STATISTICAL MECHANICS

Lab Report

Name: *Sarthak Jain*

Roll No: *2020PHY1201*

Submitted to: *Mr. Sushil Kumar Singh and Dr. Savinder Kaur*

Date: 19/04/2023

Course: *BSc Physics (H) 6th Sem*

EXPERIMENT 1:
CODE AND RESULTS

Gmail Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com> Wed, Apr 19, 2023 at 12:29 AM
To: jain.sarthak38@gmail.com

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 1: Coin Tossing

Code1 *

(about 10 lines)

```
import numpy as np
import matplotlib.pyplot as plt
import random
def macrostates(N_t,N_c):
result = []
for i in range(N_t): # (N_t)
outcomes = []
for j in range(N_c):
out = random.randint(0,1)
outcomes.append(out)
result.append(outcomes)
return result
def macrostates_counter(result,N_c):
result = np.array(result)
total = [] ; hc_array = []
for i in range(len(result)):
macro = []
h = sum(result[i])
hc_array.append(h)
t = N_c - h
macro.append(h)
macro.append(t)
total.append(macro)
return total,hc_array
```

Code2

(about 10 lines)

```
def head_count(hc_array,N_c):
pass
freq_h = []
```

```
for i in range(N_c+1):
f_h = hc_array.count(i)
freq_h.append(f_h)
return np.array(freq_h)
if __name__ == "__main__":
# Random number generator
# 1 = Heads
# 0 = Tails
N_c = 3
N_t = [10,100,1000,10000]
prob_nc = [] ; num_of_heads_nc = []
prob_nt = [] ; num_of_heads_nt = []
for k in N_t:
num_of_heads = [i for i in range(0,N_c+1)]
num_of_heads_nt.append(num_of_heads)
result = macrostates(k,N_c)
total,hc_array = macrostates_counter(result,N_c)
freq_h = head_count(hc_array,N_c)
probability = freq_h/k
prob_nt.append(probability)
fig1,ax1 = plt.subplots()
```

Code3

```
 for i in range(len(prob_nt)):
ax1.plot(num_of_heads_nt[i],prob_nt[i],label = "No of trails = "+str(N_t[i]))
ax1.scatter(num_of_heads_nt[i],prob_nt[i])
ax1.set(xlabel = "No of Heads",ylabel = "Probabilty of macrostates",title = "Probility of
macrostates V/s No. of Heads at constant no. of coins ("+str(N_c)+")")
ax1.grid(ls = "--")
ax1.legend()
plt.show()
N_c = [1,2,3,4,5,6,7,8,9,10]
N_t = 1000
for k in N_c:
num_of_heads = [i for i in range(0,k+1)]
num_of_heads_nc.append(num_of_heads)
result = macrostates(N_t,k)
total,hc_array = macrostates_counter(result,k)
freq_h = head_count(hc_array,k)
probability = freq_h/N_t
prob_nc.append(probability)
print("Probability (at constant no. of trails):\n",prob_nc)
print("\nProbability (at constant no. of coins)\n",prob_nt)
fig2,ax2 = plt.subplots()
```

Code4

(about 10 lines)

```
 for i in range(len(prob_nc)):
ax2.plot(num_of_heads_nc[i],prob_nc[i],label = "No of coins = "+str(N_c[i]))
ax2.scatter(num_of_heads_nc[i],prob_nc[i])
ax2.set(xlabel = "No of Heads",ylabel = "Probabilty of macrostates",title = "Probility of
macrostates V/s No. of Heads at constant no. of trails ("+str(N_t)+")")
ax2.grid(ls = "--")
ax2.legend()
plt.show()
N_c = 3
N_t = 50
no_of_trails = []
cum_prob_head = [] ; cum_prob_tail = [] ; cum_freq_head = [] ; cum_freq_tail = []
prob_head = 0 ; prob_tail = 0 ; freq_head = 0 ; freq_tail = 0
for k in range(1,N_t+1):
no_of_trails.append(k)
total_outcomes = 3*k
result = macrostates(k,N_c)
freq_head = freq_head + (sum(result[k-1]))
freq_tail = freq_tail + (N_c - sum(result[k-1]))
prob_head = freq_head/total_outcomes
prob_tail = freq_tail/total_outcomes
cum_freq_head.append(freq_head)
cum_freq_tail.append(freq_tail)
cum_prob_head.append(prob_head)
cum_prob_tail.append(prob_tail)
freq_head = cum_freq_head[k-1]
freq_tail = cum_freq_tail[k-1]
```

Code5

(about 10 lines)

```
 fig3,ax3 = plt.subplots()
ax3.plot(no_of_trails,cum_prob_head,label = "Heads")
ax3.plot(no_of_trails,cum_prob_tail,label = "Tails")
ax3.scatter(no_of_trails,cum_prob_head)
ax3.scatter(no_of_trails,cum_prob_tail)
ax3.set(xlabel = "No. of trails",ylabel = "Cumulative probability of head and tails",title =
"Cumulative probability Vs no of trails")
ax3.grid(ls = "--")
ax3.legend()
plt.show()
```

Code6

(about 10 lines)

Plot1

Submitted files

Figure_7 - Sarthak Jain.png

Plot2

Submitted files

Figure_8 - Sarthak Jain.png

Plot3

Submitted files

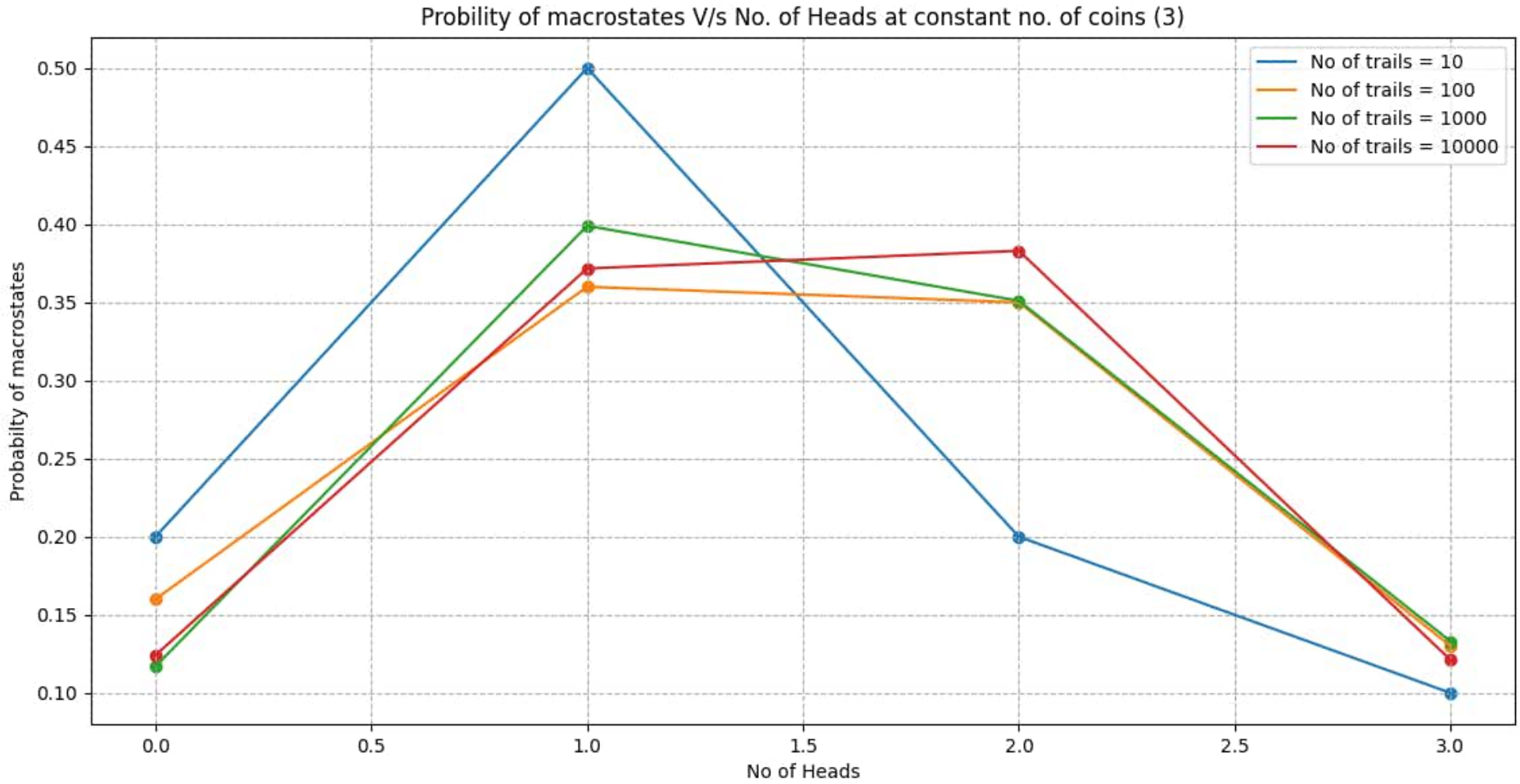
Figure_9 - Sarthak Jain.png

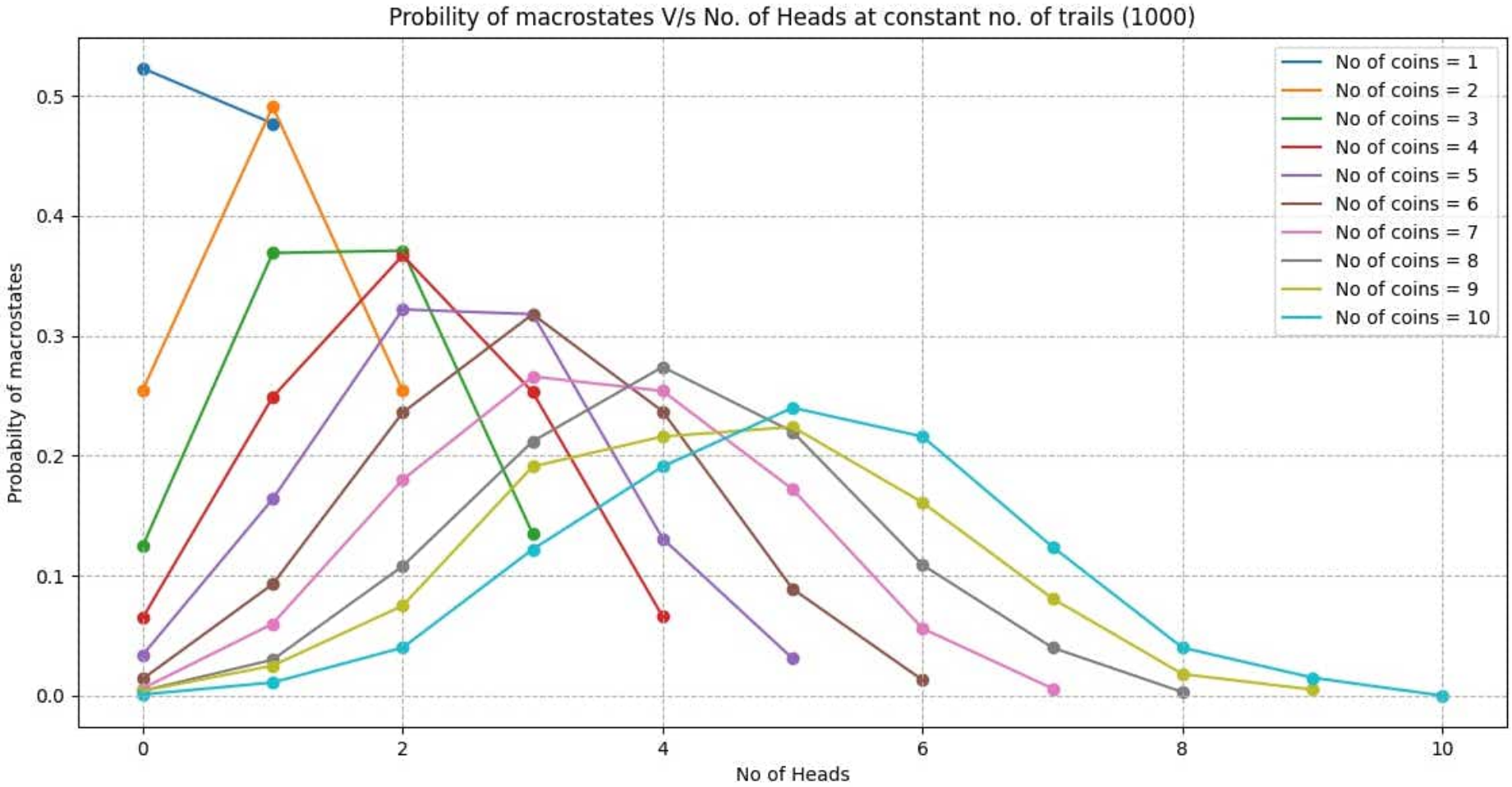
Plot4

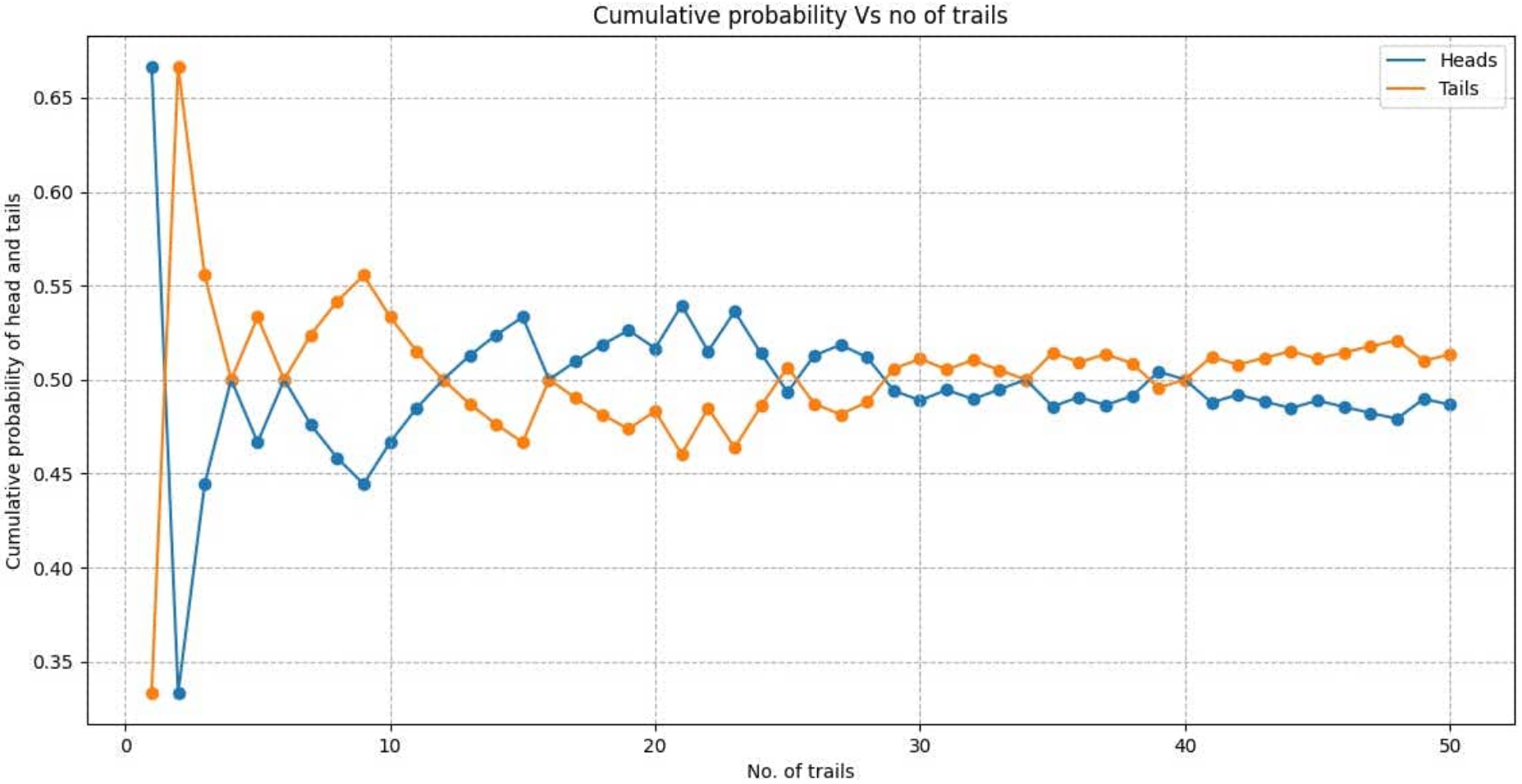
No files submitted

Comments

[Quoted text hidden]


Probility of macrostates V/s No. of Heads at constant no. of coins (3)
No of trails = 10
No of trails = 100
No of trails = 1000
No of trails = 10000
Probabilty of macrostates
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.0
0.5
1.0
1.5
2.0
2.5
3.0
No of Heads


Probility of macrostates V/s No. of Heads at constant no. of trails (1000)
Probabilty of macrostates
No of Heads
0.0
0.1
0.2
0.3
0.4
0.5
0
2
4
6
8
10
No of coins = 1
No of coins = 2
No of coins = 3
No of coins = 4
No of coins = 5
No of coins = 6
No of coins = 7
No of coins = 8
No of coins = 9
No of coins = 10


Cumulative probability Vs no of trails
Heads
Tails
Cumulative probability of head and tails
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0
10
20
30
40
50
No. of trails

EXPERIMENT 2:
CODE AND RESULTS

Gmail

Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com>
To: jain.sarthak38@gmail.com

Wed, Apr 19, 2023 at 1:01 AM

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 2: Distribution Functions

Code1 *

(about 10 lines)

```
import numpy as np
import matplotlib.pyplot as plt
def max_bolt(x):
return np.exp(-x)
def bose_einstein(x,alpha):
return 1/(np.exp(x+alpha) - 1)
def fermi_dirac(x,alpha):
return 1/(np.exp(x+alpha) + 1)
if __name__ == "__main__":
x_range_max = np.linspace(-4,4,50)
x_range_bose = np.linspace(0.1,4,50)
x_range_fermi = np.linspace(-4,4,50)
alpha = [0,1]
k = 8.617333 * 10**(-5)
T = np.array([10,100,1000,5000])
fermi_total_fermi = [] ; fermi_total_bose = [] ; fermi_total_max = []
f_max_bolt = max_bolt(x_range_max)
f_bose_einstein = bose_einstein(x_range_bose,alpha[0])
f_fermi_dirac = fermi_dirac(x_range_fermi,alpha[1])
fig, ax1 = plt.subplots(1, 3, figsize=(10, 4))
```

Code2

(about 10 lines)

```
 ax1[0].plot(x_range_max,f_max_bolt)
ax1[0].scatter(x_range_max,f_max_bolt,marker = ".")
ax1[1].plot(x_range_bose,f_bose_einstein,c = "g")
ax1[1].scatter(x_range_bose,f_bose_einstein,c = "g",marker = ".")
ax1[2].plot(x_range_fermi,f_fermi_dirac,c = "r")
ax1[2].scatter(x_range_fermi,f_fermi_dirac,c = "r",marker = ".")
for i in range(3):
```

```
ax1[i].set(xlabel = "$\epsilon$/KT",ylabel = "f($\epsilon$)")
ax1[i].grid(ls = "--")
ax1[0].set(title = "Maxwell Boltzman Distribution")
ax1[1].set(title = "Bose Einstein Distribution")
ax1[2].set(title = "Fermi Dirac Distribution")
plt.show()
for i in range(len(T)):
fermi_x = x_range_fermi * T[i]*k
fermi_total_fermi.append(fermi_x)
```

Code3

```
for i in range(len(T)):
fermi_x = x_range_max * T[i]*k
fermi_total_max.append(fermi_x)
for i in range(len(T)):
fermi_x = x_range_bose * T[i]*k
fermi_total_bose.append(fermi_x)
fig2,ax2 = plt.subplots()
fig3,ax3 = plt.subplots()
fig4,ax4 = plt.subplots()
```

Code4

(about 10 lines)

```
for i in range(len(fermi_total_fermi)):
ax2.plot(fermi_total_fermi[i],f_fermi_dirac,label = "At T = "+str(T[i])+" K")
ax2.scatter(fermi_total_fermi[i],f_fermi_dirac,marker = ".")
ax3.plot(fermi_total_max[i],f_max_bolt,label = "At T = "+str(T[i])+" K")
ax3.scatter(fermi_total_max[i],f_max_bolt,marker = ".")
ax4.plot(fermi_total_bose[i],f_bose_einstein,label = "At T = "+str(T[i])+" K")
ax4.scatter(fermi_total_bose[i],f_bose_einstein,marker = ".")
ax2.set(xlabel = "$\epsilon$",ylabel = "f($\epsilon$)",title = "Fermi Dirac distribution at constant
temperature")
ax3.set(xlabel = "$\epsilon$",ylabel = "f($\epsilon$)",title = "Maxwell Boltzman distribution at
constant temperature")
ax4.set(xlabel = "$\epsilon$",ylabel = "f($\epsilon$)",title = "Bose Einstein distribution at
constant temperature")
ax2.grid(ls = "--")
ax3.grid(ls = "--")
ax4.grid(ls = "--")
ax2.legend()
ax3.legend()
```

```
ax4.legend()
plt.show()
```

Code5

(about 10 lines)

Code6

(about 10 lines)

Plot1

Submitted files

Figure_10 - Sarthak Jain.png

Plot2

Submitted files

Figure_11 - Sarthak Jain.png

Plot3

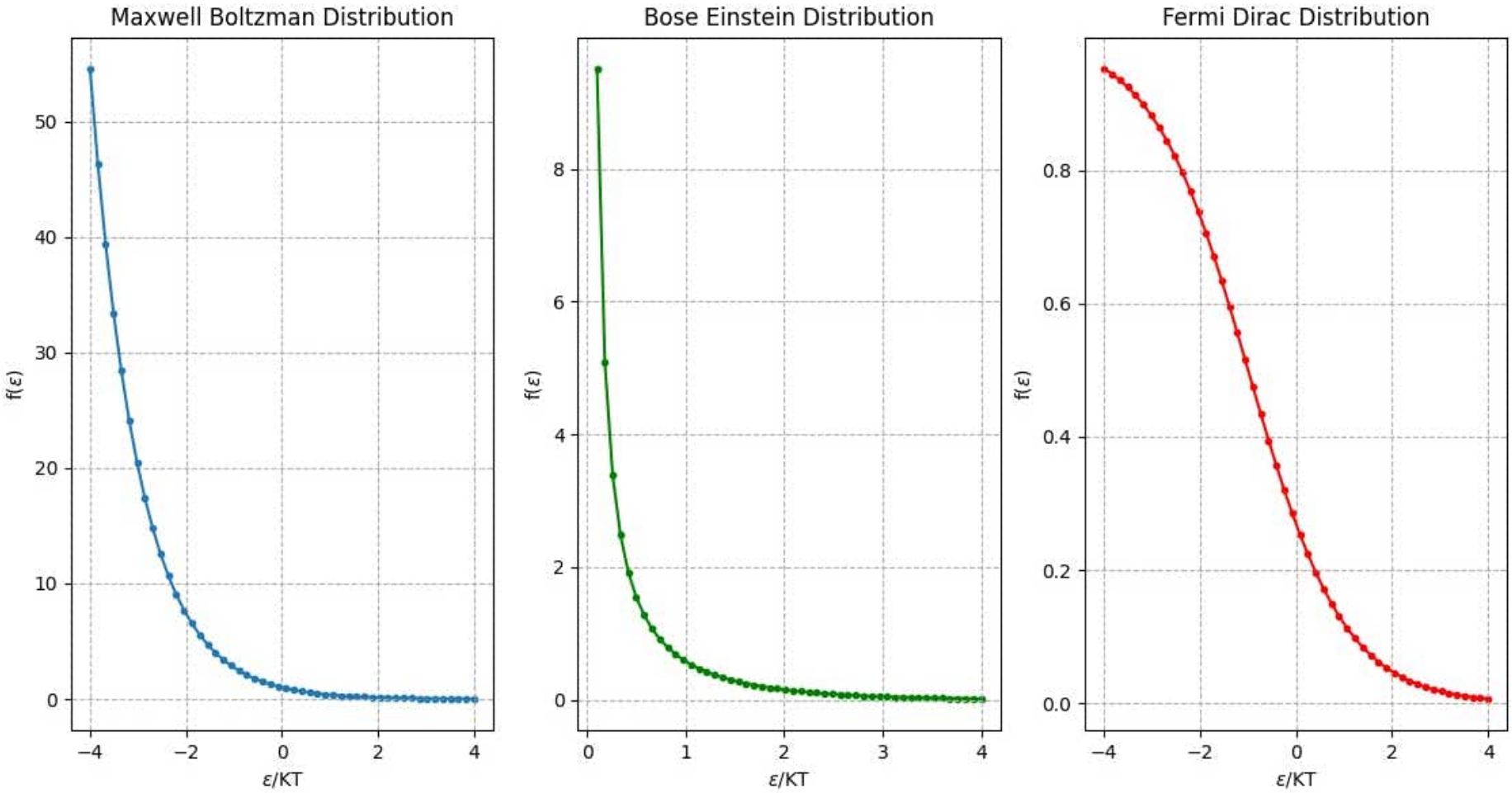
Submitted files

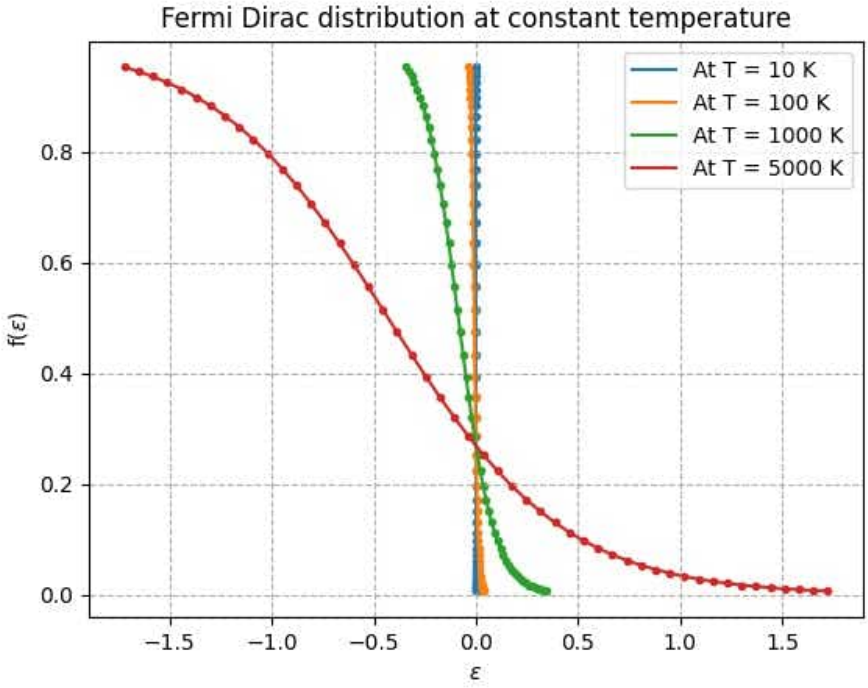
Figure_12 - Sarthak Jain.png

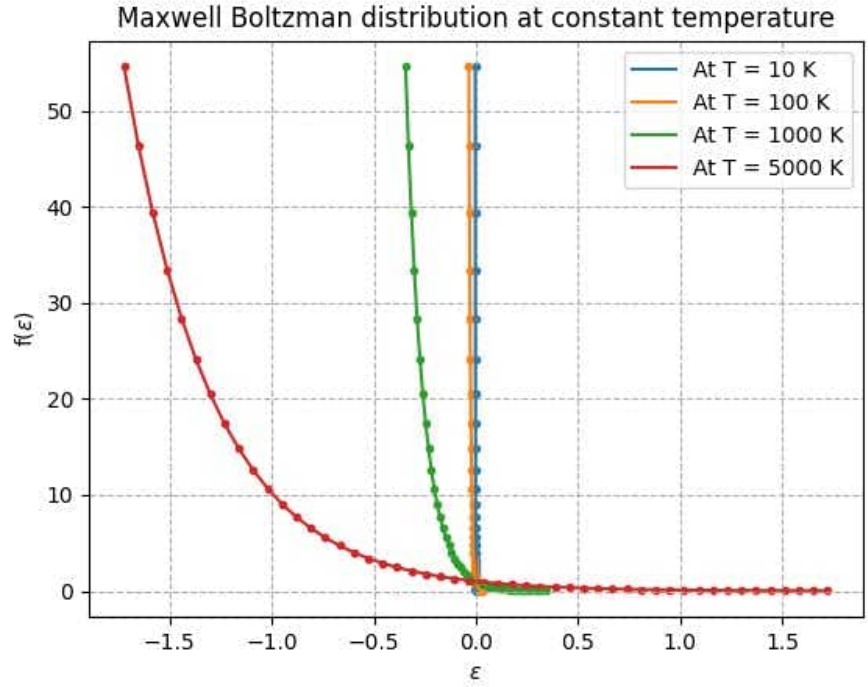
Plot4

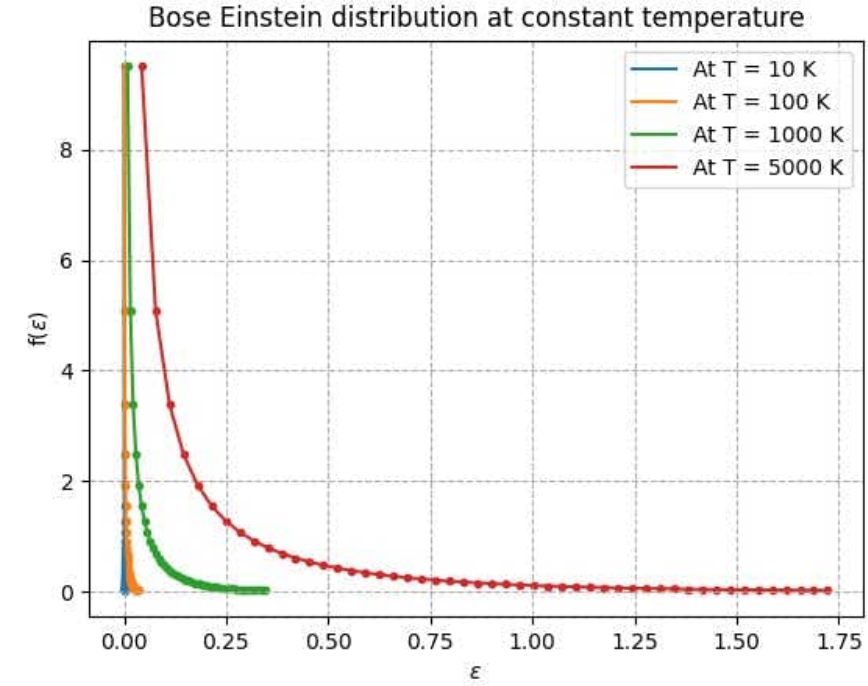
Submitted files

Figure_13 - Sarthak Jain.png


Maxwell Boltzman Distribution
f(ε)
ε/KT
Bose Einstein Distribution
f(ε)
ε/KT
Fermi Dirac Distribution
f(ε)
ε/KT


Fermi Dirac distribution at constant temperature
At T = 10 K
At T = 100 K
At T = 1000 K
At T = 5000 K
f(ε)
ε


Maxwell Boltzman distribution at constant temperature
At T = 10 K
At T = 100 K
At T = 1000 K
At T = 5000 K
f(ε)
ε


Bose Einstein distribution at constant temperature
At T = 10 K
At T = 100 K
At T = 1000 K
At T = 5000 K
f(ε)
ε

EXPERIMENT 3:
CODE AND RESULTS

Gmail

Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com> Thu, Feb 2, 2023 at 8:34 PM
To: jain.sarthak38@gmail.com

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 3: To study the specific heat of solids.

Code1 *

(about 10 lines)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
theta_e = 230
theta_d = 343
r = theta_e/theta_d
nu = np.linspace(0,2,100)
def dulong_petit(x):
return np.ones(len(x))
def einstein(x):
return ((1/x)**2) * ((np.exp(1/x))/(np.exp(1/x) - 1)**2)
einstein = np.vectorize(einstein)
def debye(x):
u1 = -3*x/(np.exp(x)-1)
u2 = 12/x**3
ine = quad(lambda x : (x**3/(np.exp(x)-1)),0,x)
return u1 + u2*ine[0]
debye = np.vectorize(debye)
```

Code2

(about 10 lines)

```
def dos_einstein(nu):
if abs(nu-1) <= 0.02:
return 1
else:
return 0
def dos_debye(nu):
if nu <= 1/r:
return nu**2
```

```
else:
return 0
dos_einstein = np.vectorize(dos_einstein)
dos_debye = np.vectorize(dos_debye)
"Einstein distribution Law"
Temp_einstein = np.linspace(1e-8,2*theta_e,100)/theta_e
Specific_einstein = einstein(Temp_einstein)
"Debye distribution Law"
Temp_debye = np.linspace(1e-8,2*theta_d,100)/theta_d
Specific_debye = debye(1/Temp_debye)
"Dulong-Petit distribution Law"
Temp_dulong = np.linspace(Temp_debye[0],Temp_debye[-1],100)
Specific_dulong = dulong_petit(Temp_dulong)
```

Code3

```
# Figure 1: Density of States
fig,ax = plt.subplots(figsize = (16,8))
ax.plot(nu,dos_einstein(nu),label = 'Einstein', c = 'b')
ax.plot(nu,dos_debye(nu)/dos_debye(1/r),label = 'Debye', c = 'r')
ax.scatter(nu,dos_debye(nu)/dos_debye(1/r),marker = ".",c = "r")
ax.set_ylim([0,1.1])
ax.set(xlabel = '$\\nu$/$\\nu_x$',title = "Density of States",ylabel = "No. of States")
ax.legend()
ax.grid(ls = "--")
plt.show()
# Figure 2: Specific Heat Plot
fig1,ax1 = plt.subplots(figsize = (16,8))
ax1.plot(Temp_dulong,Specific_dulong,label = "Dulong-Petit distribution Law",ls = "--", c = 'r')
ax1.plot(Temp_einstein,Specific_einstein,label = "Einstein distribution Law",c = "b")
ax1.plot(Temp_debye,Specific_debye,label = "Debye distribution Law",c = "g")
ax1.scatter(Temp_einstein,Specific_einstein,marker = ".",c = "b")
ax1.scatter(Temp_debye,Specific_debye,marker = ".",c = "g")
ax1.set(xlabel = r"T/$\theta$",ylabel = "$\dfrac{C_v}{3R}$",title = "Specific Heat of Solids")
ax1.legend()
ax1.grid(ls = "--")
plt.show()
```

Code4

(about 10 lines)

Code5

(about 10 lines)

Code6

(about 10 lines)

Plot1

Submitted files

Figure_1 - Sarthak Jain.png

Plot2

Submitted files

Figure_2 - Sarthak Jain.png

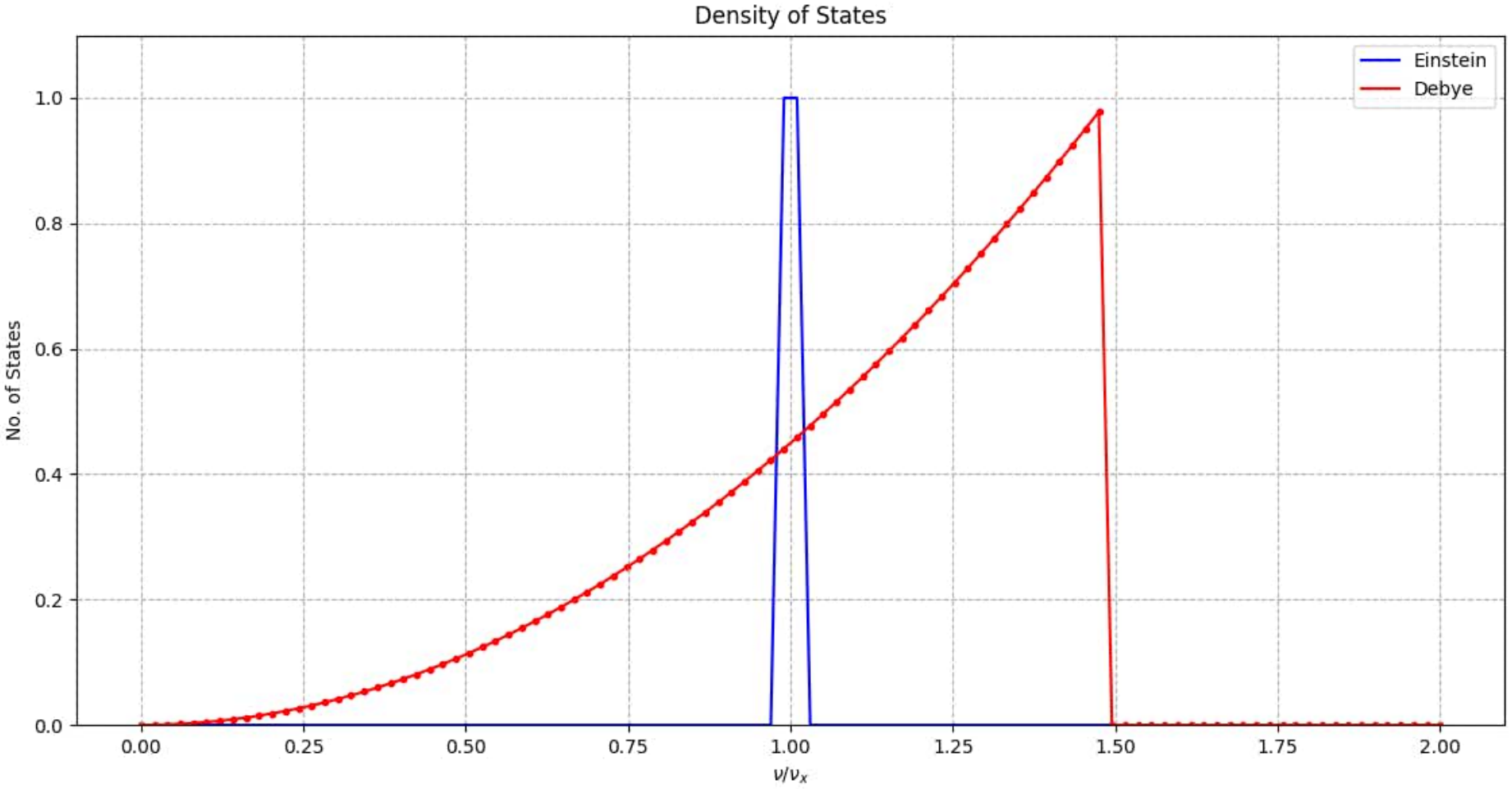
Plot3

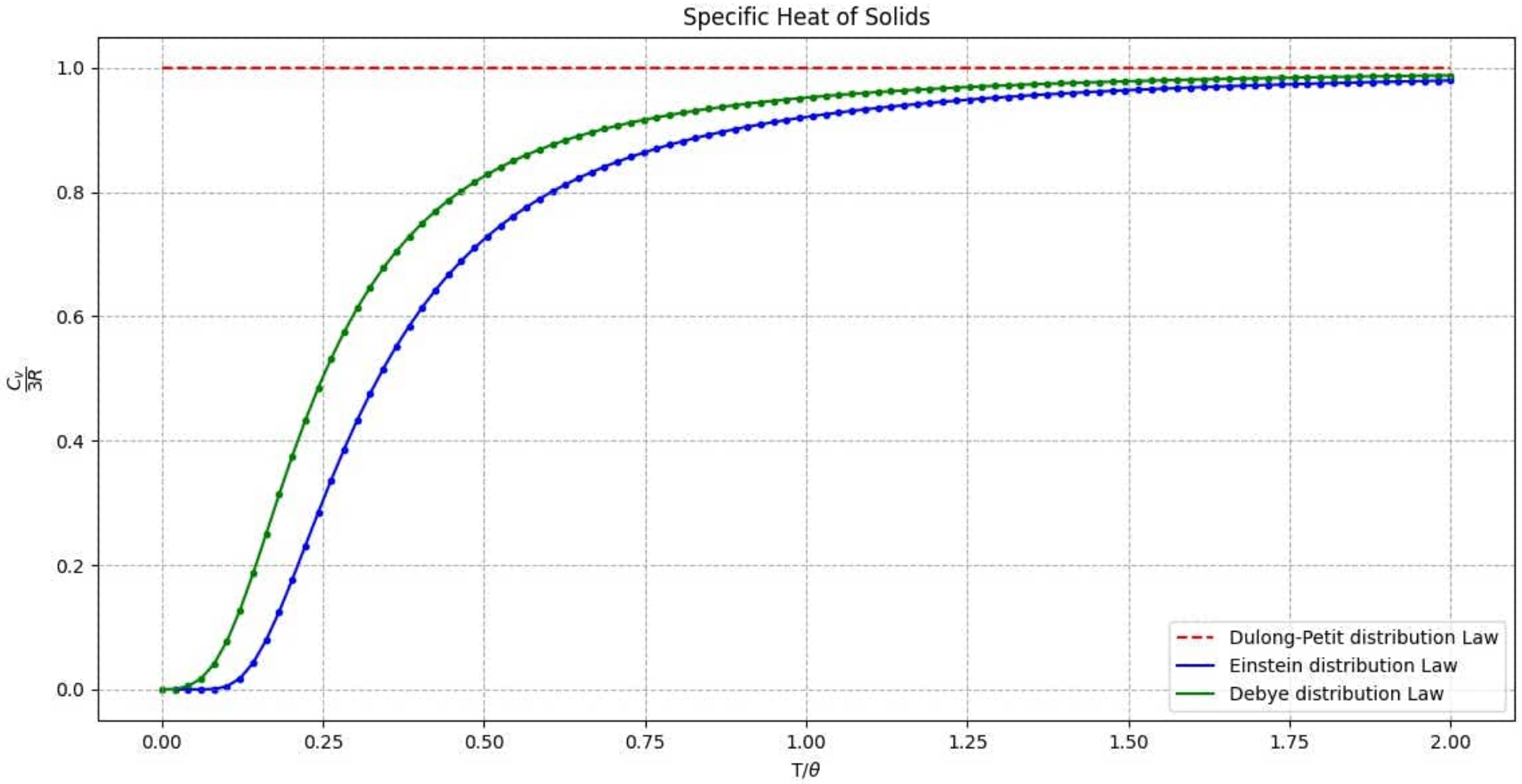
No files submitted

Plot4

No files submitted

Comments


Density of States
Einstein
Debye
No. of States
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
ν/ν_x


Specific Heat of Solids
1.0
0.8
0.6
0.4
0.2
0.0
$\frac{C_v}{3R}$
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
T/θ
Dulong-Petit distribution Law
Einstein distribution Law
Debye distribution Law

EXPERIMENT 4:
CODE AND RESULTS

Gmail Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com> Thu, Feb 16, 2023 at 11:07 PM
To: jain.sarthak38@gmail.com

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 4 - To study the radiation laws.

Code1 *

(about 10 lines)

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress
def F_rj(x):
return (x**2)
def F_p(x):
return x**3/(np.exp(x)-1)
x=np.linspace(0,12,151)
plt.scatter(x * np.pi,F_rj(x),label='Rayleigh Jeans')
plt.grid(ls='--')
#plt.xscale("log")
#plt.yscale("log")
plt.xlabel("x")
plt.ylabel("G(x)")
plt.title("Density of States(Rayleigh Jeans)")
plt.legend()
plt.show()
plt.scatter(x * np.pi,F_p(x),label='Planck')
plt.grid(ls='--')
plt.xlabel("x")
plt.ylabel("G(x)")
#plt.xscale("log")
#plt.yscale("log")
plt.legend()
plt.title("Density of States (Planck)")
plt.show()
h=6.62e-34
c=3e8
temp=[1200,1500,1800]
k=1.38*10**(-23)
l=1e-10
```

Code2

(about 10 lines)

```
def Ray(x,t):
const = 8 * ((k)**4)/((h*c)**3)
return (np.pi)*((t)**4)*const*F_rj(x)
for i in temp:
plt.scatter(x*(k*i)/(2*h),Ray(x,i)*(2*h/k*i),marker='.',label='For Temp. ' + str(i) + 'K')
plt.grid(ls='--')
plt.xlabel("v")
plt.ylabel("U(v)")
plt.legend()
plt.title("Rayleigh Jeans ")
plt.show()
def Planck(x,t):
const = 8*((k)**4)/((h*c)**3)
return (np.pi)*((t)**4)*const*F_p(x)
for i in temp:
plt.scatter(x*(k*i)/(2*h),Planck(x,i)*(2*h/k*i),marker='.',label='For Temp. ' + str(i) + 'K')
plt.grid(ls='--')
plt.xlabel("v")
plt.ylabel("U(v)")
plt.legend()
plt.title("Planck ")
plt.show()
```

Code3

```
#Plotting with Solar Temp
sol=[6000]
for i in sol:
plt.scatter(x*(k*i)/(2*h),Ray(x,i)*(2*h/k*i),marker='.',label='For Temp. ' + str(i) + 'K')
plt.grid(ls='--')
plt.xlabel("v")
plt.ylabel("U(v)")
plt.legend()
plt.title("Rayleigh Jeans (Solar Temp)")
plt.show()
for i in sol:
plt.scatter(x*(k*i)/(2*h),Planck(x,i)*(2*h/k*i),marker='.',label='For Temp. ' + str(i) + 'K')
plt.grid(ls='--')
plt.xlabel("v")
plt.ylabel("U(v)")
plt.legend()
plt.title("Planck (Solar Temp)")
plt.show()
def density(v):
return (8*np.pi)/(c**3)* (v**2)*(l**3)
```

```
v_complete=np.logspace(10,30,3000)
plt.scatter(v_complete,density(v_complete),marker='.')
plt.xscale("log")
plt.yscale("log")
plt.xlabel("Frequency(v)")
plt.ylabel("Density of states")
plt.title("Density of states with Frequency")
plt.grid(ls='--')
plt.show()
```

Code4

(about 10 lines)

Code5

(about 10 lines)

Code6

(about 10 lines)

Plot1

Submitted files

image1 - Sarthak Jain.png

Plot2

Submitted files

image2 - Sarthak Jain.png

Plot3

Submitted files

image3 - Sarthak Jain.png

Plot4

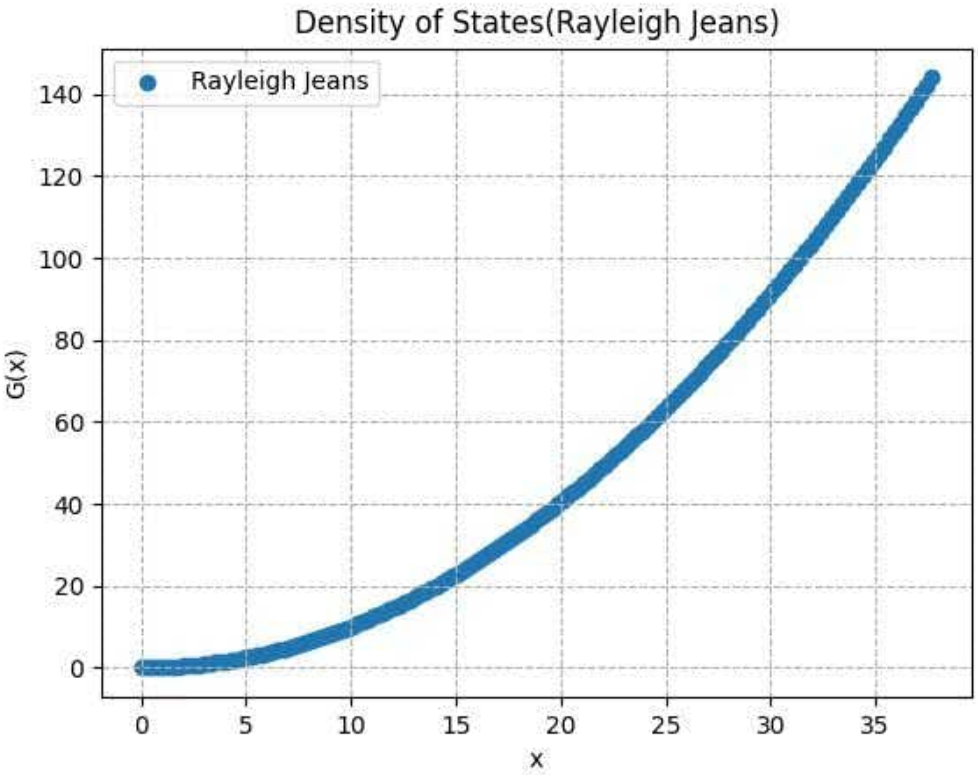
Submitted files

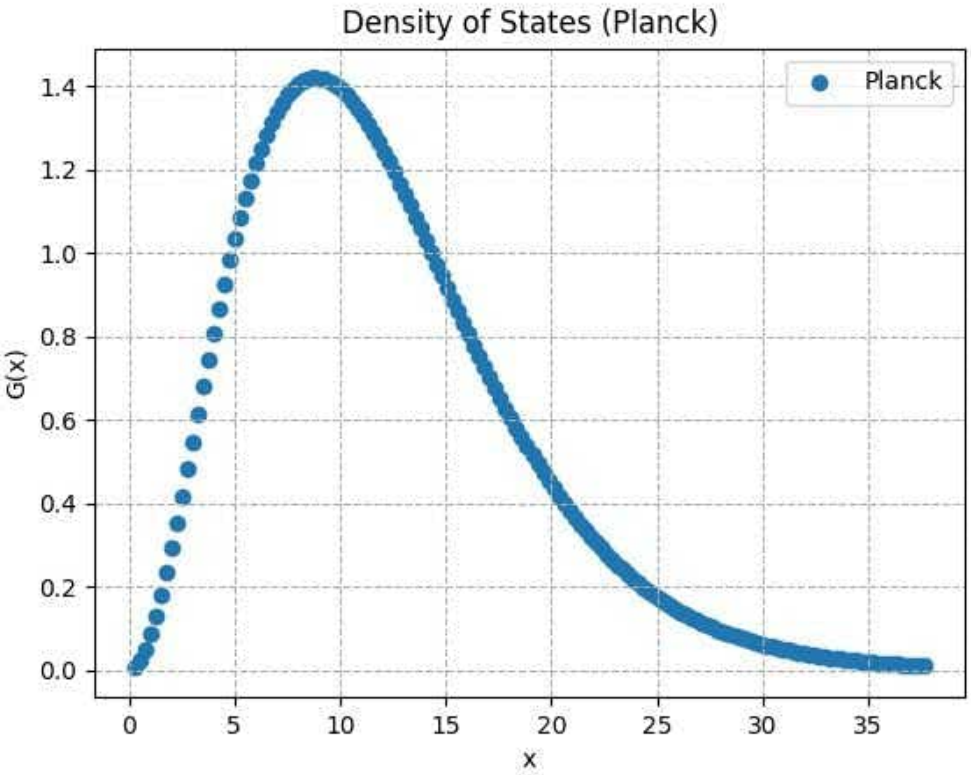
Figure_7 - Sarthak Jain.png

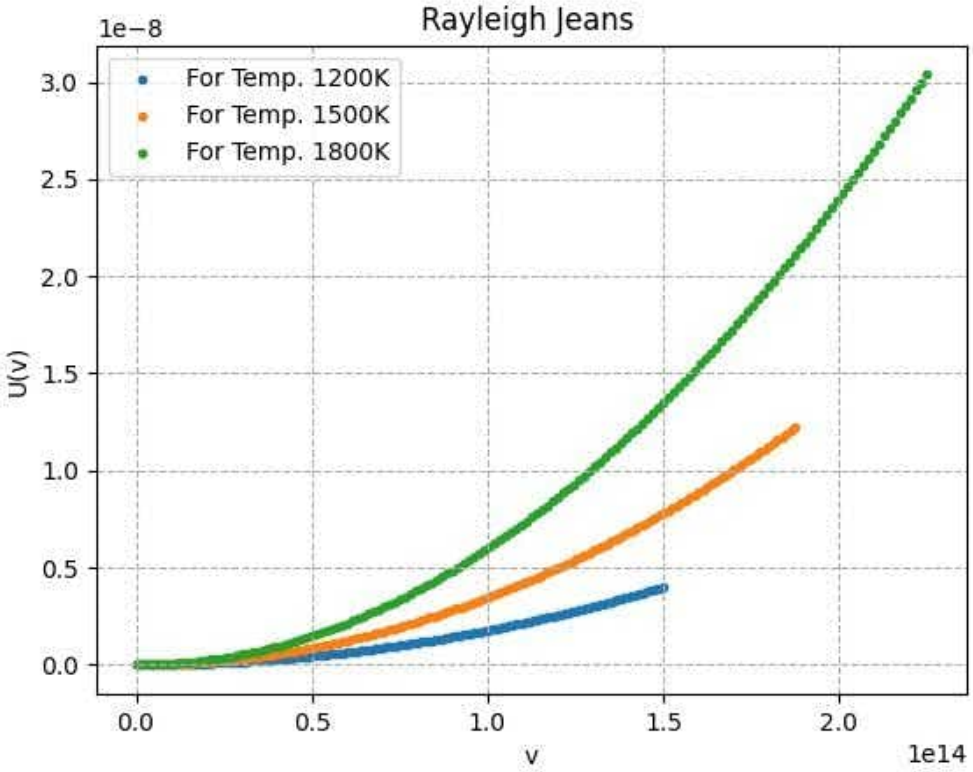
Comments

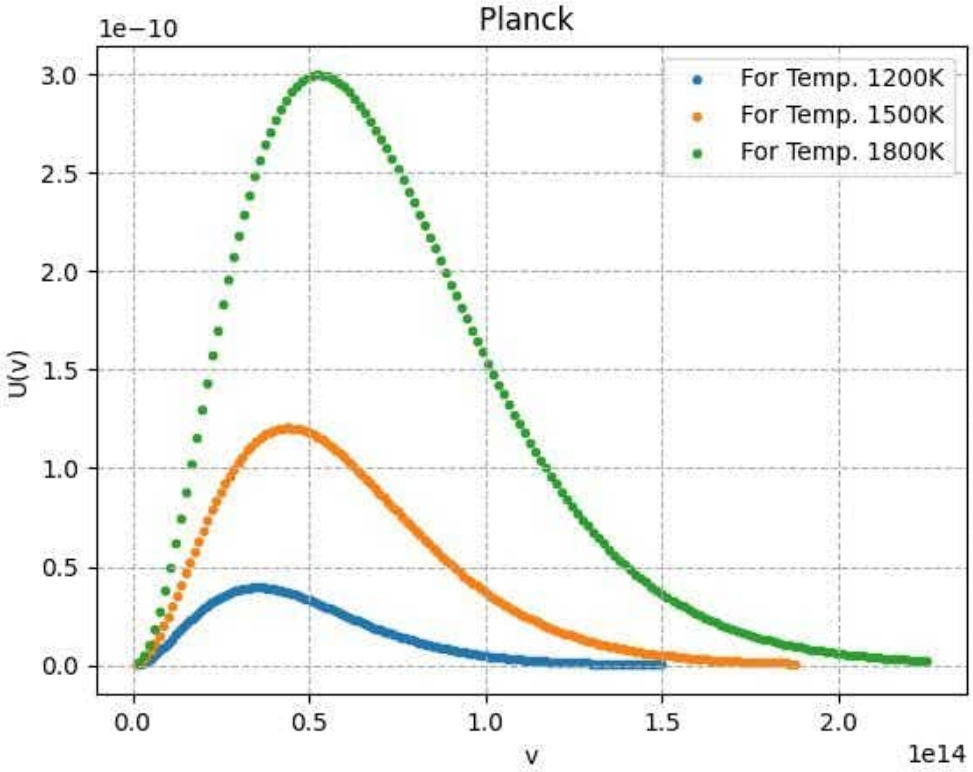
Create your own Google Form

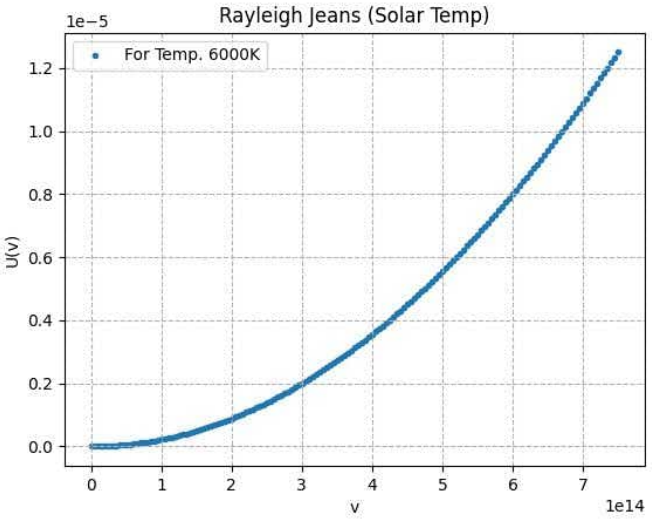
Report Abuse

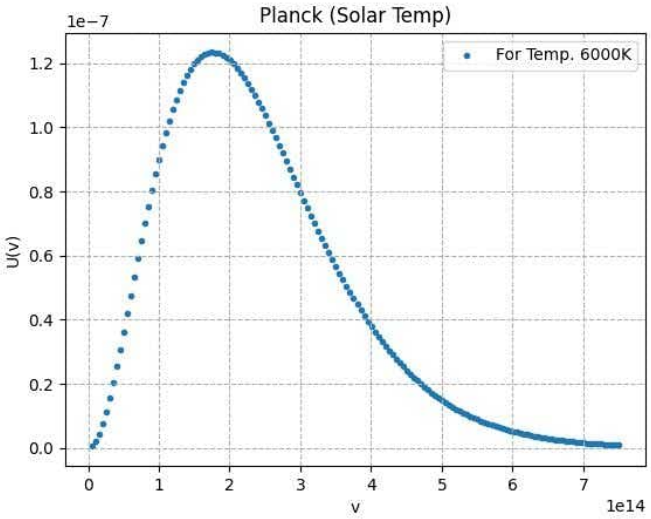

Density of States(Rayleigh Jeans)
Rayleigh Jeans
G(x)
x
0
20
40
60
80
100
120
140
0
5
10
15
20
25
30
35

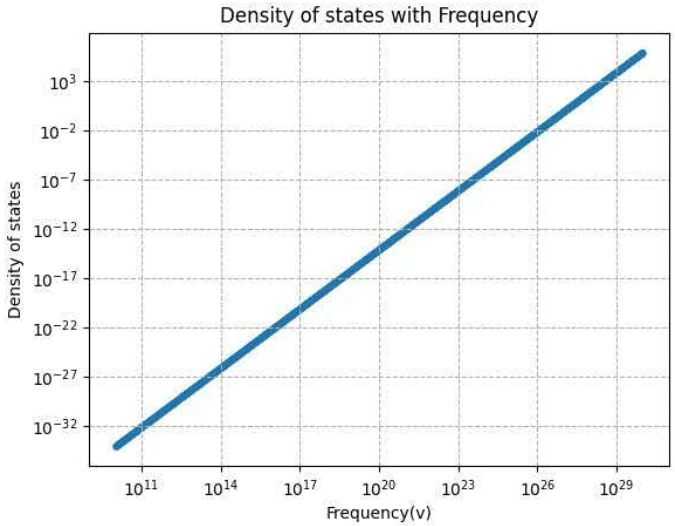

Density of States (Planck)
Planck
G(x)
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
5
10
15
20
25
30
35


Rayleigh Jeans
1e−8
For Temp. 1200K
For Temp. 1500K
For Temp. 1800K
U(v)
v
1e14
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Planck
1e−10
For Temp. 1200K
For Temp. 1500K
For Temp. 1800K
U(v)
v
1e14
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Rayleigh Jeans (Solar Temp)
1e−5
For Temp. 6000K
U(v)
v
1e14


Planck (Solar Temp)
1e−7
For Temp. 6000K
U(v)
v
1e14


Density of states with Frequency
Density of states
Frequency(v)

EXPERIMENT 5:
CODE AND RESULTS

Gmail

Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com> Tue, Feb 28, 2023 at 7:55 PM
To: jain.sarthak38@gmail.com

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 5: To study the Wien's Displacement Law and Stefan-Boltzmann Law.

Code1 *

(about 10 lines)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as int1
from scipy.stats import linregress
def F_p(x):
return (x**3)/(np.exp(x)-1)
x=np.linspace(1e-8,12,151)
X=np.array(x)
P=np.array(F_p(x))
m = np.max(F_p(x))
t = F_p(x)
i_val = np.where(t == m)
plt.scatter(x,F_p(x))
plt.grid(ls='--')
plt.xlabel("x")
plt.ylabel("U(x)")
plt.title("Spectral Energy Density")
plt.show()
Area=int1.quad(F_p,0,12)[0]
print("1.(a) Value of x_median is: \n" , Area/2)
h=6.62e-34
c=3e8
k=1.38*10**(-23)
T = 5800
b = (h*c)/(k*Area/2)
```

Code2

(about 10 lines)

```
print("1.(b) Value of Wien's Displacement Constant,b is: \n",h*c/(k*Area/2), "metres Kelvin")
print("Hence, Wien's Displacement Law has been verified \n")
print("Corresponding Wavelength is: ", b/T, "metres")
I_1=int1.simps(P,X)
print("2. The value of integral I_p is: \n",I_1)
test = ((np.pi)**4)/15
print("Value of RHS: \n",test)
print("Hence, I_p = pi**4/15 is shown \n")
def U(T):
return (np.pi**4)/(15) * 8*(np.pi)*((k*T)**4/(h*c)**3)
T=np.arange(100,10000,500)
values=[]
for i in T:
value=U(i)*(15/(np.pi)**4)
values.append(value)
def radiant_flux(T,C):
return (c/4)*((np.pi)**4/(15)) *C
```

Code3

```
for i in T:
value=U(i)*(15/(np.pi)**4)
plt.scatter(i,radiant_flux(i,value) , label='T =' + str(i))
plt.xlabel("Temperature")
plt.ylabel("F(T)")
plt.title("2.(a) Plot of Radiant Flux vs Temperature")
plt.grid(ls="--")
plt.show()
temp=[]
for i in T:
value=U(i)*(15/(np.pi)**4)
plt.scatter(np.log(i),np.log(radiant_flux(i,value)))
temp.append(radiant_flux(i,value))
slope=linregress(np.log(T) , np.log(temp))
print("slope",slope[0])
print("intercept",(slope[1]))
plt.legend()
plt.xlabel("Temperature")
plt.ylabel("F(T)")
plt.title("2.(b) Log plot of Radiant Flux vs Temperature")
plt.grid(ls='--')
plt.show()
print ("Value of Stefan Boltzman law constant is: \n" , np.exp(slope[1]), "Joules K^-4 m^-2 s^-1")
```

Code4

(about 10 lines)

Code5

(about 10 lines)

Code6

(about 10 lines)

Plot1

Submitted files

Capture - Sarthak Jain.PNG

Plot2

Submitted files

Figure_1 - Sarthak Jain.png

Plot3

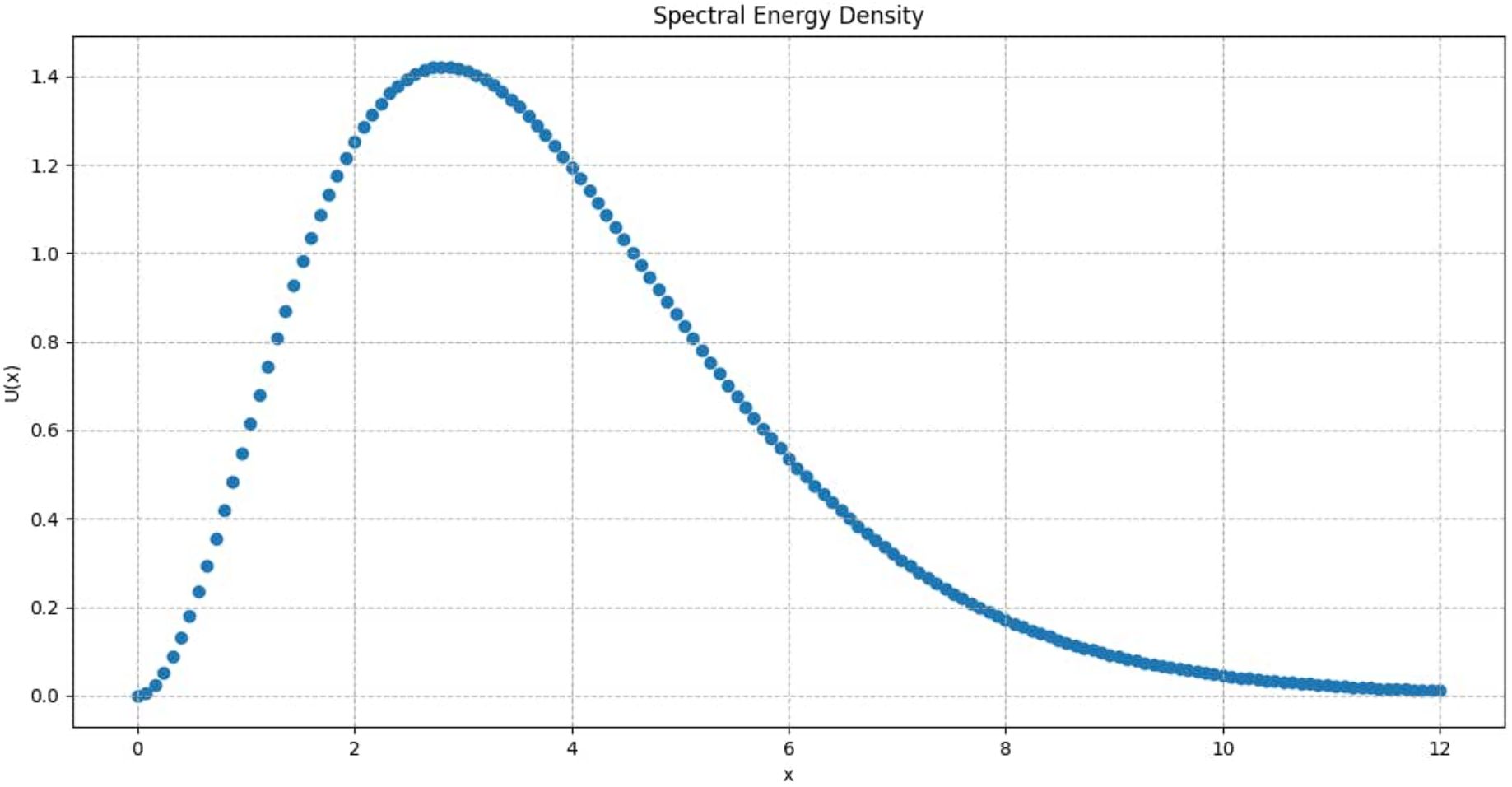
Submitted files

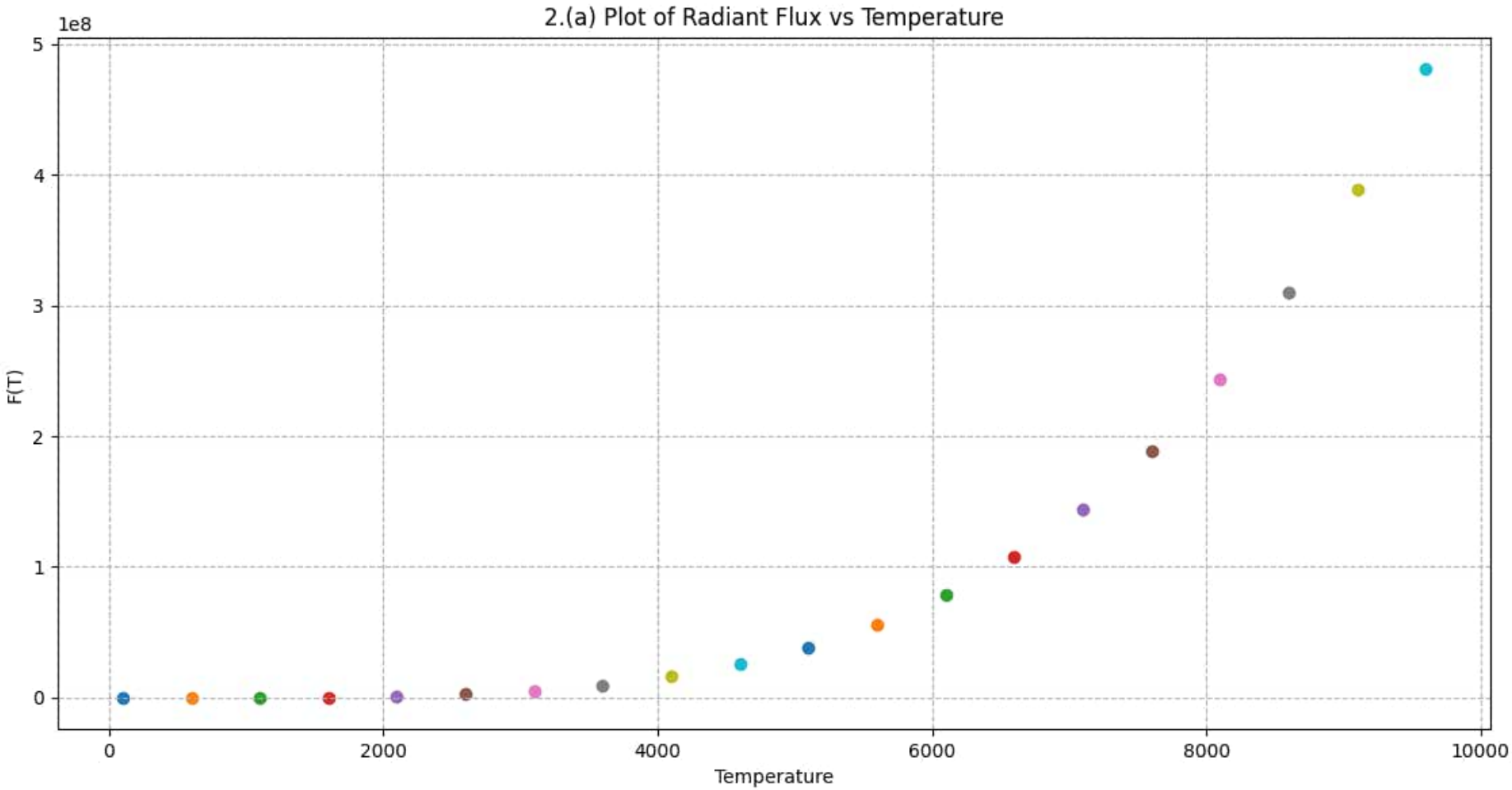
Figure_2 - Sarthak Jain.png

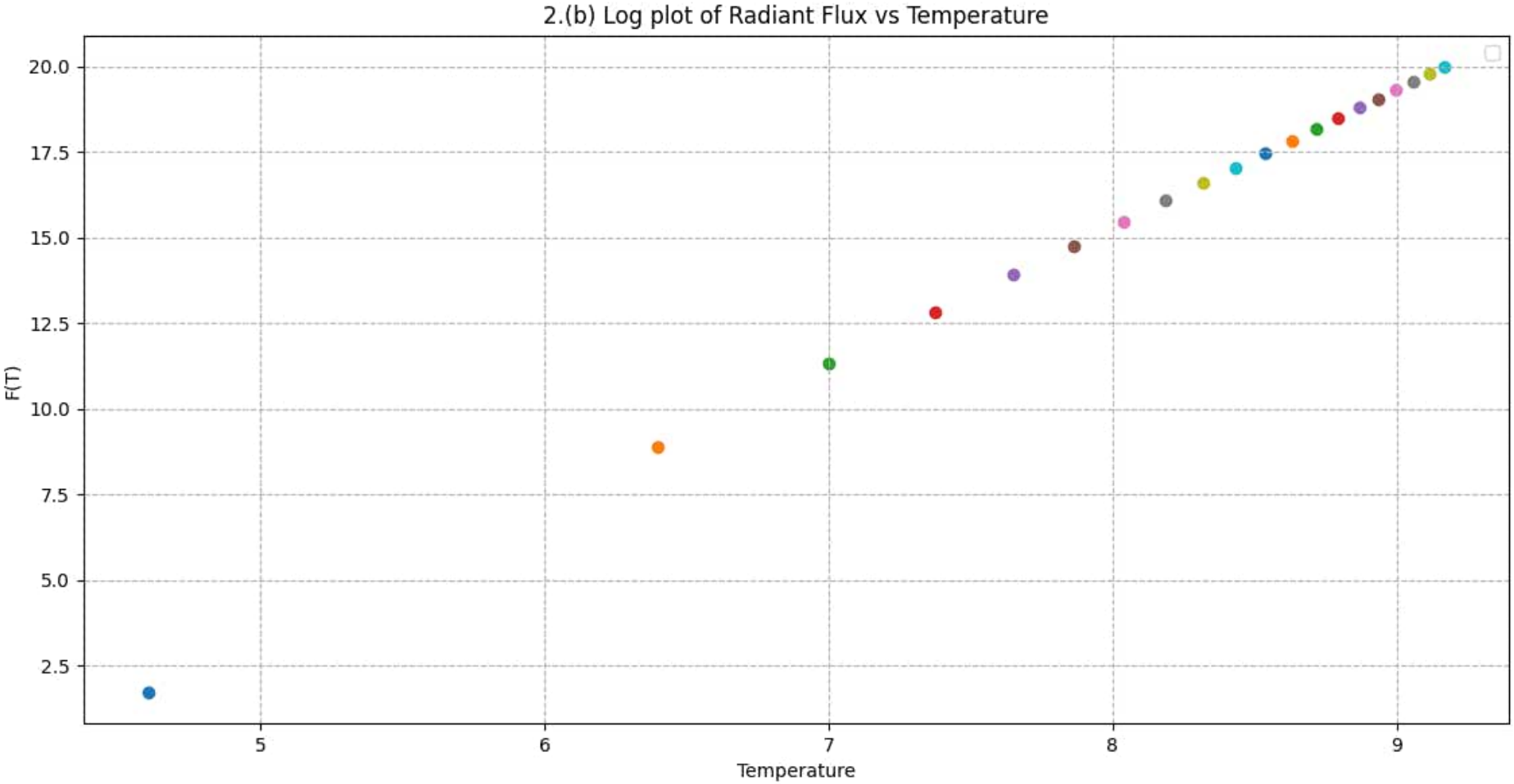
Plot4

Submitted files

Figure_3 - Sarthak Jain.png


Spectral Energy Density
U(x)
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
12


2.(a) Plot of Radiant Flux vs Temperature
1e8
5
4
3
2
1
0
F(T)
0
2000
4000
6000
8000
10000
Temperature


2.(b) Log plot of Radiant Flux vs Temperature
F(T)
Temperature
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
5
6
7
8
9

```
1.(a) Value of x_median is:
 3.240094308985853
1.(b) Value of Wien's Displacement Constant,b is:
 0.004441631315457159 metres Kelvin
Hence, Wien's Displacement Law has been verified

Corresponding Wavelength is:  7.657985026650275e-07 metres
2. The value of integral I_p is:
 6.480189299740922
Value of RHS:
 6.493939402266828
Hence, I_p = pi**4/15 is shown

slope 3.9999999999999999
intercept -16.685940839766O5
Value of Stefan Boltzman law constant is:
 5.667453531149509e-08 Joules K^-4 m^-2 s^-1
```

EXPERIMENT 6:
CODE AND RESULTS

Gmail

Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com> Mon, Mar 13, 2023 at 11:46 PM
To: jain.sarthak38@gmail.com

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 6: To form the partition function of an ideal gas and obtain the thermodynamical variables using it

Code1 *

(about 10 lines)

```
import numpy as np
import matplotlib.pyplot as plt
k = 8.617 * 10**(-5)
T_low = np.linspace(10**(-18),5000,50)
T_high = np.linspace(5000,10**(5),50)
g = [1,1] ; epsilon = [0,1]
def partition_function(T,epsilon,g):
Z_list = []
for i in T:
Z = 0
for j,m in zip(epsilon,g):
Z = Z + m * np.exp(-j/(k*i))
Z_list.append(Z)
return np.array(Z_list)
Z_1 = partition_function(T_low,epsilon,g)
Z_2 = partition_function(T_high,epsilon,g)
```

Code2

(about 10 lines)

```
def fraction_population(g,T,epsilon,Z):
frac_pop_list = []
for j in range(len(epsilon)):
frac_pop = (g[j] * np.exp(-epsilon[j]/(k*T)))/Z
frac_pop_list.append(frac_pop)
frac_pop_list = np.array(frac_pop_list)
return frac_pop_list
frac_pop_low = fraction_population(g,T_low,epsilon,Z_1)
frac_pop_high = fraction_population(g,T_high,epsilon,Z_2)
```

```
def internal_energy(frac_pop,N,epsilon,T):
N_j = N * frac_pop
inte_energy = np.zeros([len(T)])
for i in range(len(N_j)):
inte_energy = inte_energy + N_j[i]*epsilon[i]
return inte_energy
U_low = internal_energy(frac_pop_low,1,epsilon,T_low)
U_high = internal_energy(frac_pop_high,1,epsilon,T_high)
```

Code3

```
def entropy(Z,N,T,U):
S = (N*k*np.log(Z/N)) + (U/T) + (N*k)
return S
S_low = entropy(Z_1,1,T_low,U_low)
S_high = entropy(Z_2,1,T_high,U_high)
def free_energy(N,T,Z):
F = -N*k*T*np.log(Z)
return F
F_low = free_energy(1,T_low,Z_1)
F_high = free_energy(1,T_high,Z_2)
```

Code4

(about 10 lines)

```
def graph(x1,x2,y1,y2,title,y_label,frac_pop_low,frac_pop_high,key):
fig1,ax1 = plt.subplots(1,2)
fig1.suptitle(title)
if key == 0:
ax1[0].scatter(x1,y1,label = "Low Temperature", c = 'green')
ax1[0].set_xlabel("T")
ax1[0].set_ylabel(y_label)
ax1[0].grid(ls = "--")
ax1[0].legend()
ax1[1].scatter(x2,y2,label = "High Temperature", c ='orange')
ax1[1].set_xlabel("T")
ax1[1].set_ylabel(y_label)
ax1[1].grid(ls = "--")
ax1[1].legend()
plt.show()
elif key == 1:
for i in range(len(frac_pop_low)):
ax1[0].scatter(x1,frac_pop_low[i],label = "Low Temperature")
ax1[1].scatter(x2,frac_pop_high[i],label = "High Temperature")
ax1[0].set_xlabel("T")
ax1[0].set_ylabel("$\\dfrac{N_i}{N}$")
ax1[1].set_xlabel("T")
```

```
ax1[1].set_ylabel("$\\dfrac{N_i}{N}$")
ax1[0].grid(ls = "--")
ax1[1].grid(ls = "--")
ax1[0].legend()
ax1[1].legend()
plt.show()
```

Code5

(about 10 lines)

```
graph(T_low,T_high,Z_1,Z_2,"Partition Function","Z",None,None,0)
graph(T_low,T_high,None,None,"Fraction Population",None,frac_pop_low,frac_pop_high,1)
graph(T_low,T_high,U_low,U_high,"Internal Energy","U",None,None,0)
graph(T_low,T_high,S_low,S_high,"Entropy","S",None,None,0)
graph(T_low,T_high,F_low,F_high,"Helmholtz free energy","F",None,None,0)
```

Code6

(about 10 lines)

Plot1

Submitted files

image1 - Sarthak Jain.png

Plot2

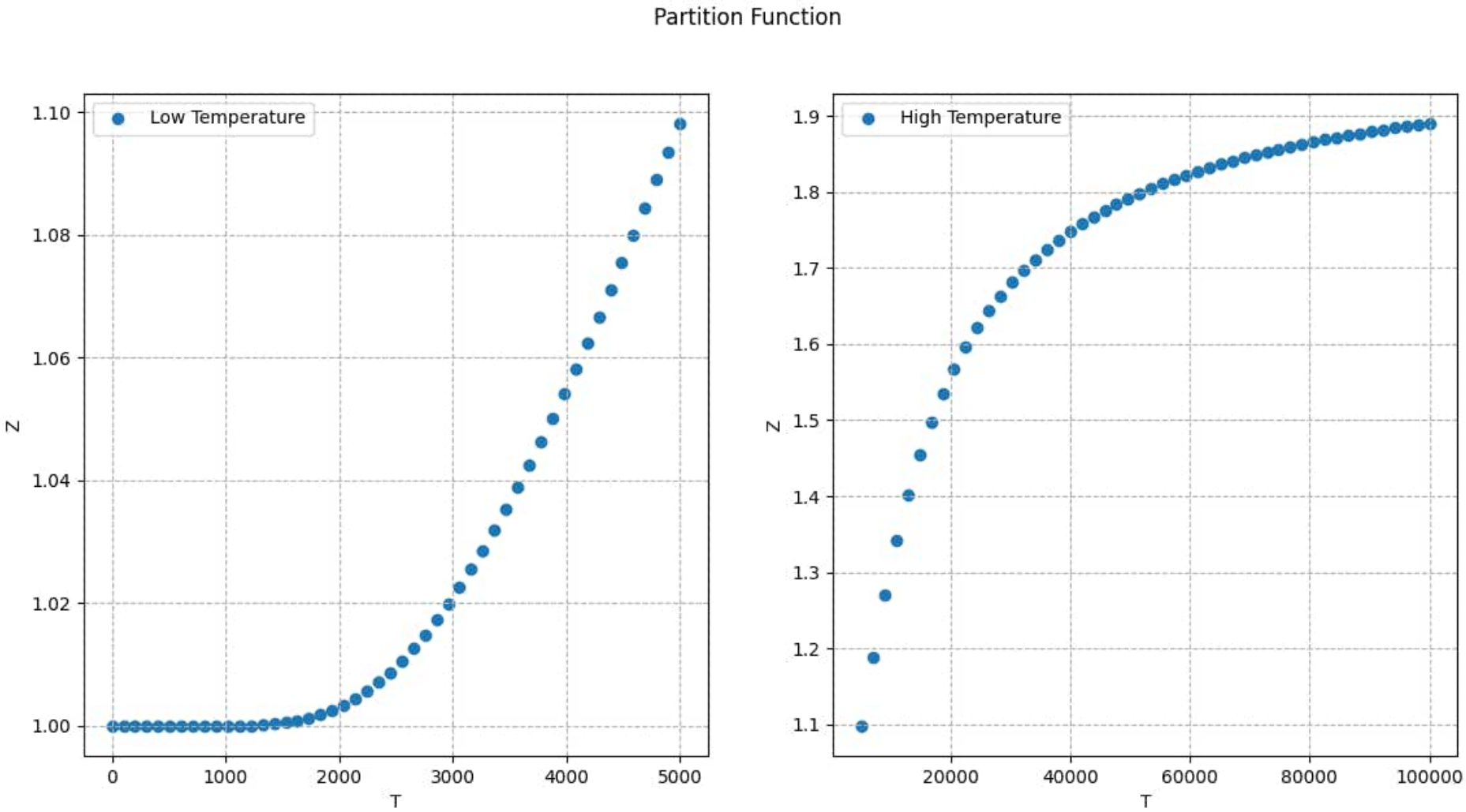
Submitted files

image2 - Sarthak Jain.png

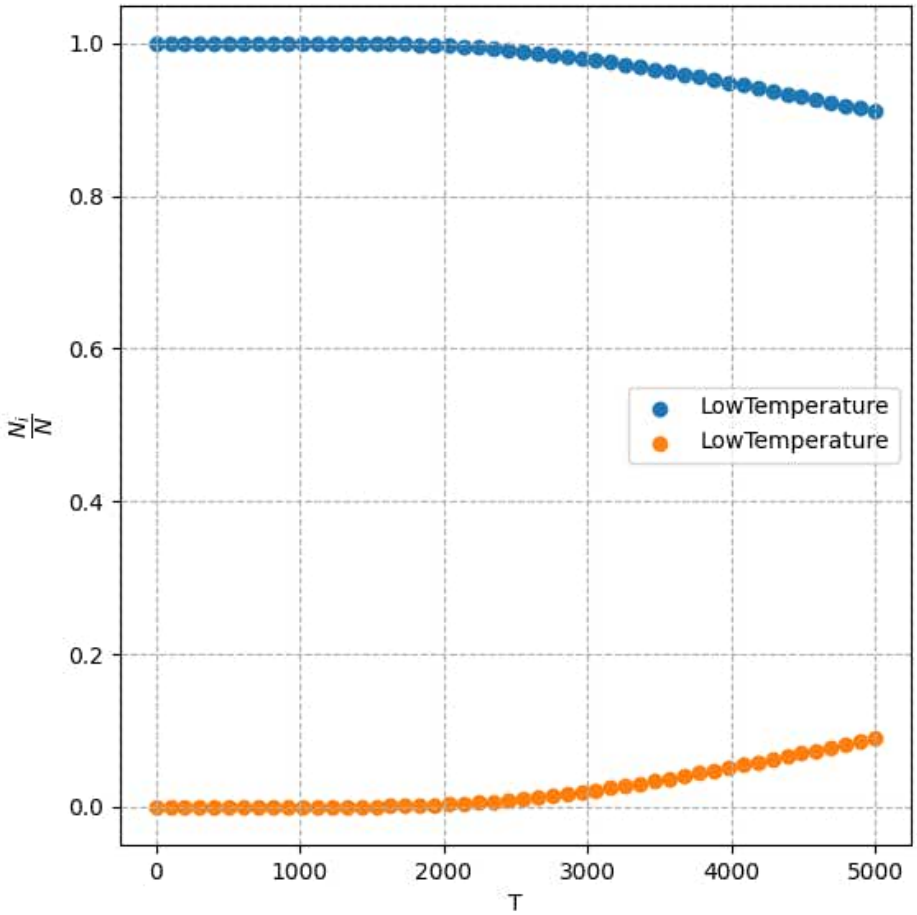
Plot3

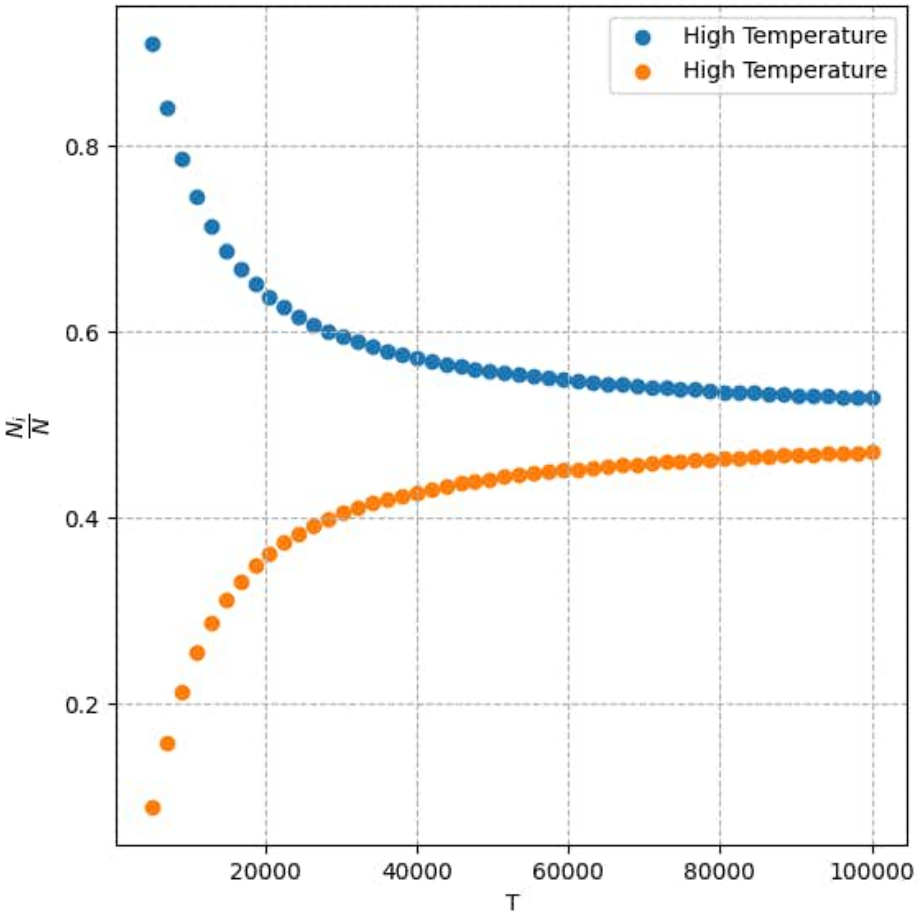
Submitted files

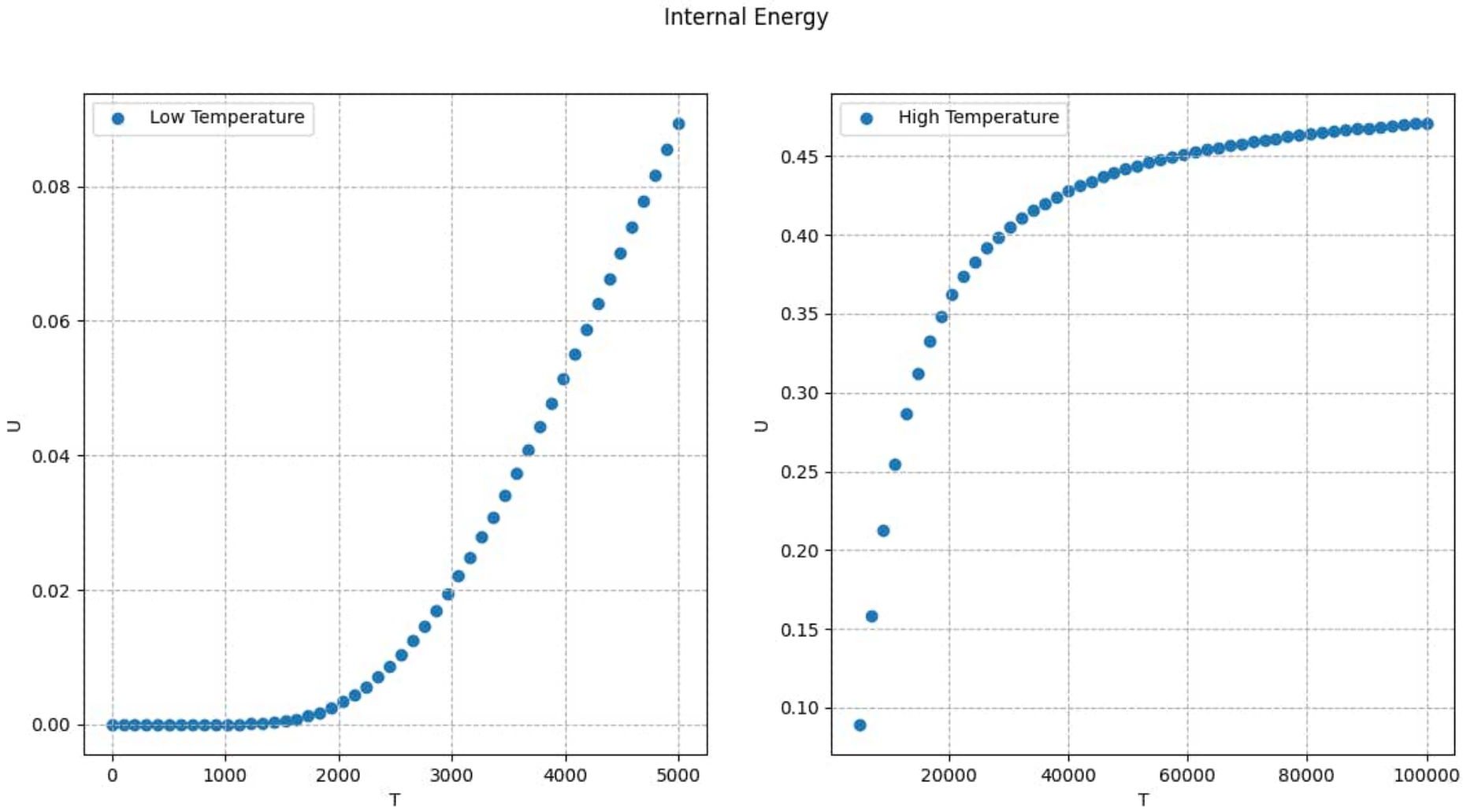
Figure_5 - Sarthak Jain.png

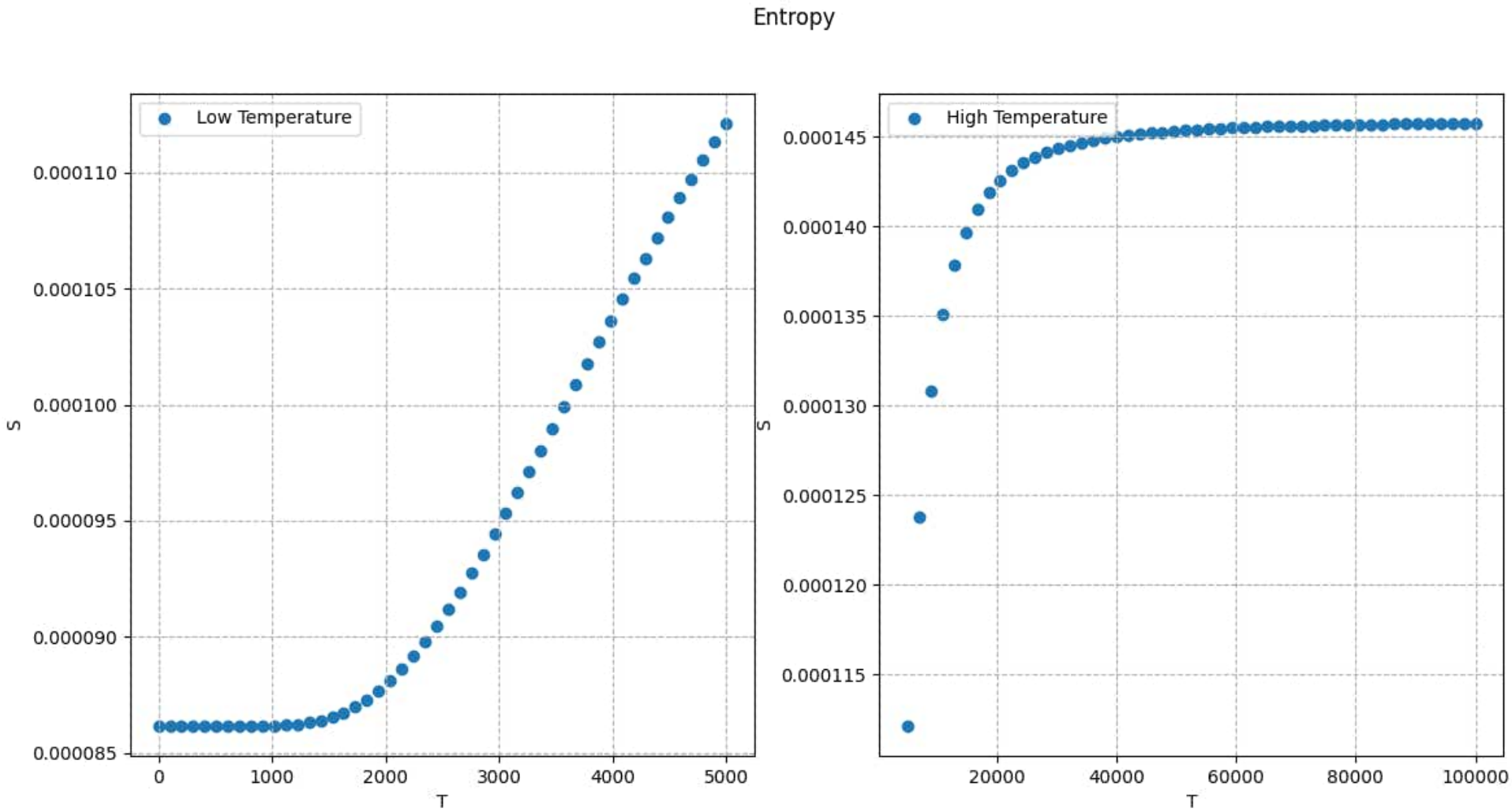

Partition Function
Low Temperature
High Temperature
Z
T
1.10
1.08
1.06
1.04
1.02
1.00
0
1000
2000
3000
4000
5000
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
20000
40000
60000
80000
100000

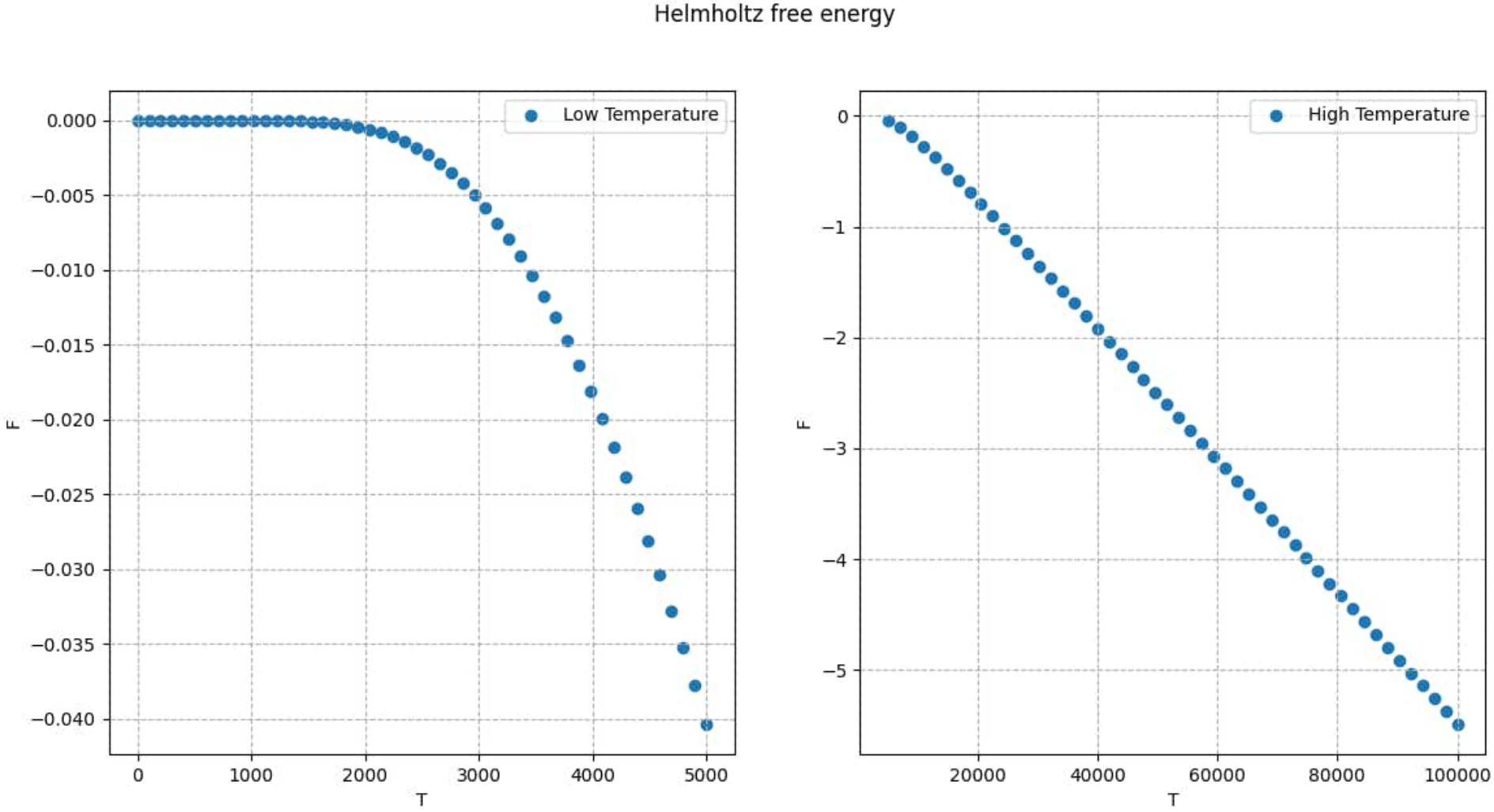
Fraction Population






Internal Energy
Low Temperature
High Temperature
U
T
0.00
0.02
0.04
0.06
0.08
0
1000
2000
3000
4000
5000
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
20000
40000
60000
80000
100000


Entropy
Low Temperature
High Temperature
S
T
0.000085
0.000090
0.000095
0.000100
0.000105
0.000110
0.000115
0.000120
0.000125
0.000130
0.000135
0.000140
0.000145
0
1000
2000
3000
4000
5000
20000
40000
60000
80000
100000


Helmholtz free energy
Low Temperature
High Temperature
F
T
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
−0.030
−0.035
−0.040
0
1000
2000
3000
4000
5000
−1
−2
−3
−4
−5
20000
40000
60000
80000
100000

EXPERIMENT 7:
CODE AND RESULTS

Gmail

Sarthak Jain <jain.sarthak38@gmail.com>

Computational Lab File

Google Forms <forms-receipts-noreply@google.com>
To: jain.sarthak38@gmail.com

Tue, Apr 18, 2023 at 8:59 PM

Thanks for filling in Computational Lab File

Here's what was received.

Computational Lab File

Email *

jain.sarthak38@gmail.com

Class *

Semester 6 BSc H Physics

Paper *

Statistical Mechanics

Name *

Sarthak Jain

Roll No. *

2020PHY1201

Aim *

Experiment 7: Canonical Ensemble - Maxwell Boltzmann (Ideal Gas)

Code1 *

(about 10 lines)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.integrate import quad
from scipy.stats import linregress
h = 6.63*10**(-34)
k = 1.38*10**(-23)
N = 6.023*10**(23)
m = 1.67*10**(-27)
V = np.linspace(20*10**(-3),50*10**(-3),10)
T = np.linspace(150,450,10)
#Partition Function
def Z_VT(n,V,T):
const = h**2/(8*m*(V**(2/3))*k*T)
return(np.pi/2)*(n**2)*np.exp((-const*(n**2)))
def inte(n_i,n_f,V,T):
I = quad(lambda n: Z_VT(n,V,T),n_i,n_f)
return I
part_fn = np.zeros((len(V),len(T)))
for i in range(len(V)):
for j in range(len(T)):
part_fn[i,j] = inte(0,10**(11),V[i],T[j])[0]
```

Code2

(about 10 lines)

```
part_fn_log = np.zeros((len(V),len(T)))
for i in range(len(V)):
for j in range(len(T)):
part_fn_log[i,j] = np.log(inte(0,10**(11),V[i],T[j]))[0]
```

```
plt.plot(np.log(V),part_fn_log[0,:])
plt.xlabel("Log V")
plt.ylabel("Log Z")
plt.title("Partition Function (at constant Temperature)")
plt.grid(ls="--")
plt.show()
plt.plot(np.log(T),part_fn_log[:,0])
plt.xlabel("Log T")
plt.ylabel("Log Z")
plt.title("Partition Function (at constant Volume)")
plt.grid(ls="--")
plt.show()
#Pressure
def pressure(V,T):
for i in range(len(T)):
Diff = ((part_fn_log[i,:][:-1]) - (part_fn_log[i,:][1:]))/(V[:-1]- V[1:])
p = N*k*T[i]*Diff
plt.plot(V[:-1],p,label='For T= '+str(T[i]))
#return p
```

Code3

```
pressure(V,T)
plt.legend()
plt.xlabel("Volume")
plt.ylabel("Pressure")
plt.title("Pressure (at constant Temperature)")
plt.grid(ls="--")
plt.show()
#Internal Energy
def av_energy(T):
E=[]
for i in range(len(V)):
Diff = ((part_fn_log[i,:][:-1]) - (part_fn_log[i,:][1:]))/(T[:-1]- T[1:])
e = k*T[:-1]**2 *Diff
E.append(e)
plt.legend()
plt.xlabel("Temperature")
plt.ylabel("Energy")
plt.title("Energy")
plt.grid(ls="--")
plt.plot(T[:-1],e,label='For V= '+str(V[i]))
plt.show()
return E
U = N*av_energy(T)[1]
plt.plot(T[:-1],U)
plt.xlabel("Temperature")
plt.ylabel("Energy")
plt.title("Internal Energy")
```

```
plt.grid(ls="--")
plt.show()
```

Code4

(about 10 lines)

```
#Specific Heat
slope = linregress(T[:-1],U/N)[0]
print('Specific Heat is:' ,slope)
#Entropy
def entropy(U,T):
for i in range(len(T)):
Diff = ((part_fn_log[i,:][:-1]) - np.log(N)) #(part_fn_log[i,:][:-1]) - (part_fn_log[i,:][1:]))/(V[:-1]-
V[1:])
S = (U/T[:-1]) + (N*k*(Diff + 1))
plt.plot(V[:-1],S,label='For T= '+str(T[i]))
#return p
entropy(U,T)
plt.legend()
plt.xlabel("Volume")
plt.ylabel("Entropy")
plt.title("Entropy(at constant Temperature)")
plt.grid(ls="--")
plt.show()
```

Code5

(about 10 lines)

Code6

(about 10 lines)

Plot1

Submitted files

image1 - Sarthak Jain.png

Plot2

Submitted files

image2 - Sarthak Jain.png

Plot3

Submitted files

Figure_4 - Sarthak Jain.png

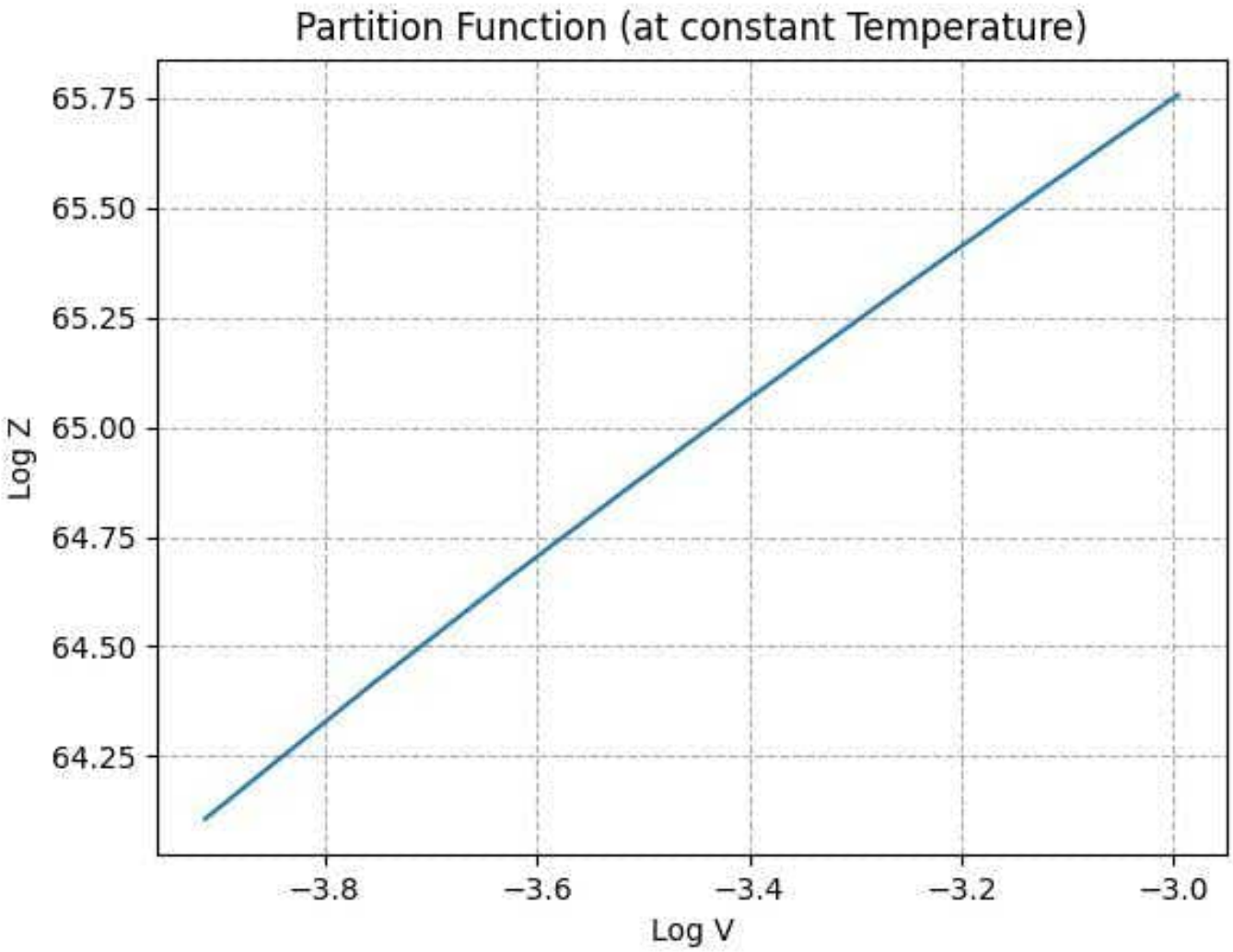
Plot4

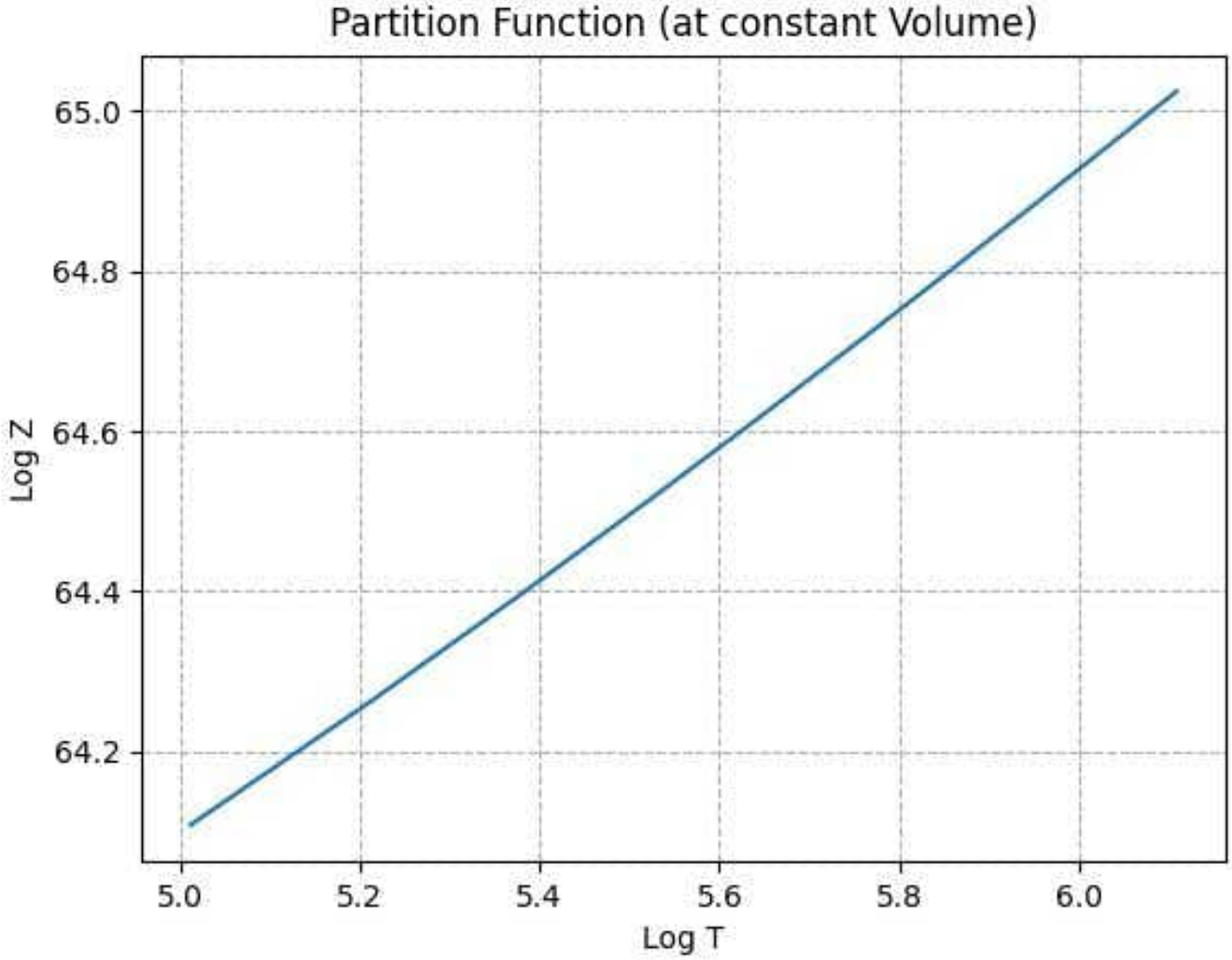
Submitted files

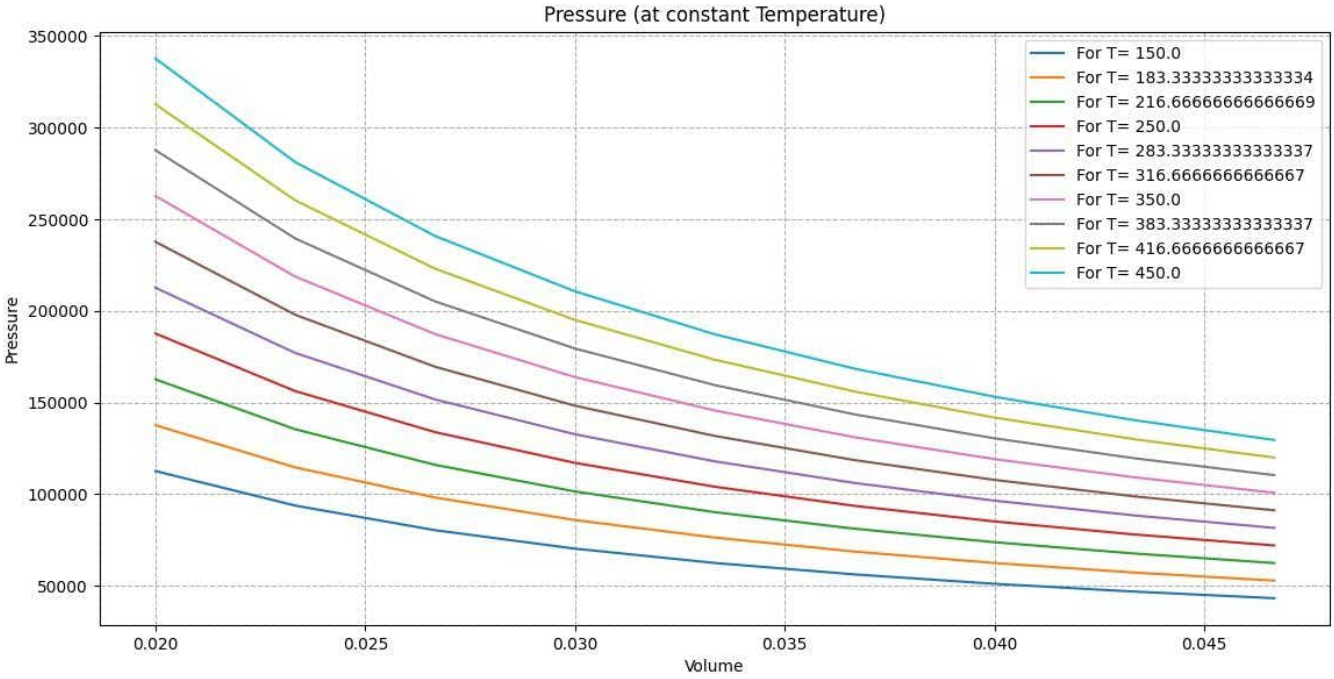
Figure_5 - Sarthak Jain.png

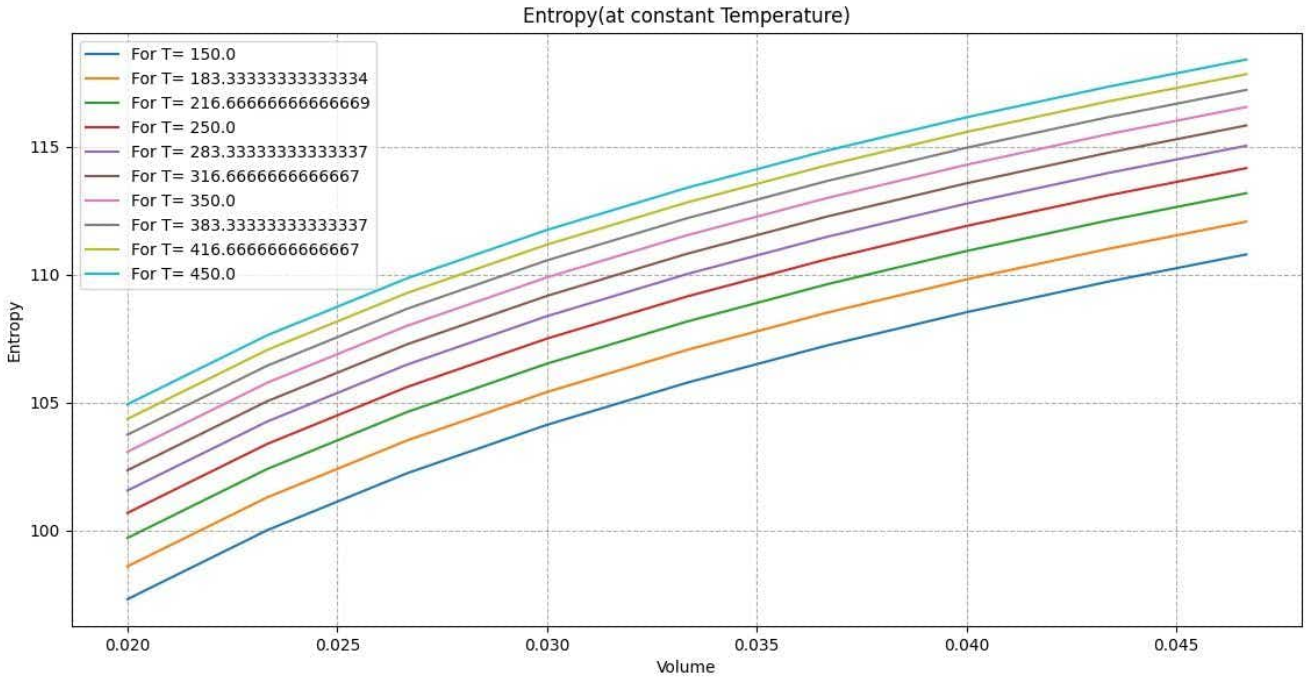
Comments

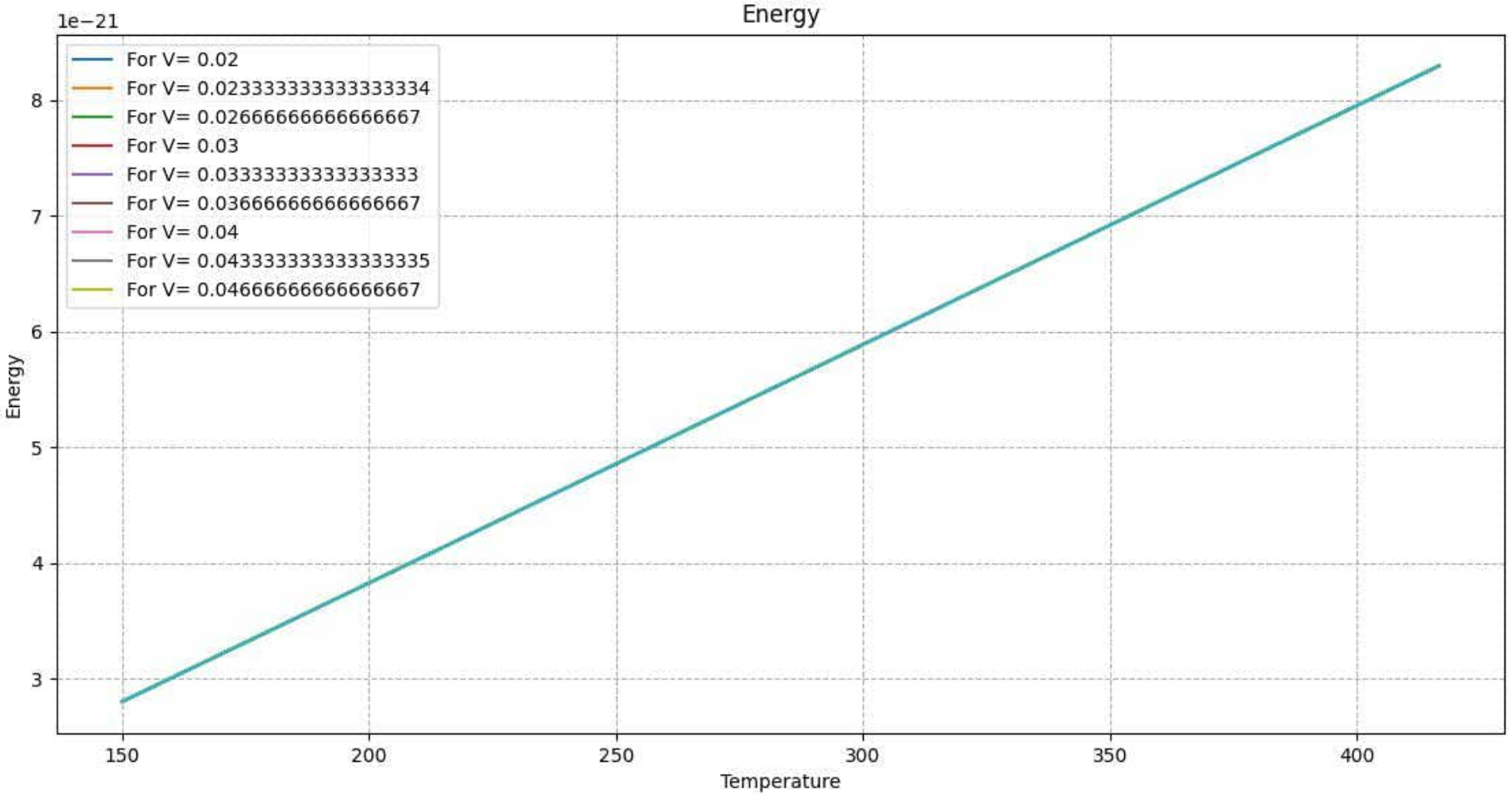
Value of Specific Heat is: 2.0605877338521136e-23

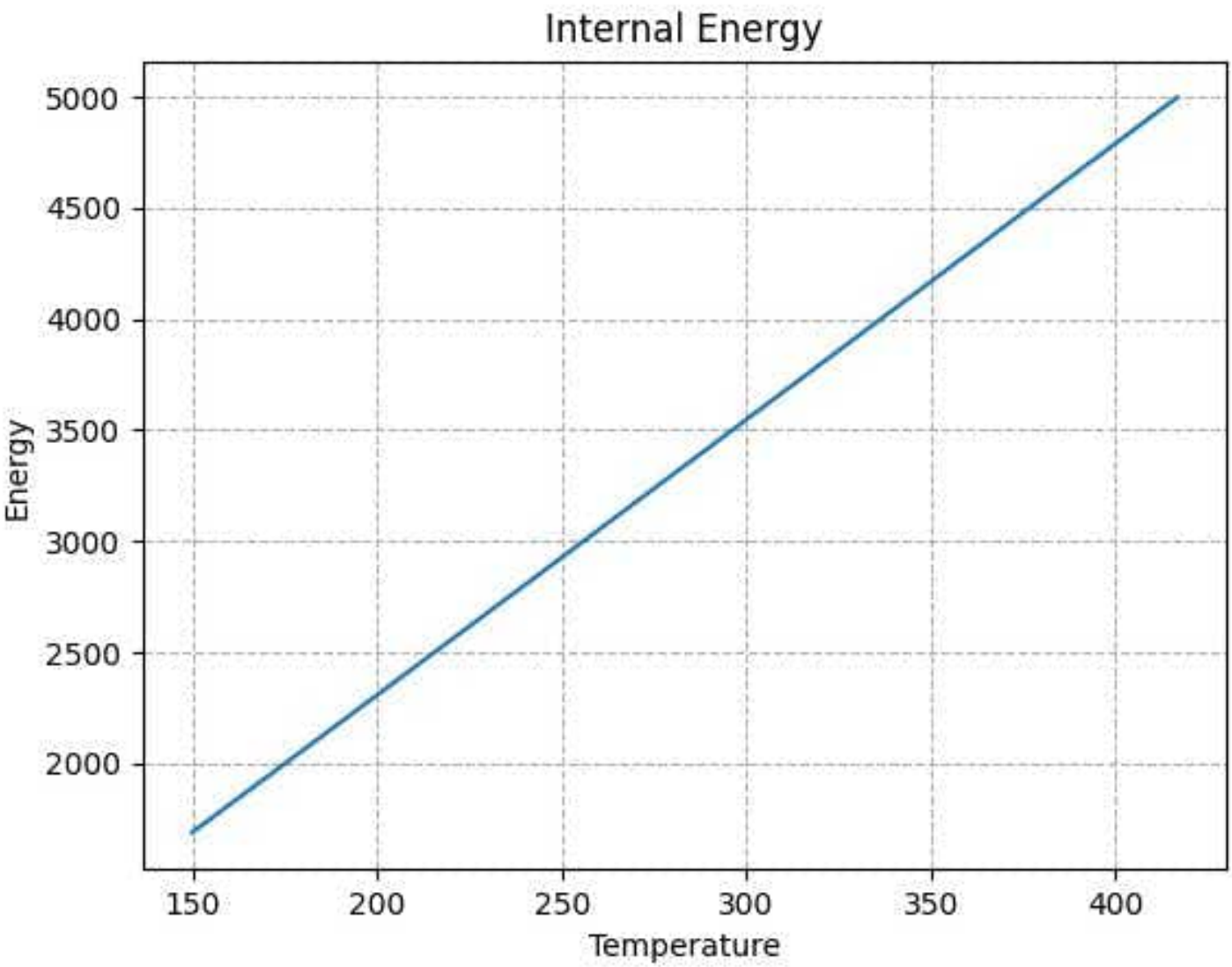

Partition Function (at constant Temperature)
65.75
65.50
65.25
65.00
64.75
64.50
64.25
Log Z
−3.8
−3.6
−3.4
−3.2
−3.0
Log V


Partition Function (at constant Volume)
65.0
64.8
64.6
64.4
64.2
Log Z
5.0
5.2
5.4
5.6
5.8
6.0
Log T


Pressure (at constant Temperature)
Pressure
Volume
350000
300000
250000
200000
150000
100000
50000
0.020
0.025
0.030
0.035
0.040
0.045
For T= 150.0
For T= 183.33333333333334
For T= 216.66666666666669
For T= 250.0
For T= 283.33333333333337
For T= 316.6666666666667
For T= 350.0
For T= 383.33333333333337
For T= 416.6666666666667
For T= 450.0


Entropy(at constant Temperature)
Entropy
Volume
115
110
105
100
0.020
0.025
0.030
0.035
0.040
0.045
For T= 150.0
For T= 183.33333333333334
For T= 216.66666666666669
For T= 250.0
For T= 283.33333333333337
For T= 316.6666666666667
For T= 350.0
For T= 383.33333333333337
For T= 416.6666666666667
For T= 450.0


Energy
1e−21
For V= 0.02
For V= 0.023333333333333334
For V= 0.02666666666666667
For V= 0.03
For V= 0.03333333333333333
For V= 0.03666666666666667
For V= 0.04
For V= 0.043333333333333335
For V= 0.04666666666666667
Energy
Temperature
8
7
6
5
4
3
150
200
250
300
350
400


Internal Energy
5000
4500
4000
3500
3000
2500
2000
Energy
150
200
250
300
350
400
Temperature