Statistical Thermodynamics CA2

Q1) a In terms of the microstate variables given, we define the internal energy U of the system as.

$$U = \frac{1}{2}m\sum_{i=1}^{N}(V_i^0)^2 + \frac{1}{2}M(V^0)^2$$

U is given by the sum of all the kinetic energy of the particles and the piston. We know there is no force between the particles and the piston.

1b) We know that enthalphy $= U+PV$. But we are told this is a 1-dimensional system so enthalpy $= Fx+U$ $\left(p = \frac{F}{A}\right)$

$$H(t_0) = \frac{1}{2}m\sum_{i=1}^{N}(V_i^0)^2 + \frac{1}{2}M(V^0)^2 + Fx^0$$

1c) $V_i(t) = V_i^0$, as particles don't experience force.

From equations of motion $x_i(t) = x_i^0 + v_i^0(t-t_0)$

$v = u+at$ and $s = ut + \frac{1}{2}at^2$

1d) for the piston $a = \frac{-F}{M}$, so $V(t) = V^0 = \frac{-F}{M}(t-t_0)$

From equations of motion, $X(t) = X^0 + V^0(t-t_0) - \frac{1}{2}\frac{F}{M}(t-t_0)^2$

1e) finding times when particle moves towards piston and moves towards wall and collides.

$t = \tau$, when $t_0 = 0$

$X(\tau) = x(\tau)$, so $x + v\tau = X + V\tau - \frac{F}{2M}\tau^2$

$(\tau^2)(F/2M) + \tau(v - V) + (x - X) = 0$

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

$\tau = 0$ when $X = x$

$V - v\sqrt{(V-v)^2 - \frac{2F}{M}(X - x)} < 0$, for particles moving to right

When particles moving to left colliding with wall but in our simplification the left piston.

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

$(\tau^2)(\frac{F}{2M}) + \tau(-V - v) + (-x - X) = 0$, like before

so $\tau = \frac{M}{F}\left(V + v \pm \sqrt{(V+v)^2 + \frac{2F}{M}(X + x)}\right)$, when $v < 0$

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

the two waiting times we have found are 1) for when the particle moves towards the right hand side and collides with the piston and the second when the particle moves towards the left hand side, hits the wall and comes back to collide with the piston.

f) For the particle moving towards the right we have the conservation of momentum equation gives us.

$$M\vec{V} + m\vec{v} = M\vec{W} + m\vec{w}$$

the conservation of energy equation gives us

$$MV^2 + mv^2 = MW^2 + mw^2.$$

the particle moves to the right collides with the piston and then moves left while the piston doesn't change direction.

Solving these two equations for ω and W we get $\vec{\omega} = \dfrac{2MV + (m-M)V}{m+M}$ and $\vec{W} = \dfrac{2mv + (M-m)V}{m+M}$

(for $v>0$)

Now for a particle moving to the left we have two new equations like above given by

$$m\vec{v} - M\vec{V} = m\vec{\omega} - m\vec{V} \quad \text{(conservation of momentum)}$$

and $$mv^2 + MV^2 = m\omega^2 + MV^2 \quad \text{(conservation of energy)}$$

the particle moves towards the left and then collides with the piston causing it to then move towards the right.

Solving the two conservation equations we get

$$\vec{\omega} = \frac{-2MV + (m-M)v}{m+M} \quad \text{and} \quad \vec{W} = \frac{-2mv + (M-m)V}{m+M}$$

(for $v<0$)

The two equations for the velocity immediately after the collision for the piston W and particle ω are

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

and $$\vec{\omega} = \frac{\pm 2MV + (m-M)v}{m+M}$$

Q2) The following images are screen grabs of the code and output for Q2.

a)

```python

import numpy as np
import matplotlib.pyplot as plt

z = np.zeros((1000,2))
print(z)

```

```
In [1]: runcell(0, 'C:/Users/kadams/Downloads/untitled0.py')
[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]

In [2]:
```

b)

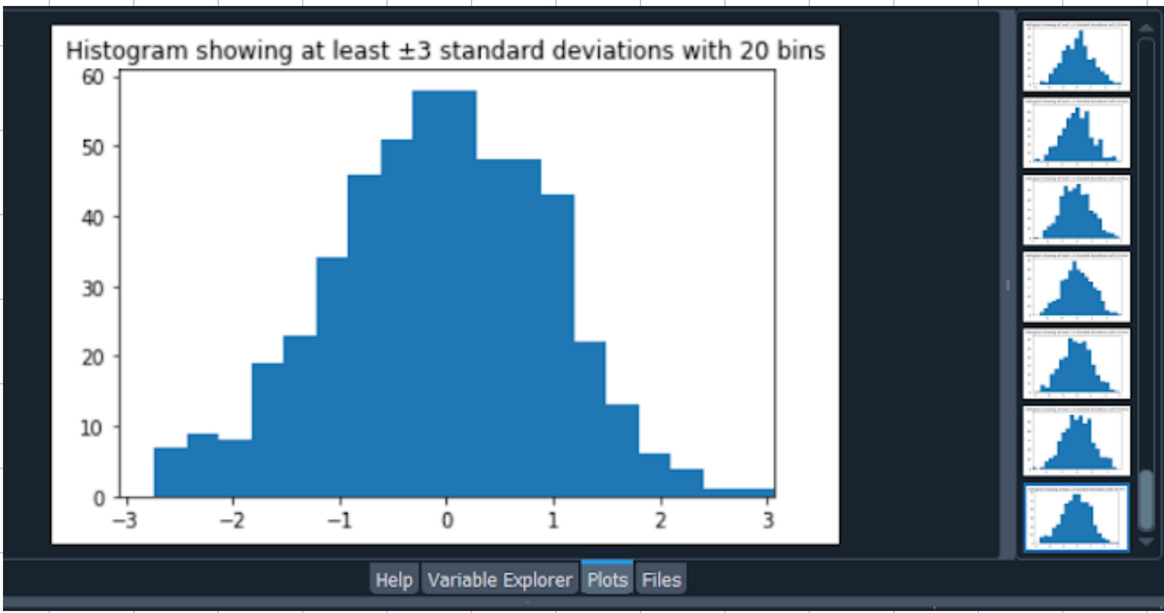
```python

def function(z,a,b):
    m = sum(z)**a
    n = sum(z)**b
    return[m,n]
set = np.arange(0,1,0.001)
zn = np.random.random(10)
x,y = function(zn,2,3)
print('calling the function with a 10 element array of random numbers between 0 and 1')

```

c)

```python
OurGaussianDist = np.random.normal(size = 500)
sigma = st.stdev(OurGaussianDist)
plt.hist(OurGaussianDist,20)
plt.xlim(sigma*-3,sigma*3)
plt.title('Histogram showing at least ±3 standard deviations with 20 bins')
plt.show()
```



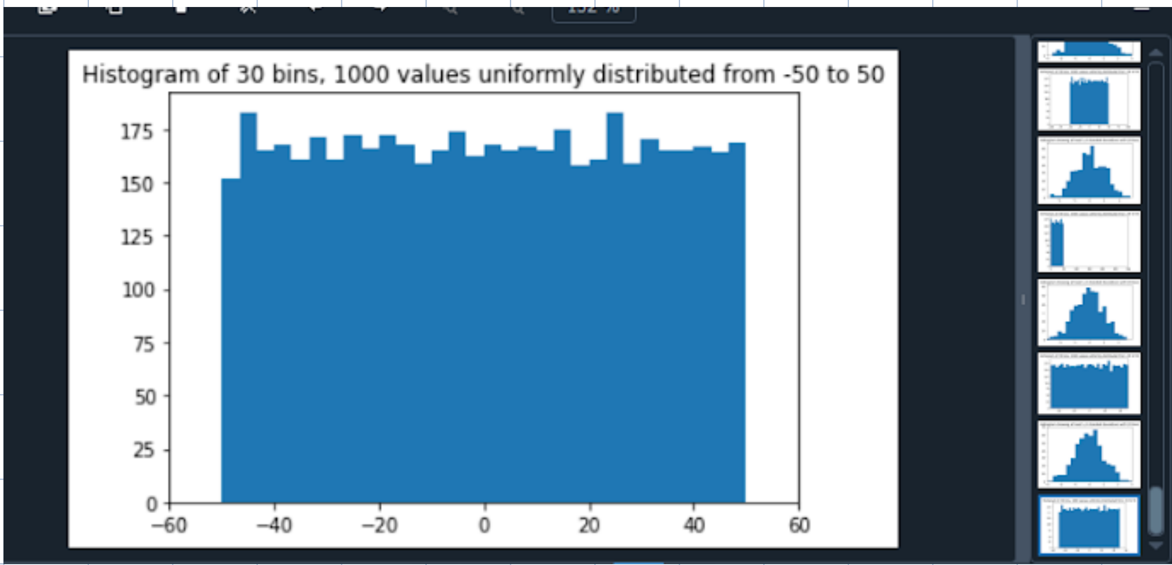
d)

```python
r = [-10,10]
first = np.random.choice(r,20)
second = np.random.choice(r,20)
third = np.random.choice(r,20)
print(first)
print(second)
print(third)
print('we have created 3 arrays of 20 elements chosen randomly to have the value +10 or -10')
```

```
[ 10  10  10 -10 -10 -10  10  10  10  10 -10  10  10 -10  10  10  10  10
  10 -10]
[ 10  10  10  10 -10  10  10  10 -10 -10 -10 -10  10  10 -10  10  10  10
  10  10]
[ 10 -10  10 -10  10  10 -10  10 -10 -10  10 -10 -10 -10 -10 -10 -10 -10
 -10 -10]
we have created 3 arrays of 20 elements chosen randomly to have the value +10 or -10
```

e)

```python
e = np.linspace(-50,50,51)
zempty = []
for a in range (50):
    b = 0
    while b < 100:
        r = np.linspace(e[a],e[a+1],100)
        z = np.random.choice(r)
        b += 1
        zempty.append(z)
plt.hist(zempty,bins = 30)
plt.title('Histogram of 30 bins, 1000 values uniformly distributed from -50 to 50')
plt.xlim(-60,60)
plt.show()
```



Q3 (a)

For this exercise we plot a time series of the motion of the piston as it performs a damped oscillation towards equilibrium. The equipartition of energy theorem is used to set a reasonable scale for the initial velocities of the piston and particles i.e. an equal amount of energy is allocated to each constituent N. The system is initialized with a starting temperature T_0 and a force F. The initial piston and particle velocities are given from a random Gaussian distribution. We expect the piston to converge to an equilibrium value of 166.7.

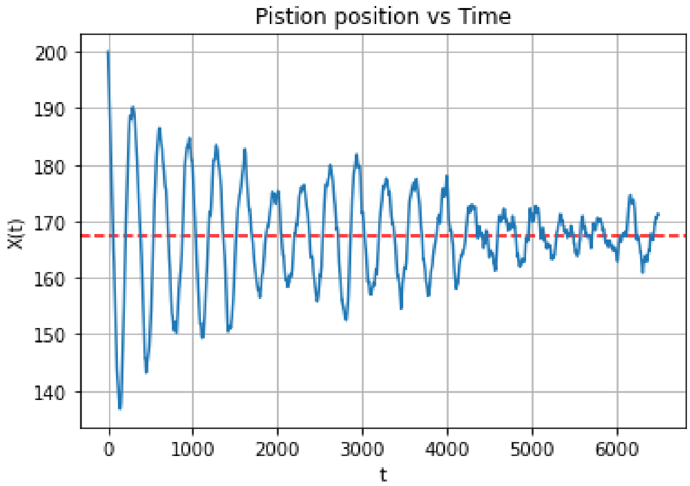


Figure 1 above shows the position of the piston X(t) against time t, showing the dampening over time. The dotted red line shows the convergence of the piston to an expected equilibrium position of around 166.7.

Q3 (b)

In this question we want to show that our system agrees with the equation of state for the one dimensional ideal gas. We show this by showing how the starting position of the piston varies with the force i.e. calculating the average piston position X over several final oscillations. We plot the average piston position over 7 values of force of 0.1, 0.3, 1, 3, 10, 30 and 100.

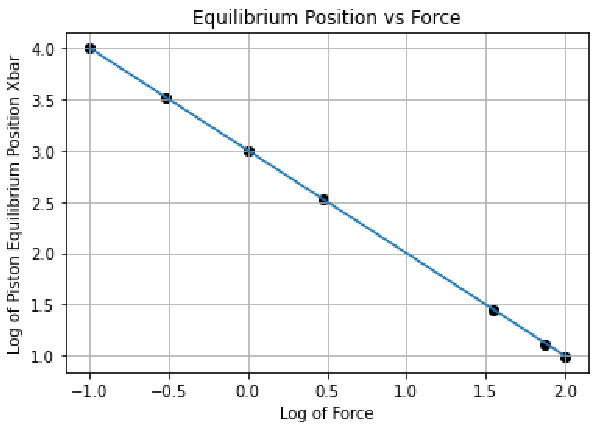


Figure 2 above shows a log-log plot of the Equilibrium position against a range of forces

Q3 (c)

Here we initialize the initial velocities to be a random value of +Vo or –Vo. We can see that this distribution tends towards a Maxwell distribution. We then fit a Maxwell distribution curve on a separate plot. The system thermalizes because of the collisions and interactions between the particles and the piston and not because of the interactions between the particles.

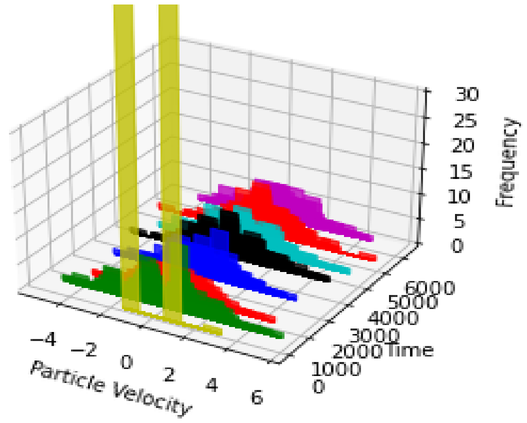


Figure 3 above shows a 3-D histogram of the initial bi-polar distribution of the particles' velocities.

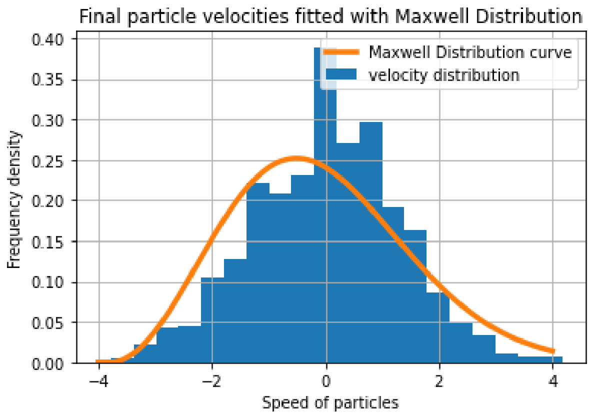


Figure 4 above shows a Maxwell distribution curve fitted to the particle velocity distribution of the system with a standard deviation of 2.33

Code

(a)

```
import math

import numpy as np

import matplotlib.pyplot as plt

import random



#this is a cleaned, ordered and easy to follow version of the code


#lets create all variables

#remember capital letters are for piston and lowercase for particle

M = 100.0

m = 1.0

T_o = 1.0

F = 10.0

R = 1.0

N = 1000.0

k = 1.0

X_o = (2*N*R*T_o)/(F)

#The above variables are all found from the question and equations we will use


#lets create our lists

T_list = []

PistonPos_list = []

Vel_list = []

x_o = []


# x_o is the particles start position and X_o is the pistons'

#set up while loop to increment from 0 to 1000
```

```
count1 = 0

while count1 < 1000:

    x_o.append(random.uniform(-X_o,X_o))

    count1 = count1 + 1


# distributing particles questions between X_o and X_o as the question asks

#rembeber lowercase 'x' refers to the particle

# uppercase refers to the piston


rms = np.sqrt((k*T_o)/(m))

mu, sigma = 0, rms


v_o = []

count2 = 0

while count2 < 1000:

    v_o.append(random.gauss(mu,sigma))

    count2 = count2+1


V_o = random.gauss(mu, sigma)

if V_o > 0:

    V_o = -1*V_o




#we just set the particle and piston velocities for random gaussian values


x_o = np.array(x_o)

v_o = np.array(v_o)

#creating variables into arrays

#creating an overall time measuring variable and setting to 0 like question asks

time_measure = 0
```

```
# To hold the current microstate variables of the particles we will
c_microstates = np.column_stack((x_o, v_o))
#this function stacks 1-d arrays as columns in a 2-d array


#lets define our waiting time equation we showed in q1
# we will break down the equation and create different notations for them
# this is so we dont mess up when typing the equation in
# use fi as 1st variables we can see to group together
# and se as 2nd etc etc
def waiting_time(x,v,X,V,F,M):
   a = M/F
   b = V-v
   c = (V-v)**2
   d = x-X


   e = V+v
   f = (V+v)**2
   g = x+X


   if v>0:
      return a*(b + np.sqrt((b**2 -2*(F/M)*d)))


   if v<0:
      return a*(e + np.sqrt(e**2 + 2*(F/M)*g))



#now we need to define the equations for piston and particle immediately after collision



def velocityimm_after (m,M,v,V):
```

```
    if v > 0:
        W = ((2*m*v) + (M-m)*V)/(m+M)
        w = ((2*M*V) + (m-M)*v)/(m+M)

    if v < 0:
        W = ((-2*m*v) + (M-m)*V)/(m+M)
        w = ((-2*M*V) + (m-M)*v)/(m+M)

    return W,w

#now we have defined all the equations we will need
# now need to define to the system what will happen after

T_list.append(time_measure)
PistonPos_list.append(X_o)
Vel_list.append(V_o)
collisions = 0
maximum = N*20
# creating another while loop with above
while collisions < maximum:

    tau = []
    for x,v in c_microstates:
        tau.append(waiting_time(x,v,X_o,V_o,F,M))

    quickest_time = np.min(tau)
    index = tau.index(quickest_time)
 #as explained in step 3 we should displace each particle according to the equations of motion
    displaced = []
```

```
    for x,v in c_microstates:
        x_after = x + (quickest_time*v)
        displaced.append(x_after)

    x_after = np.abs(displaced[index])
    c_microstates = np.column_stack((displaced,v_o))



#define the new pistion velocity
# letter n will be used to signify new with same notation as previous
# no more o in subscript as not initial anymore
    nV = V_o+(-1*F/M)*quickest_time
    nparticleV = velocityimm_after(m,M,c_microstates[index,1], nV)[1]
    npistonV = velocityimm_after(m,M,c_microstates[index,1], nV)[0]
    time_measure += quickest_time
# adding here
#now update lists made at the start
    v_o_list = v_o.tolist()
    v_o_list = v_o_list[:index] + [nparticleV] + v_o_list[index+1:]
    v_o = np.array(v_o_list)
    c_microstates = np.column_stack((displaced, v_o))

    T_list.append(time_measure)
    PistonPos_list.append(x_after)
    Vel_list.append(npistonV)
    collisions+=1
    X_o = x_after
    V_o = npistonV

plt.plot(T_list,PistonPos_list)
```

```
Expected_Equilibrium_position = np.sum(PistonPos_list)/len(PistonPos_list)
plt.subplots()
plt.axhline(Expected_Equilibrium_position, linestyle='--', color = 'black')
plt.legend(loc="upper right")
plt.plot(T_list,PistonPos_list)
plt.title('Pistion position vs Time')
plt.xlabel('Time t')
plt.ylabel('Piston Position X(t)')
plt.grid(True)
plt.show()
```

(b)

```
F_list = [0.1, 0.5, 10.0, 25.0, 50.0, 75.0, 100.0]
AveragePiston_list = []
time_measure = 0
    x_o = np.array(x_o)
    v_o = np.array(v_o)
    c_microstates = np.column_stack((x_o, v_o))

    PistonPos_list.append(X_o)
    Vel_list.append(V_o)
    T_list.append(time_measure)

    maximum = N*20
    collisions = 0
```

```
while collisions < maximum:
   tau = []
   for x,v in c_microstates:
      tau.append(waiting_time(x,v,X_o,V_o,F,M))

quickest_time = np.min(tau)
index = tau.index(quickest_time)

displaced = []



for x,v in c_microstates:
   x_after = x + (v*quickest_time)
   displaced.append(x_after)

   x_after = np.abs(displaced[index])

   c_microstates = np.column_stack((displaced,v_o))

   nV = V_o+(-1*F/M)*quickest_time

   nparticleV = velocityimm_after(m,M,c_microstates[index,1], nV)[1]
   npistonV = velocityimm_after(m,M,c_microstates[index,1], nV)[0]

   time_measure += quickest_time

   v_o_list = v_o.tolist()
   v_o_list = v_o_list[:index] + [nparticleV]+v_o_list[index+1:]
   v_o = np.array(v_o_list)

   c_microstates = np.column_stack((displaced, v_o))
```

```
        PistonPos_list.append(x_after)
        Vel_list.append(npistonV)

        T_list.append(time_measure)

        collisions+=1
        X_o = x_after
        V_o = npistonV

    AveragePiston_list.append(np.average(PistonPos_list, weights = T_list))
plt.scatter(np.log10(F_list), np.log10(AveragePiston_list))
#now we add to the theoretical plot we want
exp_values = []

for F in F_list:
    exp_values.append(np.log10(N/F))

plt.plot(np.log10(F_list), exp_values, label = 'Analytic Equilibrium')
plt.scatter(np.log10(F_list), np.log10(AveragePiston_list), label = 'Our Simulation', color = 'black')
plt.xlabel('Log of Force')
plt.ylabel('$Log of Piston Equilibrium Position X_{bar}$')
plt.grid(True)
plt.title('Equlilibrium Position vs Force')
```

(c)

```
Mbi_Vel = []
Mbi_Tem = []
V_dist = np.array(V_dist)
Time_dist = np.array(Time_dist)
```

```
Mbi_Vel.append(v_o)

plt.hist(final_velocities, bins = 20, density = True, label = 'velocity distribution')

Time_dist = (T_list[0], T_list[2000], T_list[4000], T_list[8000], T_list[12000], T_list[14000],
T_list[17000], T_list[19999])

V_dist = (Mbi_Vel[0], Mbi_Vel [2000], Mbi_Vel [4000], Mbi_Vel [8000], Mbi_Vel [12000], Mbi_Vel
[14000], Mbi_Vel [17000], Mbi_Vel [19999])

hist, bins = np.histogram(v, bins=nbins)

  xs = (bins[:-1] + bins[1:])/2

            ax.bar(xs, hist, zs=t, zdir='y', color=c, ec=c, alpha=0.8)

ax.set_xlabel('Particle Velocity')

ax.set_ylabel('Time')

ax.set_zlabel('Frequency')

ax.set_zlim(0, 30)


Mdist = stats.maxwell

params = Mdist.fit(final_velocities  )

print('SD is', params[1])

x = np.linspace(-4, 4, 1000)

plt.hist(final_velocities, bins = 20, density = True, label = 'velocity distribution')

colors = ['y', 'g', 'r', 'b', 'black', 'c', 'r', 'm']

plt.xlabel('Velocities')

plt.ylabel('Frequency density')

plt.xlabel('Speed of particles')

plt.title('Final particle velocities fitted with Maxwell Distribution')

plt.plot(x, Mdist(x, *params), lw=3.5, label = 'Maxwell Distribution curve')

plt.legend()

plt.show()
```