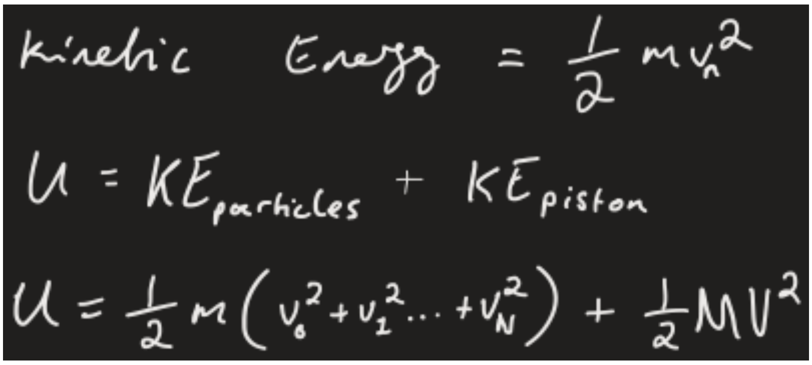
Thermo CA 2

Question 1

a)

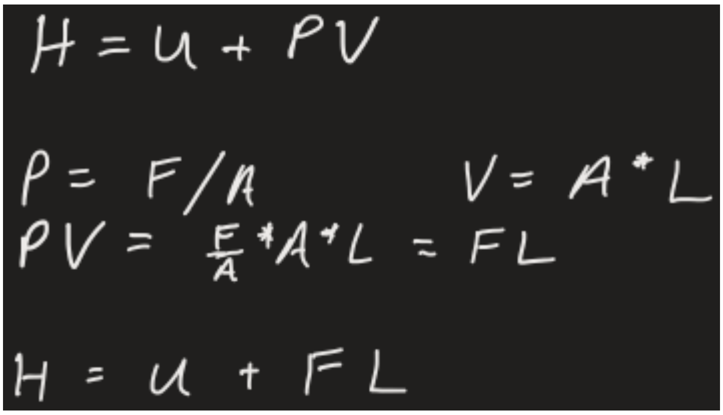


$$\text{kinetic Energy} = \frac{1}{2} m v_n^2$$

$$U = KE_{particles} + KE_{piston}$$

$$U = \frac{1}{2} m \left(v_0^2 + v_1^2 \ldots + v_N^2\right) + \frac{1}{2} M V^2$$

b) The equation of enthalpy (H) for an ideal gas:



$$H = U + PV$$

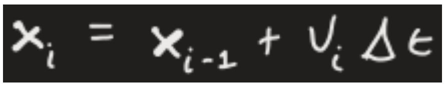
$$P = F/A \qquad V = A * L$$

$$PV = \frac{F}{A} * A * L = FL$$

$$H = U + FL$$

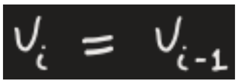
Where L is the X position of the piston.

c)



$$x_i = x_{i-1} + v_i \Delta t$$

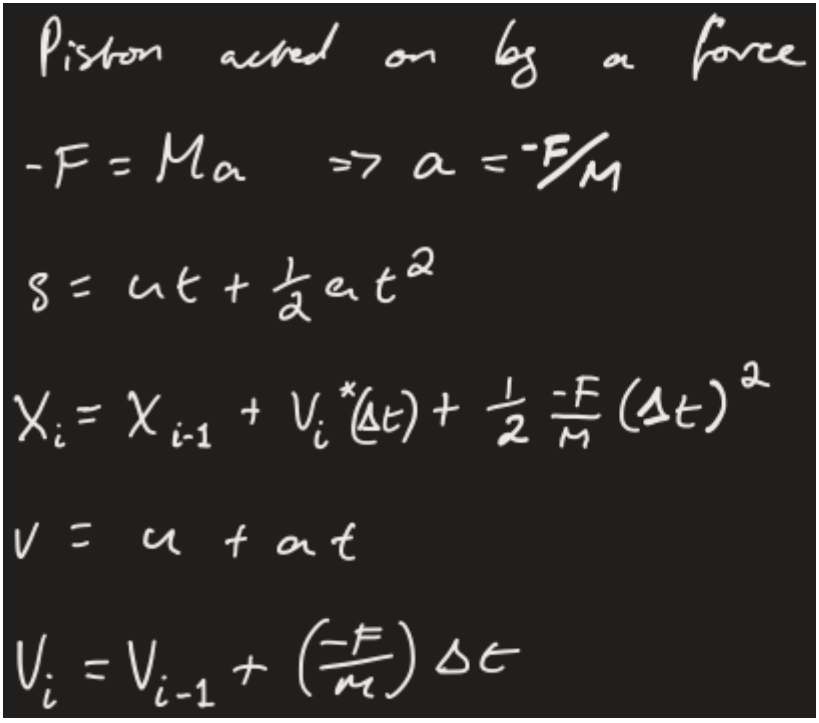
At every incremental step, the particle position is updated by the previous position + the change in position for that incremental timestep.



$$v_i = v_{i-1}$$

The velocity of the particle does not change unless the particle is involved in a collision with the piston, where the velocity of the particle is governed by the equation for ω.

d)



Piston acted on by a force

$$-F = Ma \Rightarrow a = {}^{-F}/_{M}$$

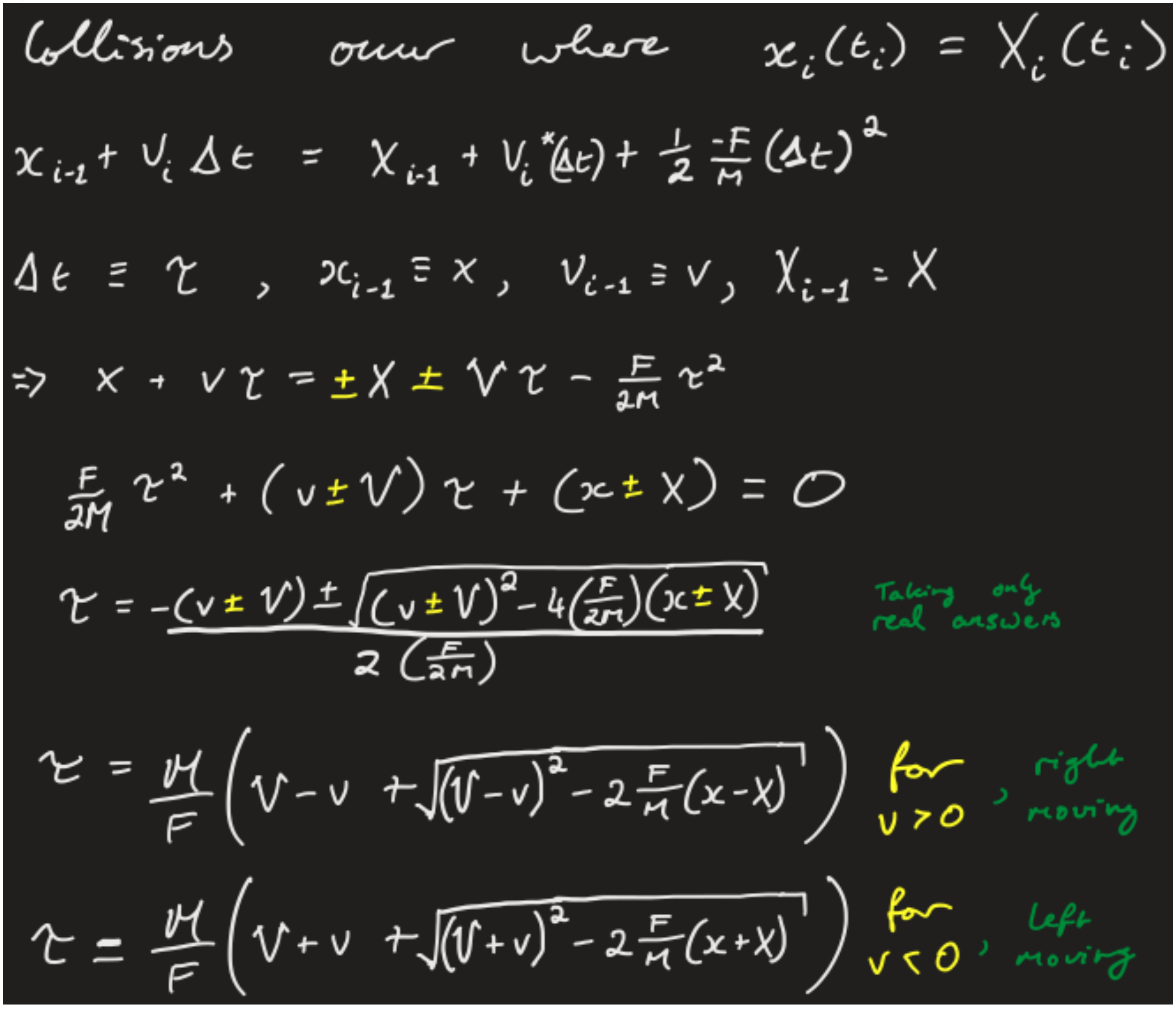
$$s = ut + \tfrac{1}{2}at^2$$

$$X_i = X_{i-1} + V_i^*(\Delta t) + \tfrac{1}{2}\tfrac{-F}{M}(\Delta t)^2$$

$$v = u + at$$

$$V_i = V_{i-1} + \left(\tfrac{-F}{M}\right)\Delta t$$

e)



Collisions occur where $x_i(t_i) = X_i(t_i)$

$$x_{i-1} + v_i\,\Delta t = X_{i-1} + V_i^*(\Delta t) + \tfrac{1}{2}\tfrac{-F}{M}(\Delta t)^2$$

$$\Delta t \equiv \tau,\quad x_{i-1} \equiv x,\quad v_{i-1} \equiv v,\quad X_{i-1} = X$$

$$\Rightarrow x + v\tau = \pm X \pm V\tau - \tfrac{F}{2M}\tau^2$$

$$\tfrac{F}{2M}\tau^2 + (v \pm V)\tau + (x \pm X) = 0$$

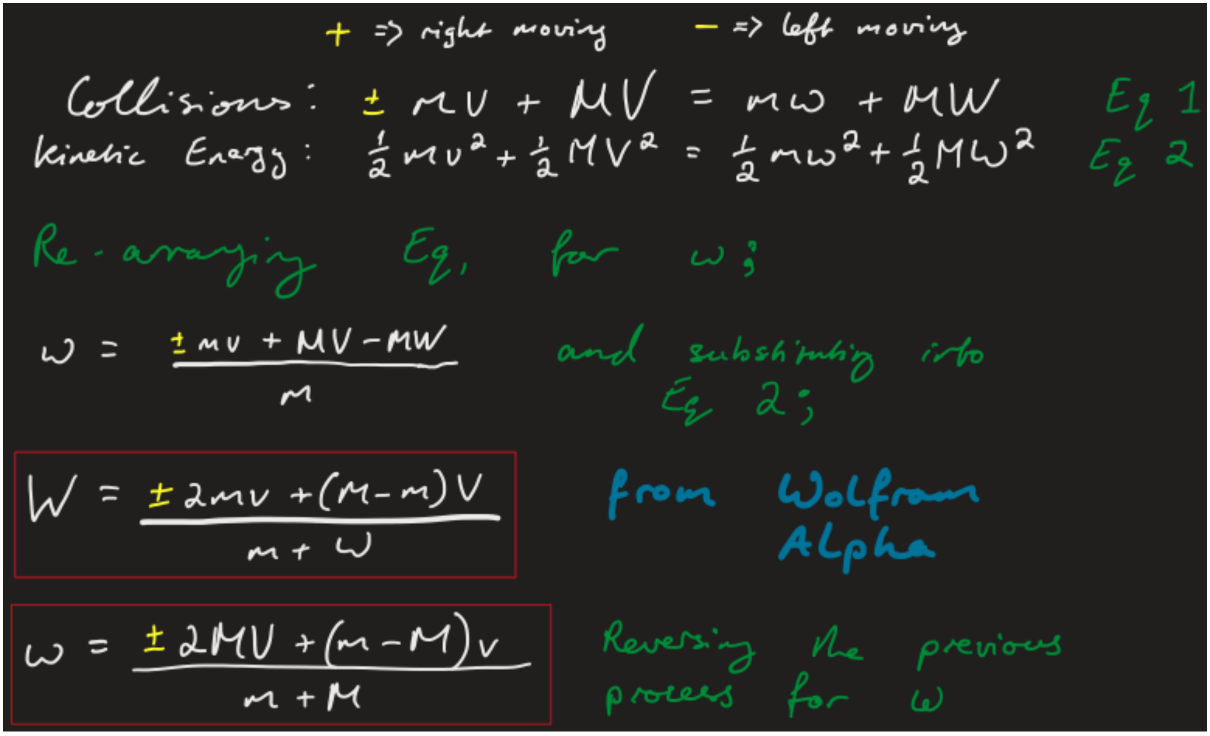
$$\tau = \frac{-(v \pm V) \pm \sqrt{(v \pm V)^2 - 4\left(\frac{F}{2M}\right)(x \pm X)}}{2\left(\frac{F}{2M}\right)}$$

Taking only real answers

$$\tau = \frac{M}{F}\left(V - v + \sqrt{(V - v)^2 - 2\tfrac{F}{M}(x - X)}\right)$$ for $v > 0$, right moving

$$\tau = \frac{M}{F}\left(V + v + \sqrt{(V + v)^2 - 2\tfrac{F}{M}(x + X)}\right)$$ for $v < 0$, left moving

f)



+ => right moving   − => left moving

Collisions: $\pm mv + MV = mw + MW$ Eq 1

Kinetic Energy: $\frac{1}{2}mv^2 + \frac{1}{2}MV^2 = \frac{1}{2}mw^2 + \frac{1}{2}MW^2$ Eq 2

Re-arranging Eq, for w;

$$w = \frac{\pm mv + MV - MW}{m}$$

and substituting into Eq 2;

$$\boxed{W = \frac{\pm 2mv + (M-m)V}{m + W}}$$

From Wolfram Alpha

$$\boxed{w = \frac{\pm 2MV + (m-M)v}{m + M}}$$

Reversing the previous process for w

Question 2

a)

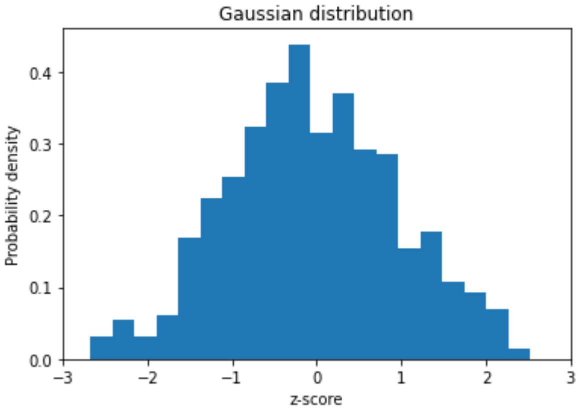
```
z = np.array([(0,0)]*1000)
```

b)

```
def func(array,a,b):
    return (sum(array)**a),(sum(array)**b)
rand_array = np.random.rand(10)
x, y = func(rand_array,2,3)
```

c)

```
gauss_array = np.random.normal(size = 500)
```

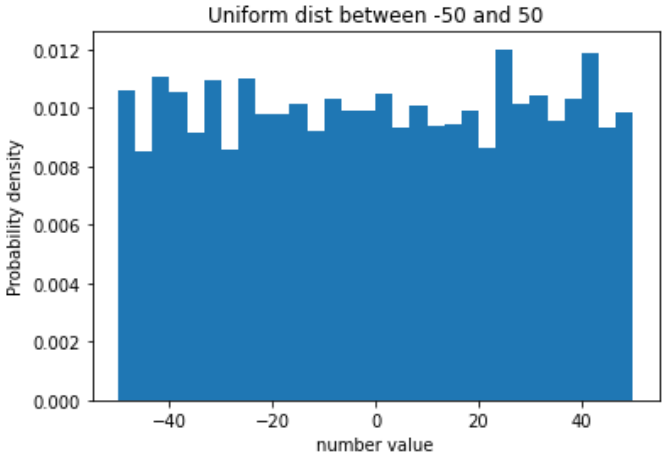


d)

```python
for i in range(3):
    choice_array = [10 if x==1 else -10 for x in np.random.choice(2,10)]
```

e)

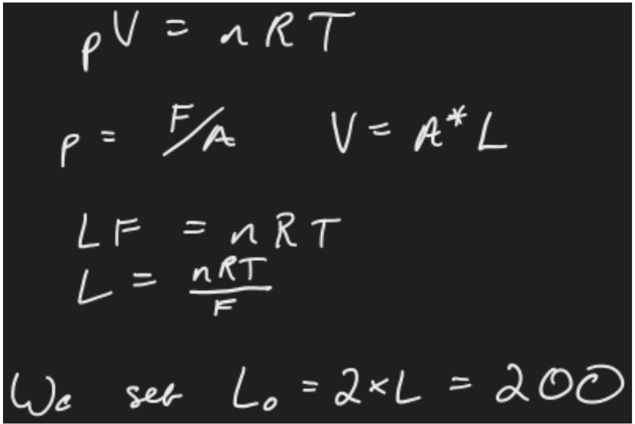
```python
z2 = []
for i in range(50):
    for j in range(100):
        z2.append(np.random.uniform(-50,50))
```



Question 3

a)

The final equilibrium was calculated:

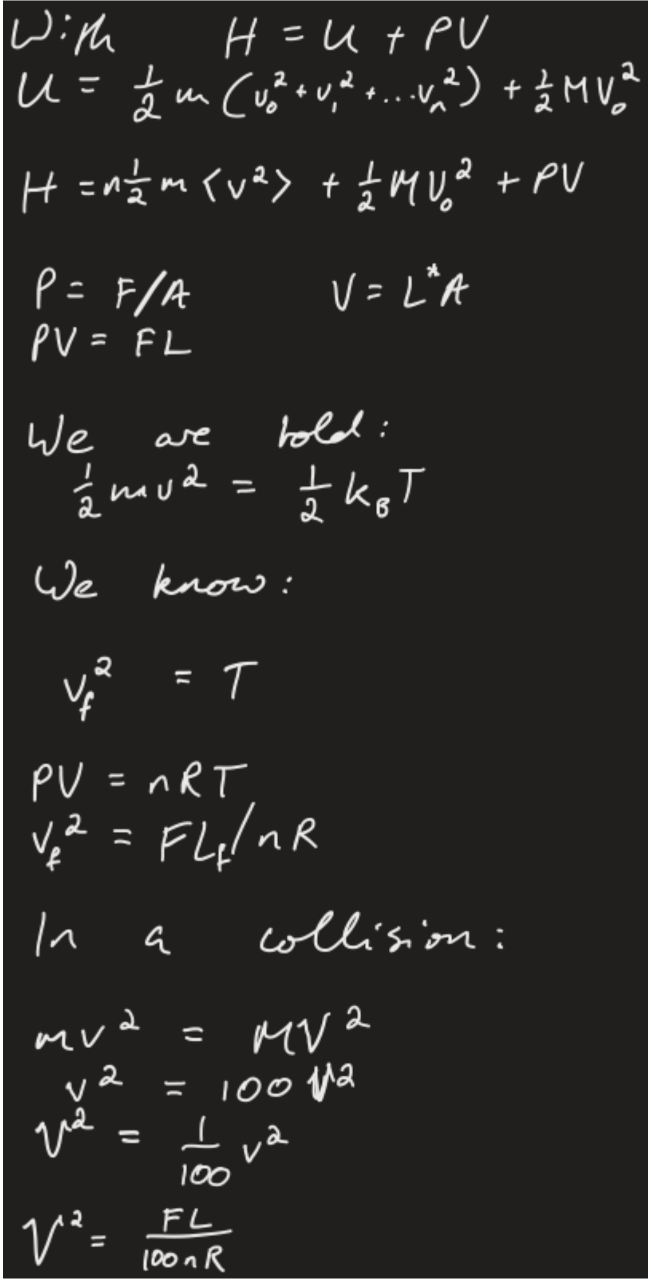


$$pV = nRT$$

$$p = F/A \qquad V = A*L$$

$$LF = nRT$$
$$L = \frac{nRT}{F}$$

We set $L_o = 2 \times L = 200$



With $\quad H = U + PV$

$$U = \frac{1}{2} m (v_0^2 + v_1^2 + \ldots v_n^2) + \frac{1}{2} M V_0^2$$

$$H = n \frac{1}{2} m \langle v^2 \rangle + \frac{1}{2} M V_0^2 + PV$$

$$P = F/A \qquad V = L*A$$
$$PV = FL$$

We are told:

$$\frac{1}{2} m v^2 = \frac{1}{2} k_B T$$

We know:

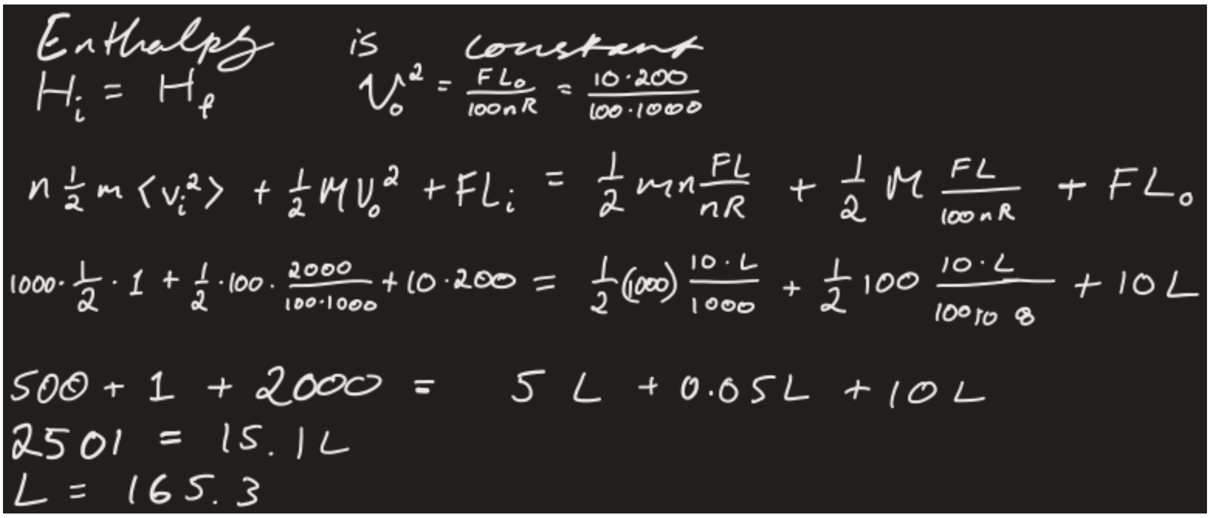
$$v_f^2 = T$$

$$PV = nRT$$
$$v_f^2 = FL_f / nR$$

In a collision:

$$mv^2 = MV^2$$
$$v^2 = 100 V^2$$
$$V^2 = \frac{1}{100} v^2$$

$$V^2 = \frac{FL}{100 nR}$$



Enthalpy is constant

$H_i = H_f$ $\qquad V_0^2 = \frac{FL_0}{100nR} = \frac{10 \cdot 200}{100 \cdot 1000}$

$$n\frac{1}{2}m\langle v_i^2\rangle + \frac{1}{2}MV_0^2 + FL_i = \frac{1}{2}mn\frac{FL}{nR} + \frac{1}{2}M\frac{FL}{100nR} + FL_0$$

$$1000 \cdot \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 100 \cdot \frac{2000}{100 \cdot 1000} + 10 \cdot 200 = \frac{1}{2}(1000)\frac{10 \cdot L}{1000} + \frac{1}{2}100\frac{10 \cdot L}{100 \; 10 \; 8} + 10L$$

$$500 + 1 + 2000 = 5L + 0.05L + 10L$$

$$2501 = 15.1L$$

$$L = 165.3$$

Which gives the equilibrium position that the simulation tends towards.

A program was set up to calculate and plot a time series of the piston motion performing a damped oscillation. This was done with unitless values.
This was coded in Jupyter notebook rather than linearly programming, since calling the results of a cell after simulation is more efficient than waiting for a simulation to run with every change.

Functions were set up to calculate the time until next collision and velocities after collision, as outlined in the manual. Further functions were defined to express the equations of motion to update the position of the particles and the piston, as well as the velocity of the piston. The velocity equation is shown:

$$V_i = V_{i-1} + \frac{-F}{M}\Delta t$$

The programmed functions are shown:

```python
def x_motion_particle(previousx,v,delta_t):
    currentx = previousx + v*delta_t
    return currentx

def X_motion_piston(previousX,V,delta_t,F,M):
    currentX = previousX + V*delta_t + 0.5*(-F/M)*(delta_t**2)
    return currentX

def V_motion_piston(previousV,delta_t,F,M):
    return previousV + (-F/M)*delta_t
```

For every time step of the simulation, the minimum waiting time (delta_t) for the next collision to occur is calculated using this function:

```python
def min_waiting_time(time0,N,M,F,V_0,v_vel,X_0,x_pos):
    index = 0
    for i in range(0,N):
        time1 = waiting_time(M,F,V_0,v_vel[i],X_0,x_pos[i])
        if time1 <= time0:
            time0 = time1
            index = i
    return time0, index
```

This is significantly better than methods that involve lists and the min() function, since list appending and searching are computationally expensive.

The main loop to simulate the system is shown:

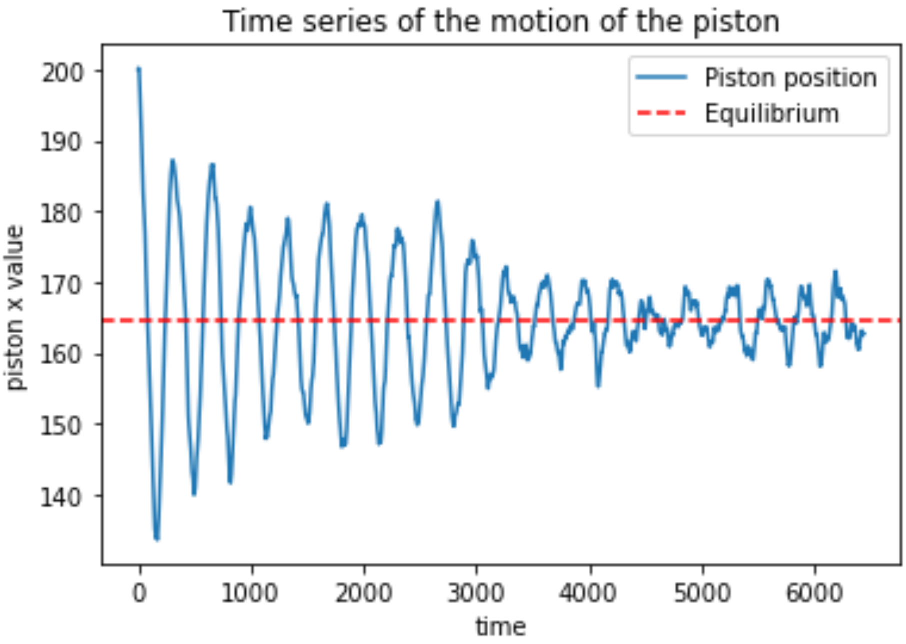
```
for i in time_steps:
    delta_t, index1 = min_waiting_time(1e10,N,M,F,V_0,v_vel,X_0,x_pos)

    for i in range(N):
        x_pos[i] = x_motion_particle(x_pos[i],v_vel[i],(delta_t))

    X_0 = abs(x_pos[index1])
    V_0 = V_motion_piston(V_0,delta_t,F,M)
    v_vel[index1] , V_0 = Wwvelocity(M,F,V_0,m,v_vel[index1],delta_t)
    t = t + (delta_t)

    v_list.append(v_vel[:])
    t_list.append(t)
    delta_t_list.append(delta_t)
    X_list.append(X_0)
    V_list.append(V_0)
```

The positions of every particle (x_pos) are updated using the equations of motion, using the minimum waiting time. The position of the piston (X_0) is set to the position of the colliding particle, and the velocity of the piston (V_0) is then updated by the equation of motion for the piston. The velocity of the colliding particle (v_vel[index]) and velocity of the piston are then updated by the formulas for w and W, using the previous velocities. Time is incremented by the waiting time, and all relevant variables are added to lists to be called later.



The plot of the results shows a damped oscillatory motion, which is expected. The piston position hovers around the equilibrium position. There is an element of randomness to the oscillations, given them their ragged shape. This effect is more noticeable as the piston position tends towards equilibrium and the piston encounters more collisions.

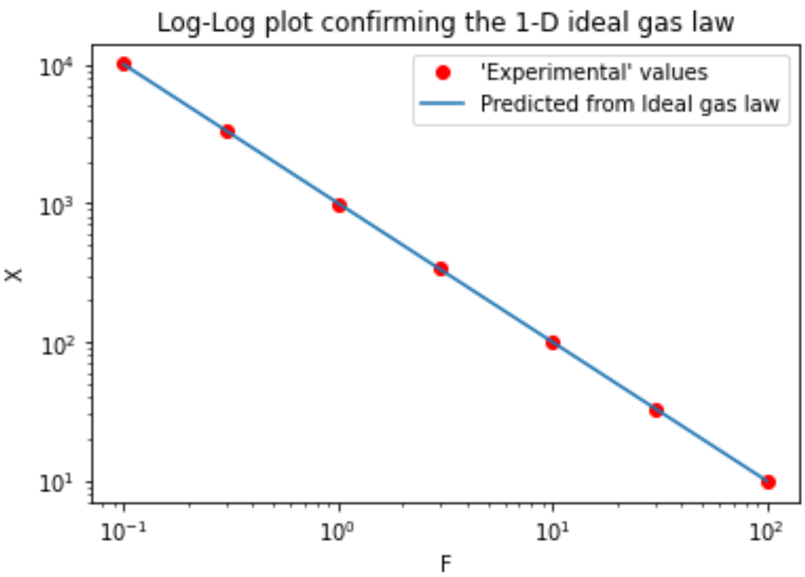
b)
In this part, we show that the system agrees with the equation of state of the 1D ideal gas by running the simulation for different Force amounts and taking the weighted average over the last few oscillations as the experimental equilibrium value. The weighted average takes the waiting times (Δt) as the weights.

The weighted average was found by defining a function:

```
def weighted(X_list, delta_t_list,selection):
    return np.average(X_list[-selection:], weights = delta_t_list[-selection:])
```

Which ran for a selection of the data provided, such that only the last few oscillations are factored in. The amount chosen for this was 5000 data points, thought this could be made more comprehensive by using a peak finding function over the data. Ultimately the same result is achieved.

The simulation step is similar to above, but run in a for loop with different values of F. The aim ultimately is to show a relationship between the force applied and the experimental equilibrium X position. Therefore, the weighted average of the X positions is plotted against the F values on a log-log plot:



c)
In this part, instead of defining the particle velocities as random values in a gaussian distribution, values are defined as either plus or minus $\sqrt{\frac{k_bT}{m}}$. The simulation is then run, and as collisions occur and the simulation evolves in time, the velocities begin to tend towards a Maxwell distribution.
A function is written to define the particle velocities in the binary distribution:

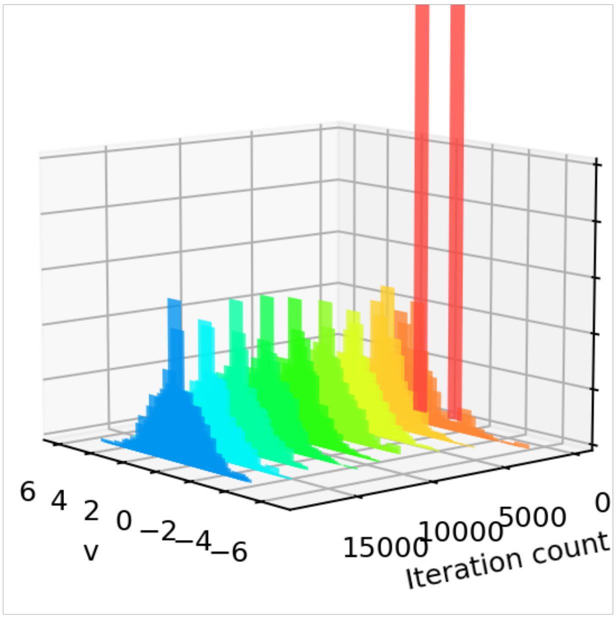
```
def choice_v(N,k_B,T,m):
    bi = np.sqrt((k_B*T_0)/m)
    choice_array = [bi if x==1 else -bi for x in np.random.choice(2,size = N)]
    return choice_array
```

The previous simulation loop is modified to record the list of velocities in every iteration of the loop, so that the histogram can be plotted later. This is achieved;

```
v_list.append(v_vel[:])
```

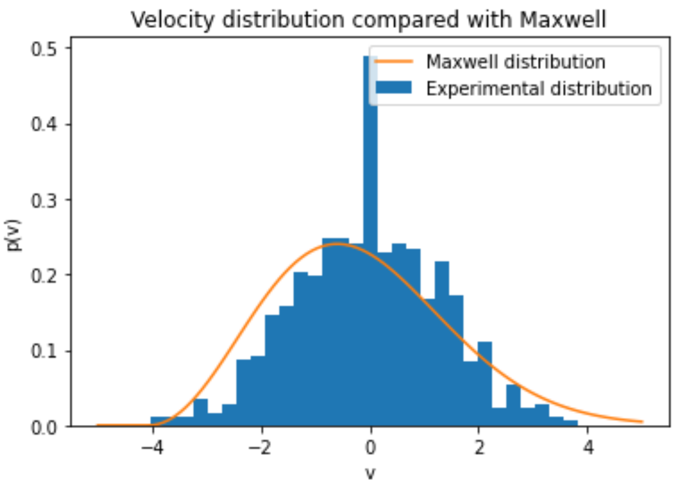
Now there exists a mega list *v_list* containing the distribution of velocities for every iteration. These are plotted on a 3d plot:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(dpi = 180)
ax = fig.add_subplot(111, projection='3d')
for z in time_steps[::spacing]:
    ys = v_list[z]
    hist, bins = np.histogram(ys, bins=nbins, density = True)
    xs = (bins[:-1] + bins[1:])/2
    ax.bar(xs, hist, zs=z, zdir='y', alpha=0.8, color = colour_list[int(z/spacing)].hex)
```



Which shows the distributing tending towards a maxwell distribution. This is fitted on a 2-D plot:

```
params = maxwell.fit(v_list[-1], scale = 1)
x = np.linspace(-5,5,100)
plt.plot(x, maxwell.pdf(x, *params))
plt.hist(v_list[-1], bins = 30,density = True)
```



It should be noted that the maxwell fit module from scipy automatically corrects for the distribution not starting at 0.

The standard deviation and mean were calculated:

```
maxwell_list = list(maxwell.rvs(*params,size = 100000))
print("Mean: " + str(sum(maxwell_list)/len(maxwell_list)))
print("Standard deviation:" +str(np.std(maxwell_list)))
```

```
>> Mean: -0.15930381464096285
>> Standard deviation:1.6503800374730984
```

The system thermalises without interactions between particles since there are still collisions between particles and the piston, admittedly resulting in a slower thermalisation.

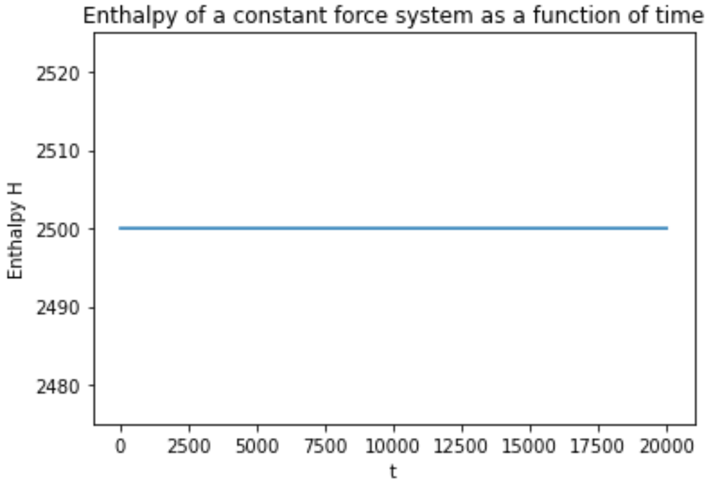
d)

i)

The enthalpy H should be constant for this system. The equation for enthalpy derived in Q1b is used, with the equation for internal energy U from 1a. H is calculated for every iteration in the simulation in the following loop;

```
H_list = []
for i in range(len(time_steps)):
    v_squared_list = []
    for j in (v_list[i]):
        v_squared_list.append(j**2)
    U = 0.5*m*(sum(v_squared_list)) + 0.5*M*(V_list[i]**2)
    FL = F*(X_list[i])
    H_list.append(U+FL)
```

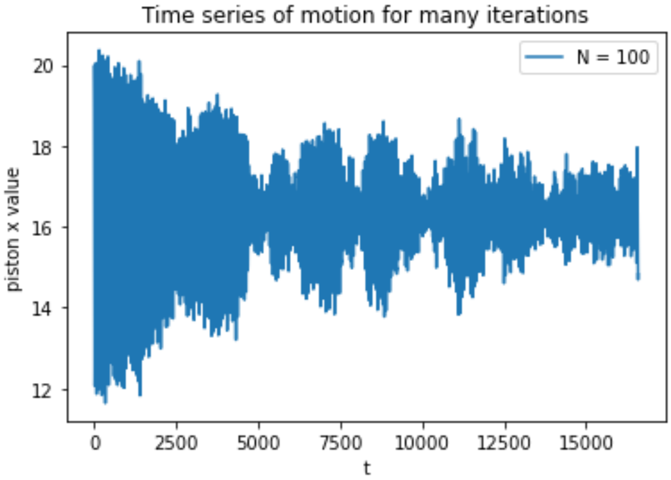
And plotted;

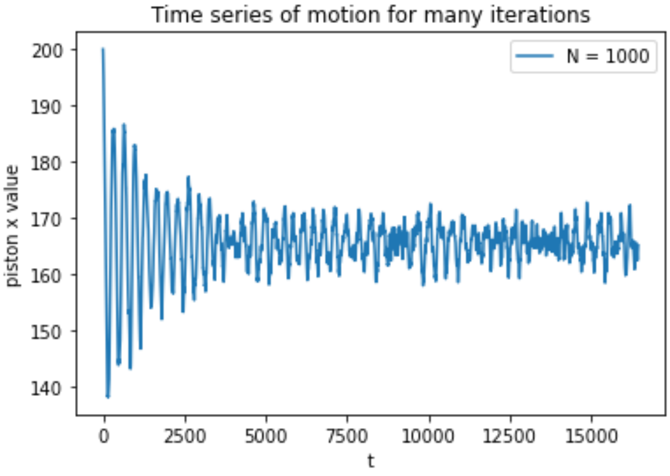


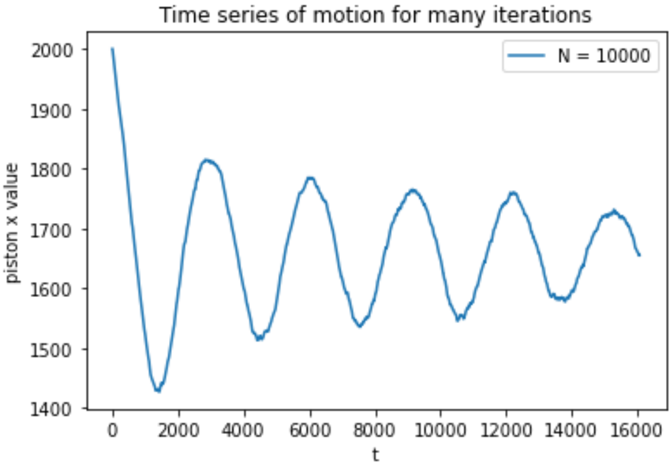
Showing the enthalpy H is conserved.

iii)

Running the system for longer leads to the piston position settling down asymptotically, with small perturbations since the particles still have velocities. This was run for N = 100, N = 1000, and N = 10000







Which shows when the N is very large, the random perturbations are smoothed out.