^{-1}, \sigma^2 = 3\mathrm{m}^2\mathrm{s}^{-2}$), mass $m = 0.01\mathrm{kg}$ in a container with $R = 1\mathrm{m} - 4\mathrm{m}$. 10 different radii, 100 variations per radius. Data taken from 5 data points on 100 frames.
Best Fit Lines: For each radius, Van der Waals eq. is used to fit the data points. The lines of best fit approach the thick red line representing the ideal gas assumption, as radii get smaller. $< b/V_{ball} >= 2.1131 \pm 0.0004$ is based on 9 biggest radii [by the author].

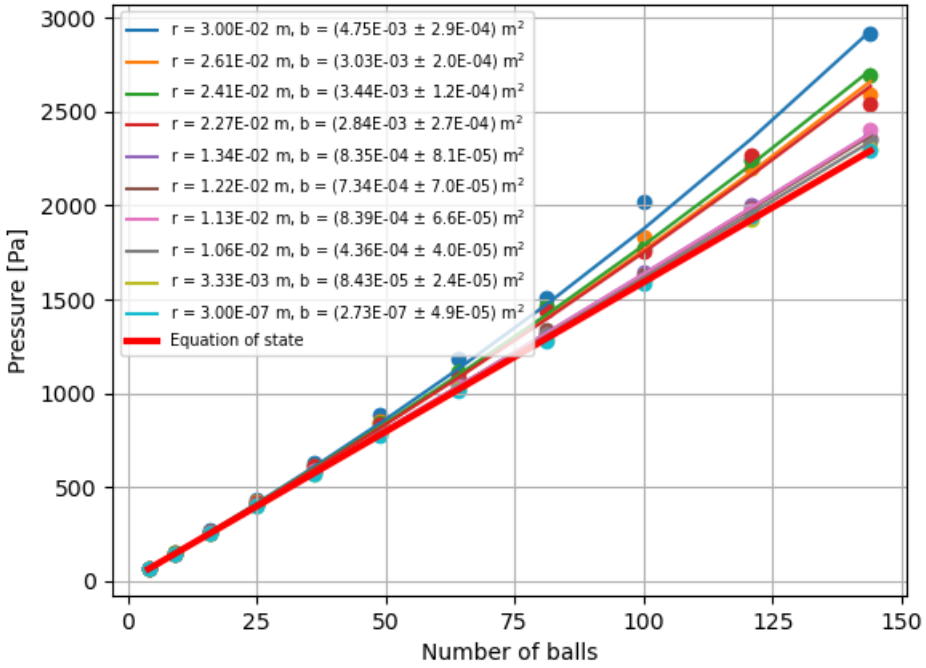


Fig. 11: **Pressure vs Number of Balls**
System configuration: 4-144 balls with initial speed distribution v=Norm($\mu = 100\mathrm{ms}^{-1}, \sigma^2 = 0.1\mathrm{m}^2\mathrm{s}^{-2}$), mass $m = 0.01\mathrm{kg}$ in a container with $R = 1\mathrm{m}$. 10 different radii, 100 variations per radius. Data taken from 10 data points on 200 frames.
Best Fit Lines: For each radius, Van der Waals eq. is used to fit the data points. The lines of best fit approach the thick red line representing the ideal gas assumption, as radii get smaller. $< b/V_{ball} >= 1.72577 \pm 0.00005$ is based on 9 biggest radii [by the author].