Entropy and the Ideal Gas

Thomas Conibear

PHYS 300

Dr. Qiang Zhu

**Introduction**

Two prominent concepts in thermodynamics are ideal gases and entropy. The most important features of an ideal gas are that it only experiences forces during collisions, each collision is always elastic, and the particles are identical with negligible volume. This simple view of a gas facilitated the development of formulations for a gas’s entropy. Entropy is generally thought of as the disorder of a system, but a more accurate description is related to a system’s macrostate and multiplicity.

A macrostate is determined by some measurable aspect of a system like temperature, pressure or volume. Macrostates consist of microstates, which are the states of each individual particle making up a systems macrostate. These two concepts lead to multiplicity, which is the number of microstates corresponding to a given macrostate. The larger the multiplicity, the more ways a certain macrostate can be created. Multiplicity is intimately related with entropy.

The equation for entropy, S, is , where k is Boltzmann’s constant () and Ω is multiplicity of the system. It is apparent that entropy is a scaled value of multiplicity with units Joules per Kelvin as Ω is unitless, and that entropy increases as multiplicity does. In fact, entropy is the result of a necessity to reduce incredibly large Ω values into smaller numbers that are easier to work with.

Entropy is useful to describe any gaseous system. An interesting system to analyze is the mixing of two monoatomic gases after a partition between them is removed. This process demonstrates how multiplicity and entropy will naturally increase until an equilibrium is reached. This describes the second law of thermodynamics which states that “any large system in equilibrium will be found in the macrostate with the greatest entropy”. So, when two gases mix their eventual equilibrium state will be the one with the largest entropy. I aim to demonstrate this using python on helium and argon gas particles. In addition, I aim to simulate various mixing scenarios of identical monoatomic gases separated by a partition in the python library vpython. Within this simulation I will calculate entropy of the entire container’s volume being filled with helium and compare these values to the entropy of argon and helium mixing.

**Methodology**

Figure One. Two monoatomic gases, He and Ar, divided by a partition. Helium particles are red and Argon particles are blue.

Figure one is a diagram of the system I will be examining. The most difficult part of calculating entropy of a gas is determining its multiplicity. But several equations have been derived using Ω that enable one to more easily calculate entropy. I will use one of those equations:

Sackur-Tetrode Equation

N = number of particles

U = internal energy of gas (J)

m = mass of gas particle (for He, kg for Ar)

k =Boltzmann’s constant ()

h = Planck’s constant ()

V = volume of gas (m3)

f = degrees of freedom (3 for Helium)

T = temperature (°K)

For U, the internal energy of the gas we utilize equipartition theorem

Let N be 100 and T be 300 °K

Which is acceptable due to the incredibly small mass of an atom and the low number of particles present in this example.

Therefore, I can calculate the initial entropy state of a section of the gas system before the partition is remove by holding most of these variables constant. The particles are identical, so mass and degrees of freedom are constant. Let the temperature and initial unmixed volume be constant as well. Thus, our entropy simply becomes a function of number of atoms chosen.

For example:

For one half of the divided container filled with Helium particles

Let

N = 100 T = 300 V = .5 \* 0.001 m3

So U is again

Thus:

However, this is only the entropy of a single section of the container. To get total entropy we can combine the entropy of both sections. The second section is identical to the first except in its mass, as it is filled with Argon. I calculated the entropy of Argon using the Sackur-Tetrode method above and thus we have

This is the total entropy of the system prior to the divider being removed. Once the divider is removed, the entropy will clearly change as the gas has more space to move and interact. Therefore, the only variable that changes once the divider is removed is the volume of the gas. We must account for the entropy of mixing which has the following equation:

But the change in entropy for both gases must be considered to get the total entropy change of the system.

Let the divider be placed directly in the middle of the container, evenly splitting the two sections, like in figure one. When this divider is removed, the volume of each gas doubles. So Vf is twice the amount of Vi.

Continuing the same example done previously with number of particles equals 100.

Thus, our total entropy after the two gases are mixed is

**Implementation in Python**

I created functions to model entropy for Helium, Argon, and entropy of mixing using the methodology above.

1. **def** Sackur\_Tetrode\_He(N1):
2. k = 1.381e-23 #Boltzmann's Constant
3. V = 0.5\*0.001 #Volume in m\*\*3
4. h = 6.626e-34 #Planck's Constant
5. m = 6.646e-27 #Mass of Helium
6. T = 300       #Temperature in Kelvins
7. U = N1\*3/2\*k\*T #Internal Energy using Equipartition Theorem
9. a = N1\*k
10. b = ((4\*pi\*m\*U)/(3\*N1\*h\*\*2))\*\*3/2
11. c = np.log(((V/N1)\*b))+5/2
12. **return** a\*c

I then calculated their values within a range of one to one thousand particles and put that into an empty list like this.

1. Particle\_Range\_He = range(1, 1000, 1)
3. **for** N1 **in** Particle\_Range\_He:
4. Values\_He.append(Sackur\_Tetrode\_He(N1))

Then the now filled lists of each respective entropy value were added to an array to get the total entropy before mixing using numpy add.

S\_Unmixed = np.add(Values\_He, Values\_Ar)

To add the change in entropy due to mixing I followed a similar method as demonstrated above. I made the entropy of mixing function, generated a range of numbers to pass through it, added those values to an empty list, then added that filled list to the S\_unmixed to get the total S\_total\_mixed array I can examine values in.

1. Values\_Mixing = []
3. **def** Entropy\_Mixing(N):
4. k = 1.381e-23
5. **return** 2\*N\*k\*np.log(2)
7. Particle\_Range\_Mixing = range(1, 1000, 1)
9. **for** N **in** Particle\_Range\_Mixing:
10. Values\_Mixing.append(Entropy\_Mixing(N))

13. S\_Total\_Mixed = np.add(S\_Unmixed, Values\_Mixing)

This S\_Total\_Mixed returns an array of the total entropy after mixing Helium and Argon. I then compiled this data into a pandas data table with columns of number of particles, entropy of gas mixed, and entropy of gas unmixed.

1. **from** collections **import** OrderedDict
2. **import** pandas as pd
4. Entropy\_Values = OrderedDict({'Number of Particles' : Particle\_Range\_Mixing,
5. 'Entropy Unmixed': S\_Unmixed,
6. 'Entropy Mixed': S\_Total\_Mixed})
8. table = pd.DataFrame(Entropy\_Values)
9. table

Then, one can see any entropy values of a certain number of particles in the table with 999 values using a pandas function that locates rows. The number entered into the function must be one less than the desired number of particles as it looks at the first pandas column, not the generated columns with values.

1. Info = table.iloc[15]
3. **print**(Info)

Which returns

Number of Particles 1.600000e+01

Entropy Unmixed 6.061080e-20

Entropy Mixed 6.091712e-20

This useful locator function allows me to compare any desired value with those I will be generating in my simulation using a Sackur-Tetrode function for only one identical gas.

**Vpython Simulation**

To simulate this process, I used the python library called vpython. Coding the movement of particles within a box is quite complex so I used code already built and functional as the base template for my code. I then added sliders for number of particles, the movement of the divider in the x-axis, and the divider moving such as to create an aperture for gases to move through. This was accomplished by looking at online examples and vpython documentation. Initially I wanted to use ipywidgets similar to project one, but these widgets performed poorly in jupyter notebook with vpython simulations. Therefore, I had to use vpython sliders and buttons which were only successful in a localhost url. Luckily, these vpython sliders were roughly analogous to ipywidget sliders so their implementation was not unfamiliar. To get the simulation to run in its own url, a document with the python code is created and added to the desktop, then through the anaconda prompt the directory is changed to desktop and this document is called with python.

**How Particles Move**

Generate random positions of particles to spawn in, classifies red or blue based on position relative to divider.

1. **for** i **in** range(self.Natoms):
2. x = L\*random()\*2-L                    # random range particles spawn in
3. y = L\*random()-L/2
4. z = L\*random()-L/2
5. **if** x < div.pos.x:
6. self.Atoms.append(sphere(pos=vector(x,y,z), radius=Ratom, color=color.blue))
7. **else**:
8. self.Atoms.append(sphere(pos=vector(x,y,z), radius=Ratom, color=color.red))

Defines an initial momentum average of particles based on equipartition theorem.

pavg = sqrt(2\*mass\*1.5\*k\*T) # kinetic energy converted to p: p\*\*2/(2mass) = (3/2)kT : average energy using equipartition theorem

Randomizes initial momentum vector components of each particle using this average

1. # Random initial momentum vector components. Phi and theta are spherical coordinates of the particle itself
2. theta = pi\*random()
3. phi = 2\*pi\*random()
4. px = pavg\*sin(theta)\*cos(phi)
5. py = pavg\*sin(theta)\*sin(phi)
6. pz = pavg\*cos(theta)
7. self.p.append(vector(px,py,pz))

Particles positions update using this random momentum vector. The vector is divided by mass to get the velocity vector which is then multiplied by dt to get the dx, dy, and dz values. Dt is set at beginning and determines frequency of particle updating its position.

1. # Update all positions
2. **for** i **in** range(self.Natoms):
3. self.Atoms[i].pos = self.apos[i] = self.apos[i] + (self.p[i]/mass)\*dt   # Move by p/m = v \* dt = dx,dy,dz

The particle continually updates its position using this code until it hits a wall or divider. This collision is detected by the particles x, y, or z position exceeding the length of the side in the x, y, or z coordinate. Whatever coordinate is detected to be exceeding, that specific momentum component is reversed. If the x-coordinate is detected to exceed the length and its momentum is negative x, its x-component of momentum is made positive, which makes it x and it bounces back at the same angle and velocity with unaffected y and z components. The same idea is applied if the x-coordinate of the particle is positive, it’s new x-component of momentum is made negative, so it bounces back.

1. **if** abs(loc.x) > L:                                   # collide left/right walls
2. **if** loc.x < 0:
3. self.p[i].x = abs(self.p[i].x)
4. **else**:
5. self.p[i].x = -abs(self.p[i].x)

This same methodology and style of calculation is repeated for the y and z position for the other walls and divider.

1. **if** abs(loc.y) > L/2:                                      # collide top/bottom walls
2. **if** loc.y < 0:
3. self.p[i].y = abs(self.p[i].y)
4. **else**:
5. self.p[i].y = -abs(self.p[i].y)
7. **if** abs(loc.z) > L/2:                     # collide front/back walls
8. **if** loc.z < 0:
9. self.p[i].z = abs(self.p[i].z)
10. **else**:
11. self.p[i].z = -abs(self.p[i].z)

Difficulties were encountered when I attempted to introduce two types of particles to mimic the differences in Helium and Argon. So, the gas in the vpython simulation is identical throughout, this changes the entropy equation and makes it dependent on number of particles present in the simulation only. The entropy calculated from the vpython simulation is a Sackur-Tetrode function for helium only inside the entire containers volume, vpython does not calculate anything.

Clearly, this will lead to differences between total entropy of just helium and those calculated with the mixing of helium and argon. Therefore, I will compare the approximation of just using helium to the accurate values from the helium and argon mixing functions.

**Results and Analysis**

Below is a table comparing the total entropy of a mixed system in the helium only function and the helium-argon mixing functions. Twenty-five values were selected for comparison to make the data easier to present and analyze. For the total entropy after mixing values I used the pandas row locator for each number of particles I wanted to find. For the helium approximation, values were taken from values returned in the command prompt after manipulating the slider. Percent difference is included as these values are representing different processes to see if entropy of one identical gas in a container approximates entropy of mixing two different gases.

|  |  |  |  |
| --- | --- | --- | --- |
| Number of Particles | Total Entropy of Mixing He and Ar (J/K) | Total Entropy from Identical He Gas Approximation (J/K) | Percent Difference |
| 5 | 1.919723e-20 | 9.40680e-21 | 51.00% |
| 68 | 2.561802e-19 | 1.25496e-19 | 51.01% |
| 107 | 4.017674e-19 | 1.96800e-19 | 51.02% |
| 149 | 5.581078e-19 | 2.73350e-19 | 51.02% |
| 179 | 6.695716e-19 | 3.27954e-19 | 51.02% |
| 234 | 8.735741e-19 | 4.27869e-19 | 51.02% |
| 278 | 1.036513e-18 | 5.07654e-19 | 51.02% |
| 316 | 1.177076e-18 | 5.76500e-19 | 51.02% |
| 368 | 1.369224e-18 | 6.70579e-19 | 51.02% |
| 394 | 1.465220e-18 | 7.17598e-19 | 51.02% |
| 428 | 1.590682e-18 | 7.79039e-19 | 51.02% |
| 485 | 1.800850e-18 | 8.81949e-19 | 51.03% |
| 525 | 1.948224e-18 | 9.54112e-19 | 51.03% |
| 579 | 2.150728e-18 | 1.05146e-18 | 51.11% |
| 618 | 2.290555e-18 | 1.12173e-18 | 51.03% |
| 643 | 2.382510e-18 | 1.16675e-18 | 51.03% |
| 713 | 2.639847e-18 | 1.29274e-18 | 51.03% |
| 760 | 2.812522e-18 | 1.37731e-18 | 51.03% |
| 798 | 2.952072e-18 | 1.44564e-18 | 51.03% |
| 836 | 3.091573e-18 | 1.51391e-18 | 51.03% |
| 868 | 3.209010e-18 | 1.57144e-18 | 51.03% |
| 911 | 3.366766e-18 | 1.64869e-18 | 51.03% |
| 952 | 3.517131e-18 | 1.72230e-18 | 51.03% |
| 995 | 3.674779e-18 | 1.79949e-18 | 51.03% |

As expected, the two values are different. This difference arises due to a lack of entropy of mixing and the small mass difference between helium and argon not being accounted for in the identical helium gas approximation. The percent difference is virtually a uniform value throughout, around fifty-one percent. This consistency in percent difference is due to the equation for entropy of mixing being the same and scaled to the number of particles. This consistent value will lead to a consistent difference in entropy, likely the fifty-one percent difference, with the remaining tenths of percentage being due to the helium argon mass difference. We can conclude that approximating the entropy of helium and argon mixing using an identical container with only helium is only useful if you account for the entropy of mixing difference. If different gases were used, the mass difference would need to be accounted for if the difference was significant as well.

**References**

1. *An Introduction to Thermal Physics* (Thermodynamics textbook)

2. <http://physics.bu.edu/~duffy/py105/Idealgas.html>

3. <https://theory.physics.manchester.ac.uk/~judith/stat_therm/node55.html>

4. <http://physics.bu.edu/~duffy/py105/Kinetictheory.html>

5. <http://www.planetb.ca/syntax-highlight-word>

6. <https://www.glowscript.org/#/user/GlowScriptDemos/folder/Examples/program/HardSphereGas-VPython>