Student Details

Name : SATYAM TIWARI

Course : B.Sc. (Hons) Physics

College Roll Number : 1930205

Exam Roll Number : 19036567094

Section : B

Group : VI

Semester (Year) : 6th (3rd Year)

Subject : Statistical Mechanics Lab

Department : Physics

College : KMC, Delhi - 110007

Contents

Total Number of Practicals Done : 6

IDE used : Jupyter Notebook

GitHub Repo Link :

AIM :

Plot the probability of various macrostates in "Coin-Tossing Experiment" VS no of heads with 4,8,16 coins etc.

Theory :

1. In general, for 'n' number of coins, there will be 2^n number of Microstates.

2. No of microstates associated with the particular macrostate (say p heads) = nCp = (n!)/((p!)*(n-p)!)

Step-1 : Import necessary libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import comb
```

Step-2 : Take input of "no of coins flipped"

In [2]:
```
n = int(input("Enter the number of coins : "))
```

```
Enter the number of coins : 100
```

Step-3 : Calculate total number of microstates (using above mentioned formula)

In [3]:
```
nom = 2**n
print("Total number of microstates = ",nom)
```

```
Total number of microstates =  1267650600228229401496703205376
```

Step-4 : Find all the macrostates & their respective probabilities

In [4]:
```
#for 'n' coins, there will be 'n+1' macrostates
nh = np.arange(n+1)
print("Number of heads (macrostates) = ",nh)

ps = [] #Defining an empty array for probabilities
for j in nh:
    ns = comb(n,j) #ns = number of possible microstates for a particular macrostate
    psi = ns/nom   #psi = probability for a given macrostate
    ps.append(psi)
print("Respective Probabilities = ",ps)
```

```
Number of heads (macrostates) =  [  0   1   2   3   4   5   6   7   8   9  10  11  12
13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
```

```
  90  91  92  93  94  95  96  97  98  99 100]
Respective Probabilities =  [7.888609052210118e-31, 7.888609052210118e-29, 3.9048614808
440084e-27, 1.275588083742376e-25, 3.093301103075262e-24, 5.939138117904503e-23, 9.4036
35353348797e-22, 1.2627738903068384e-20, 1.4679746474816996e-19, 1.5005963063146263e-1
8, 1.3655426387463099e-17, 1.1172621589742536e-16, 8.286361012392381e-16, 5.60922899300
4073e-15, 3.4857351599382454e-14, 1.998488158364594e-13, 1.0616968341311906e-12, 5.2460
31415707059e-12, 2.4190033750204773e-11, 1.0439909302719954e-10, 4.2281632676015815e-1
0, 1.6107288638482216e-09, 5.78398092018225e-09, 1.9615239642357197e-08, 6.293222718589
6e-08, 1.9131397064512386e-07, 5.518672230147804e-07, 1.5125249815960647e-06, 3.9433687
020183116e-06, 9.790432639493739e-06, 2.3170690580135184e-05, 5.232091421320847e-05, 0.
00011281697127223077, 0.00023247133474277857, 0.0004581052772872401 4, 0.000863855665741
6528, 0.001559739396477984 2, 0.0026979276047186754, 0.00447287997624412, 0.007110732269
92655, 0.010843866711637987, 0.015869073236543397, 0.022292269546572867, 0.030068642644
214563, 0.03895255978909614, 0.048474296626430755, 0.05795839814029764, 0.0665904999909
8027, 0.07352701040670738, 0.07802866410507722, 0.07958923738717877, 0.0780286641050772
2, 0.07352701040670738, 0.06659049999098027, 0.05795839814029764, 0.048474296626430755,
0.03895255978909614, 0.030068642644214563, 0.022292269546572867, 0.015869073236543397,
0.010843866711637987, 0.00711073226992655, 0.00447287997624412, 0.0026979276047186754,
0.001559739396477984 2, 0.0008638556657416528, 0.0004581052772872401 4, 0.000232471334742
77857, 0.00011281697127223077, 5.232091421320847e-05, 2.3170690580135184e-05, 9.7904326
39493739e-06, 3.9433687020183116e-06, 1.5125249815960647e-06, 5.518672230147804e-07, 1.
9131397064512386e-07, 6.2932227185896e-08, 1.9615239642357197e-08, 5.78398092018225e-0
9, 1.6107288638482216e-09, 4.2281632676015815e-10, 1.0439909302719954e-10, 2.4190033750
204773e-11, 5.246031415707059e-12, 1.0616968341311906e-12, 1.998488158364594e-13, 3.485
7351599382454e-14, 5.609228993004073e-15, 8.286361012392381e-16, 1.1172621589742536e-1
6, 1.3655426387463099e-17, 1.5005963063146263e-18, 1.4679746474816996e-19, 1.2627738903
068384e-20, 9.403635353348797e-22, 5.939138117904503e-23, 3.093301103075262e-24, 1.2755
```

Step-5 : Print Table (for visual aid)

In [5]:

```
print("n(h)   P(h)")
print("-----------")
for i in range(n+1):
    print(nh[i],"   ",ps[i])
```

```
n(h)   P(h)
-----------
0     7.888609052210118e-31
1     7.888609052210118e-29
2     3.9048614808440084e-27
3     1.275588083742376e-25
4     3.093301103075262e-24
5     5.939138117904503e-23
6     9.403635353348797e-22
7     1.2627738903068384e-20
8     1.4679746474816996e-19
9     1.5005963063146263e-18
10     1.3655426387463099e-17
11     1.1172621589742536e-16
12     8.286361012392381e-16
13     5.609228993004073e-15
14     3.4857351599382454e-14
15     1.998488158364594e-13
16     1.0616968341311906e-12
17     5.246031415707059e-12
18     2.4190033750204773e-11
19     1.0439909302719954e-10
20     4.2281632676015815e-10
21     1.6107288638482216e-09
22     5.78398092018225e-09
23     1.9615239642357197e-08
24     6.2932227185896e-08
25     1.9131397064512386e-07
26     5.518672230147804e-07
27     1.5125249815960647e-06
28     3.9433687020183116e-06
29     9.790432639493739e-06
```

30 2.3170690580135184e-05
31 5.232091421320847e-05
32 0.00011281697127223077
33 0.00023247133474277857
34 0.0004581052772872401 4
35 0.0008638556657416528
36 0.0015597393964779842
37 0.0026979276047186754
38 0.00447287997624412
39 0.0071107322699265 5
40 0.010843866711637987
41 0.015869073236543397
42 0.022292269546572867
43 0.030068642644214563
44 0.03895255978909614
45 0.048474296626430755
46 0.05795839814029764
47 0.06659049999098027
48 0.07352701040670738
49 0.07802866410507722
50 0.07958923738717877
51 0.07802866410507722
52 0.07352701040670738
53 0.06659049999098027
54 0.05795839814029764
55 0.048474296626430755
56 0.03895255978909614
57 0.030068642644214563
58 0.022292269546572867
59 0.015869073236543397
60 0.010843866711637987
61 0.0071107322699265 5
62 0.00447287997624412
63 0.0026979276047186754
64 0.0015597393964779842
65 0.0008638556657416528
66 0.0004581052772872401 4
67 0.00023247133474277857
68 0.00011281697127223077
69 5.232091421320847e-05
70 2.3170690580135184e-05
71 9.790432639493739e-06
72 3.9433687020183116e-06
73 1.5125249815960647e-06
74 5.518672230147804e-07
75 1.9131397064512386e-07
76 6.2932227185896e-08
77 1.9615239642357197e-08
78 5.78398092018225e-09
79 1.6107288638482216e-09
80 4.2281632676015815e-10
81 1.0439909302719954e-10
82 2.4190033750204773e-11
83 5.246031415707059e-12
84 1.0616968341311906e-12
85 1.998488158364594e-13
86 3.4857351599382454e-14
87 5.609228993004073e-15
88 8.286361012392381e-16
89 1.1172621589742536e-16
90 1.3655426387463099e-17
91 1.5005963063146263e-18
92 1.4679746474816996e-19
93 1.2627738903068384e-20
94 9.403635353348797e-22
95 5.939138117904503e-23
96 3.093301103075262e-24
97 1.275588083742376e-25
98 3.9048614808440084e-27

```
99     7.888609052210118e-29
100     7.888609052210118e-31
```

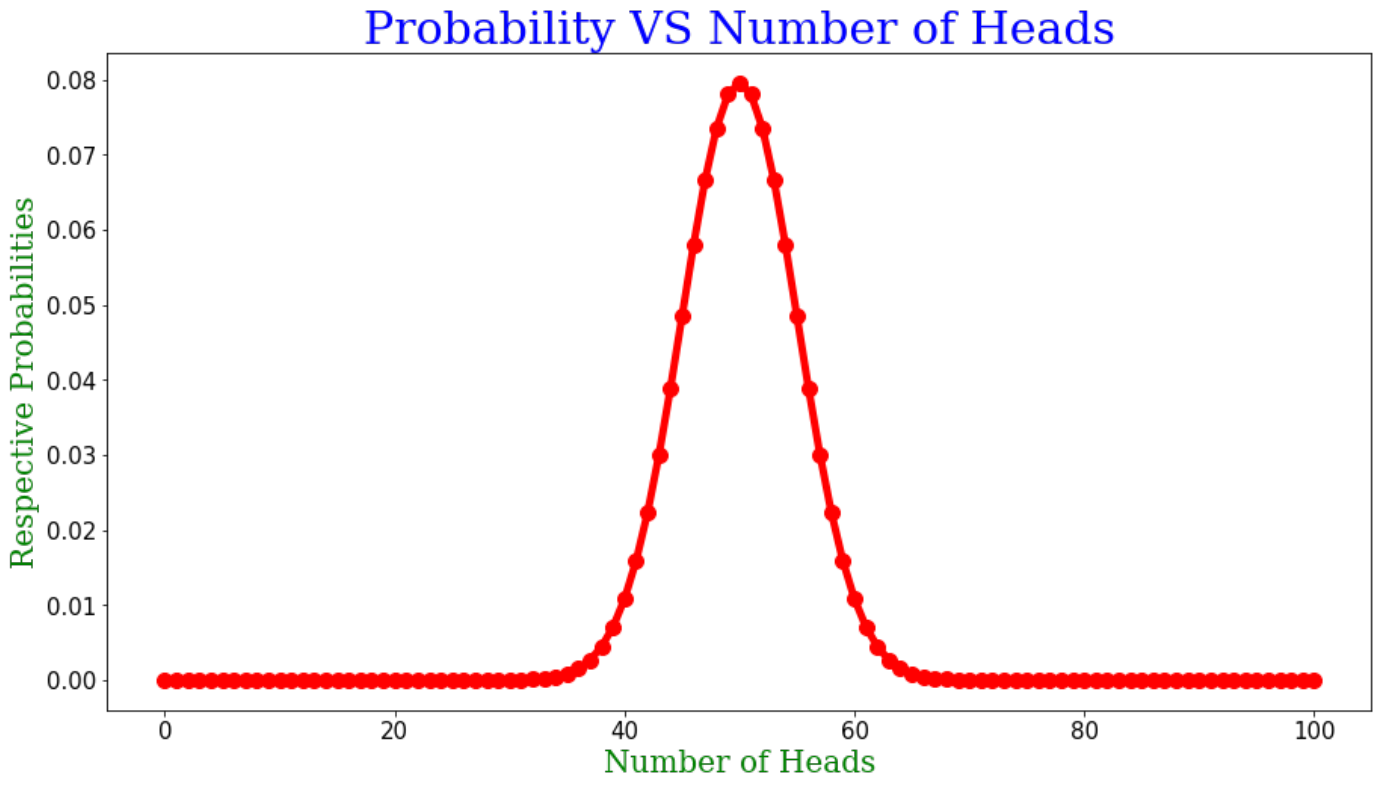
Step-6 : Plot the required graph

In [6]:

```
plt.figure(figsize=(15,8))
fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

plt.plot(nh,ps,"o-r",lw="5",ms="10")
plt.xlabel("Number of Heads",color="green",fontdict=fontji)
plt.ylabel("Respective Probabilities",color="green",fontdict=fontji)
plt.title("Probability VS Number of Heads",color="blue",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
```

Out[6]:

```
(array([-0.01,  0.  ,  0.01,  0.02,  0.03,  0.04,  0.05,  0.06,  0.07,
         0.08,  0.09]),
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



AIM:

1. Plot Maxwell Speed Distributions at different temperatures in a 3-d System.

2. Calculate :

(a) Most Probable Speed

(b) Average Speed

(c) RMS Speed

Step-1 : Import necessary libraries

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Step-2 : Define required constants

In [20]:
```
k = 1.38e-23 #Boltzmann Constant
N = 6.022e23 #Avagadro Number
```

Step-3 : Take necessary inputs from the user & find molecular mass (in kg) from given data

In [42]:
```
name = input("Enter the name of the gas : ")
M = float(input("Enter the molar mass of given gas in g/mol : "))
m = M/(N*1000) #Molecular Mass in kg (for 1 molecule)
```

```
Enter the name of the gas : Oxygen
Enter the molar mass of given gas in g/mol : 16
```

Step-4 : Define the range for velocity

In [9]:
```
v = np.arange(0,2000)
```

The distribution function for speed of particles in an ideal gas at temperature T is given by

$$f(v) = 4\pi \left(\frac{m}{2\pi kT}\right)^{3/2} v^2 e^{-mv^2/2kT}$$

m is the mass of gas molecules
k is the Boltzmann constant
T is the absolute temperature
v represents the speed of gas molecules

$$\text{Let } a = \frac{m}{2kT}$$

$$f(v) = 4\pi \left(\frac{a}{\pi}\right)^{3/2} v^2 e^{-av^2}$$

Step-5 : Using above formula, formulate a function for MSD function

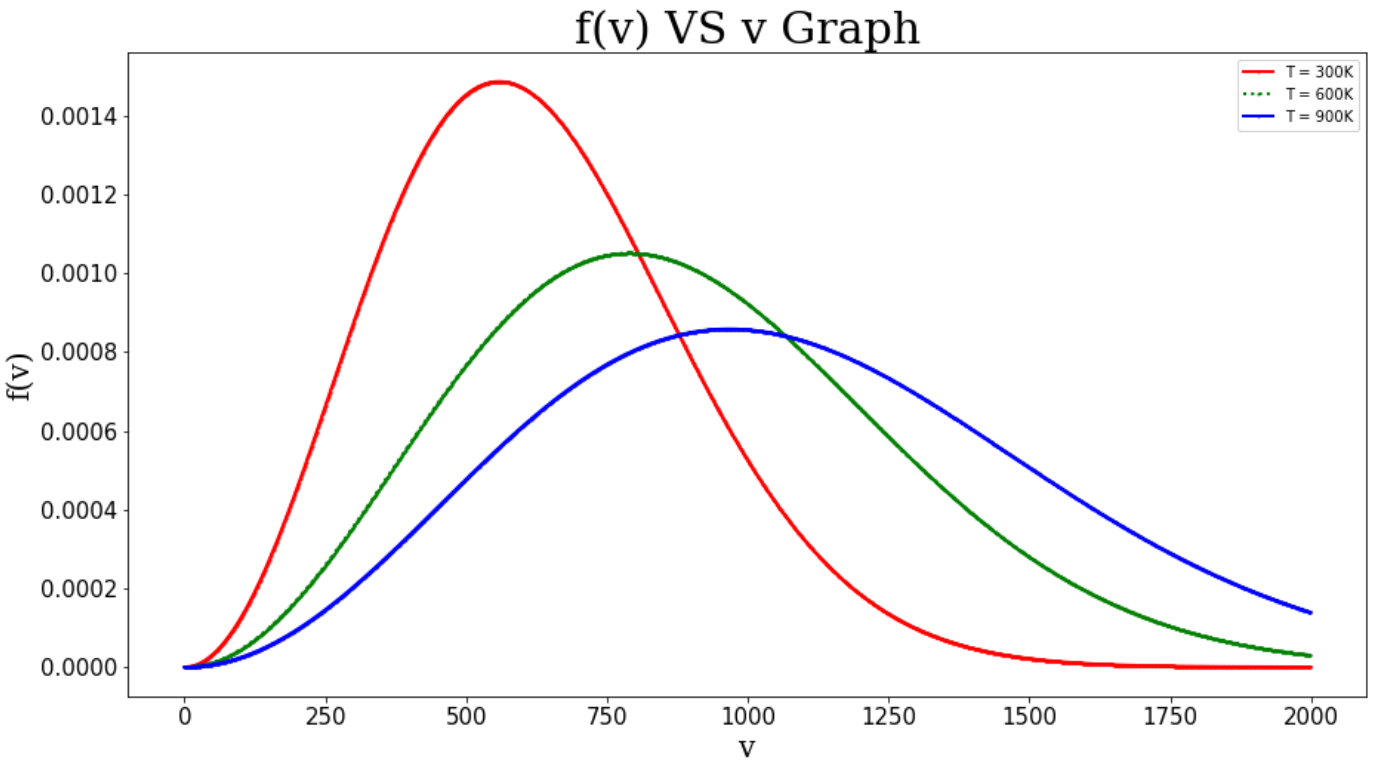
In [22]:
```
def f(v,T):
    a = m/(2*k*T)
    return 4*(np.pi)*((a/np.pi)**(3/2))*(v**2)*(np.exp(-a*(v**2)))
```

Step-6 : Using above created function, evaulate MBD function at 3 different temperatures

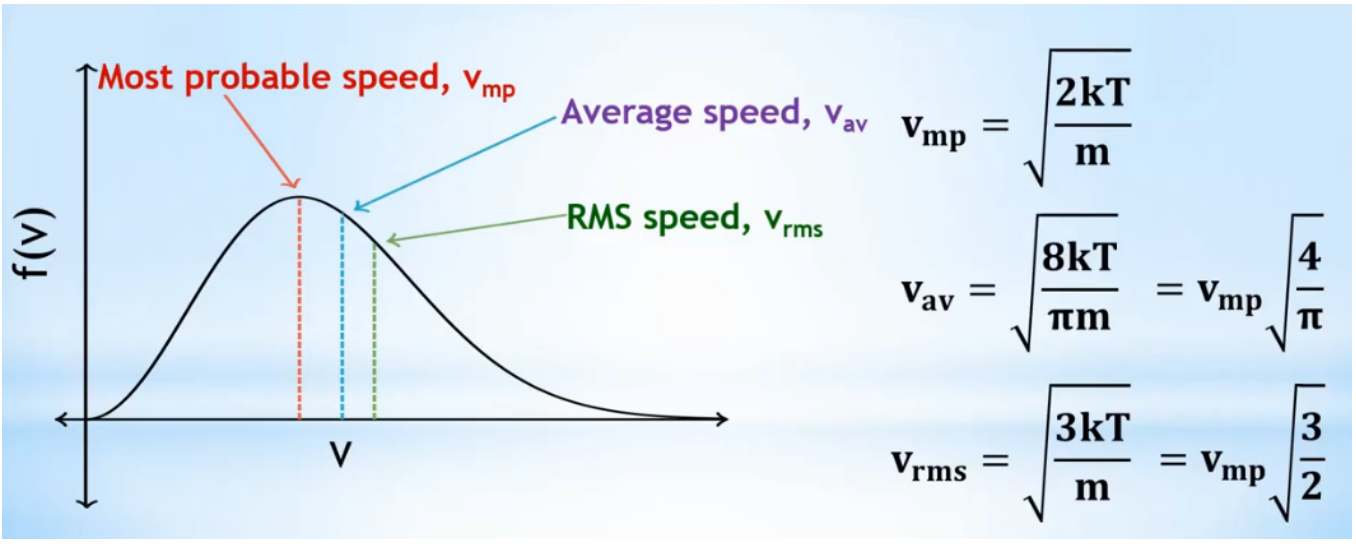
In [43]:
```
#Let us evaulate function at 3 different temperatures : Ta=300K ; Tb=600K ; Tc=900K

Ta = 300 #in Kelvin
fa = f(v,Ta)

Tb = 600 #in Kelvin
fb = f(v,Tb)

Tc = 900 #in Kelvin
fc = f(v,Tc)
```

Step-7 : Plot the MBD function at given temperatures

In [44]:
```
plt.figure(figsize=(15,8)) #Setting size of the figure
fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

plt.plot(v,fa,"o-r",lw="2",ms="1",label="T = 300K")
plt.plot(v,fb,"o:g",lw="2",ms="1",label="T = 600K")
plt.plot(v,fc,"o-b",lw="2",ms="1",label="T = 900K")
plt.legend(loc="best")
plt.xlabel("v",fontdict=fontji)
plt.ylabel("f(v)",fontdict=fontji)
plt.title("f(v) VS v Graph",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



Step-8 : Evaluate Most Probable, Average & RMS Speed



In [45]:

```
#All velocities are in m/s
a = np.where(fa == fa.max())
Vmp1 = v[a]
Vav1 = Vmp1 * np.sqrt(4/np.pi)
Vrms1 = Vmp1 * np.sqrt(3/2)
print("At T = 300K")
print("Most Probable Speed = ",Vmp1)
print("Average Speed = ",Vav1)
print("RMS Speed = ",Vrms1)

b = np.where(fb == fb.max())
Vmp2 = v[b]
Vav2 = Vmp2 * np.sqrt(4/np.pi)
Vrms2 = Vmp2 * np.sqrt(3/2)
print("\nAt T = 600K")
print("Most Probable Speed = ",Vmp2)
print("Average Speed = ",Vav2)
print("RMS Speed = ",Vrms2)

c = np.where(fc == fc.max())
Vmp3 = v[c]
Vav3 = Vmp3 * np.sqrt(4/np.pi)
Vrms3 = Vmp3 * np.sqrt(3/2)
print("\nAt T = 900K")
print("Most Probable Speed = ",Vmp3)
print("Average Speed = ",Vav3)
print("RMS Speed = ",Vrms3)
```

```
At T = 300K
Most Probable Speed =  [558]
Average Speed =  [629.63557524]
RMS Speed =  [683.40763824]

At T = 600K
Most Probable Speed =  [789]
Average Speed =  [890.29116284]
RMS Speed =  [966.32370353]

At T = 900K
Most Probable Speed =  [967]
Average Speed =  [1091.14265458]
RMS Speed =  [1184.32829064]
```

AIM:

Plot Specific Heat of Solids w.r.t. Temperature:

(a) Dulong-Petit Law

(b) Einstein Distribution Function

(c) Debye Distribution Function

Step-1 : Import necessary libraries

In [24]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
```

Step-2 : Define required constants

In [10]:
```
k = 1.38e-23 #Boltzmann Constant
N = 6.022e+23 #Avagadro Number
```

Step-3 : Take required Inputs from the User

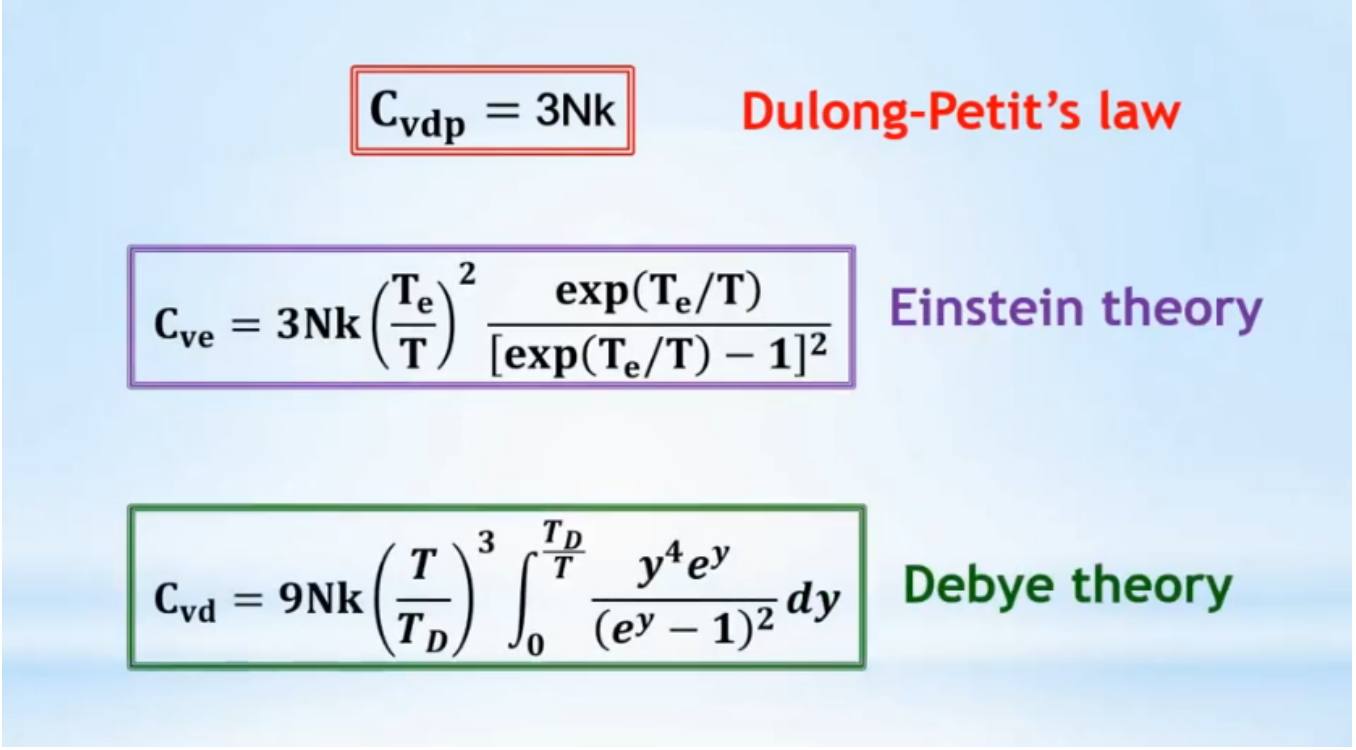
In [11]:
```
name = input("Enter the name of the Solid : ")
Te = float(input("Enter the value of Einstein Temperature in Kelvin : "))
Td = float(input("Enter the value of Debye Temperature in Kelvin : "))
```

```
Enter the name of the Solid : Cu
Enter the value of Einstein Temperature in Kelvin : 100
Enter the value of Debye Temperature in Kelvin : 100
```

Step-4 : Define Temperature range

In [12]:
```
T = np.arange(1,2*Td) #Temperature Range in Kelvin
```

Step-5 : Using for loop, define lists (for all models) for Cv at different temperatures



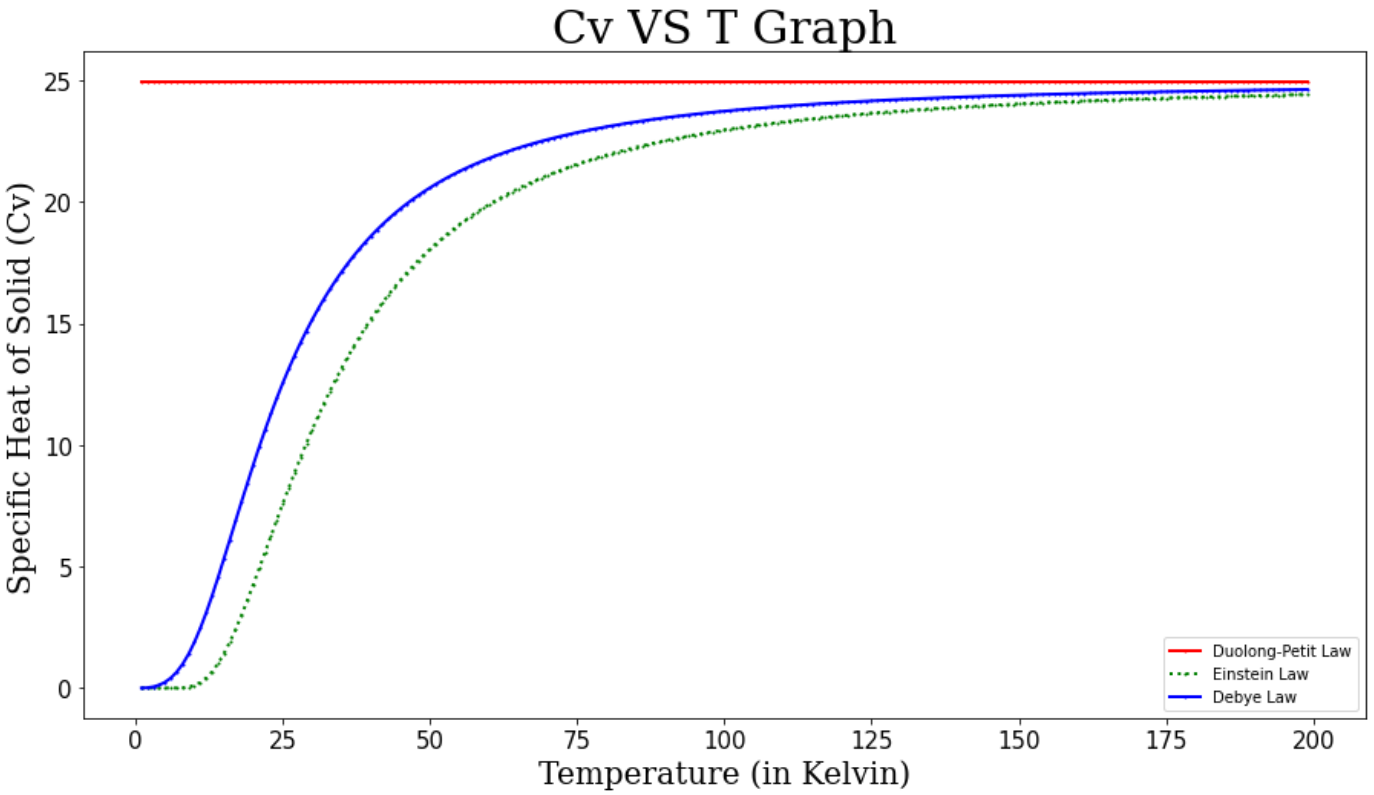
In [22]:

```
Cvdp = np.full(len(T),3*N*k) #Duolong-Petit Law

Cve = 3*N*k*((Te/T)**2)*np.exp(Te/T)/((np.exp(Te/T)-1)**2) #Einstein Law

Cvd = [] #Creating Empty list for Cv values obtained by Debye Theory
for i in range(len(T)):
    fn = lambda y: y**4 * np.exp(y) / (np.exp(y) - 1)**2
    Cvdl = quad(fn,0,Td/T[i])[0]
    Cvdl = Cvdl*9*N*k*((T[i]/Td)**3)
    Cvd.append(Cvdl) #Debye Law
```

Step-6 : Plotting various models

In [23]:

```
plt.figure(figsize=(15,8)) #Setting size of the figure
fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

plt.plot(T,Cvdp,"o-r",lw="2",ms="1",label="Duolong-Petit Law")
plt.plot(T,Cve,"o:g",lw="2",ms="1",label="Einstein Law")
plt.plot(T,Cvd,"o-b",lw="2",ms="1",label="Debye Law")
plt.legend(loc="best")
plt.xlabel("Temperature (in Kelvin)",fontdict=fontji)
plt.ylabel("Specific Heat of Solid (Cv)",fontdict=fontji)
plt.title("Cv VS T Graph",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```


Cv VS T Graph
Specific Heat of Solid (Cv)
Temperature (in Kelvin)
0
5
10
15
20
25
0
25
50
75
100
125
150
175
200
Duolong-Petit Law
Einstein Law
Debye Law

AIM:

Plot the following functions with energy at different temperatures.

1. Maxwell-Boltzmann Distribution

2. Fermi-Dirac Distribution

3. Bose-Einstein Distribution

Formula Required :

The general formula for the three distributions is:

> f(E) = 1 / (exp((E-u)/kT) + a)

```
where  f(E) = Probability that the given system has energy E
       E = Energy
       u = Chemical potential
       k = Boltzmann constant
       T = Temperature
```

Values of a for different distributions are:

- Maxwell-Boltzmann distribution : **a = 0**
- Bose-Einstein distribution : **a = -1**
- Fermi-Dirac distribution : **a = +1**

Step-1 : Import necessary libraries (numpy & matplotlib.pyplot)

In [43]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Step-2 : Define an array for Energy (E), with sufficient range

In [44]:
```
E = np.linspace(-0.5,0.5,1001)
```

Step-3 : Define constant values (e,k,u)

In [45]:
```
e = 1.6e-19 #Electric Charge (in Couloumb)
k = 1.38e-23 #Boltzmann Constant (in Joule per Kelvin)
u = 0 #Chemical Potential (Let it be zero)
```

Step-4 : Define a function Fn with parameters T & a, which returns the general fomula result

In [46]:

```
def Fn(T,a):
    return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system ir
```

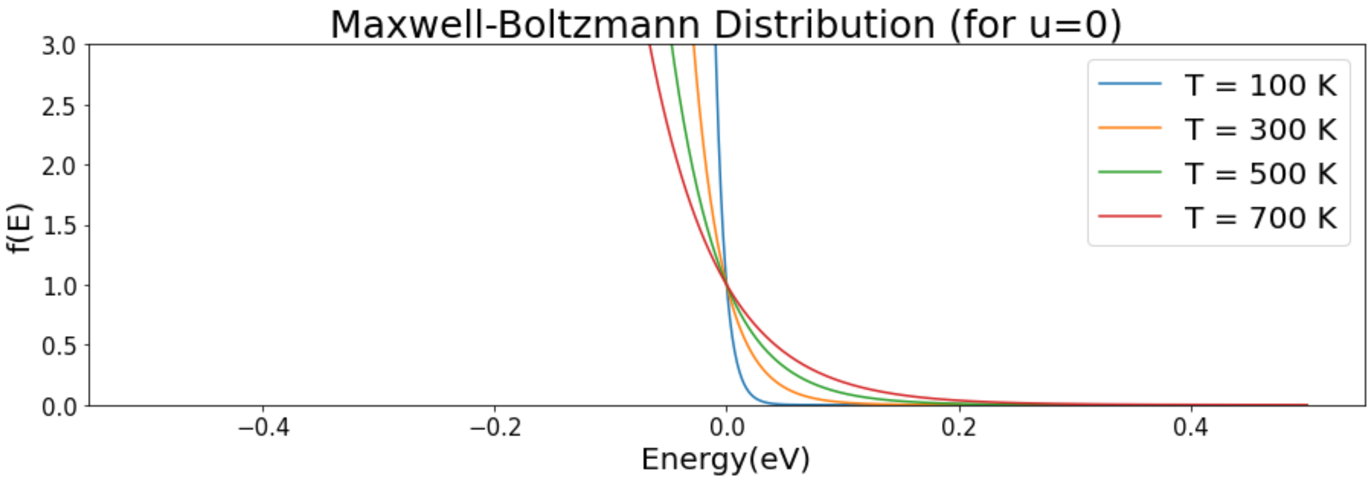
Step-5 : Plotting required graphs

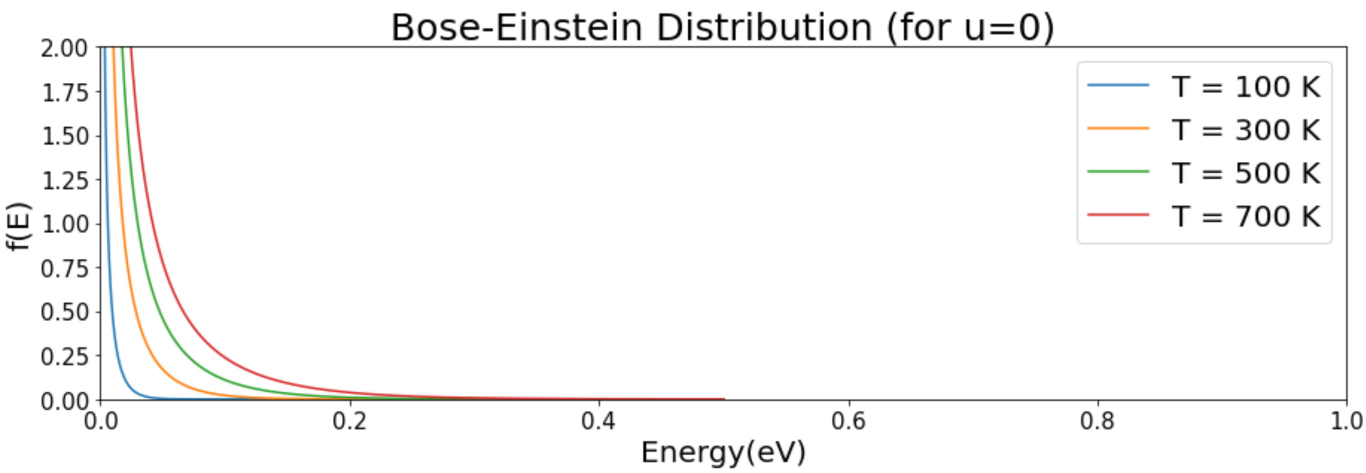
In [47]:

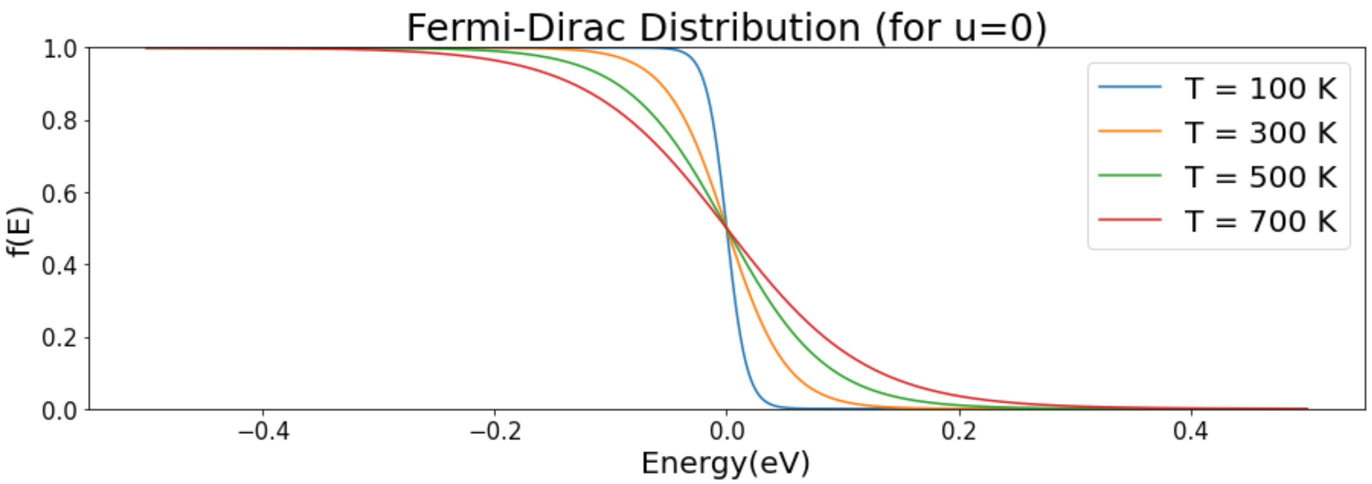
```
#Defining the Super Title of all the 4 graphs
#plt.suptitle("Plot of the following functions at different temperatures",size = 20,col

#1. Maxwell-Boltzmann Distribution
plt.figure(figsize=(15,20))
plt.subplot(4,1,1)
plt.plot(E,Fn(100,0),label='T = 100 K')
plt.plot(E,Fn(300,0),label='T = 300 K')
plt.plot(E,Fn(500,0),label='T = 500 K')
plt.plot(E,Fn(700,0),label='T = 700 K')
plt.ylim(0,3)
plt.xlabel('Energy(eV)',fontsize=20)
plt.ylabel('f(E)',fontsize=20)
plt.legend(loc='best',prop={'size':20})
plt.title("Maxwell-Boltzmann Distribution (for u=0)",fontsize=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

#2. Bose-Einstein Distribution
plt.figure(figsize=(15,20))
plt.subplot(4,1,2)
plt.plot(E,Fn(100,-1),label='T = 100 K')
plt.plot(E,Fn(300,-1),label='T = 300 K')
plt.plot(E,Fn(500,-1),label='T = 500 K')
plt.plot(E,Fn(700,-1),label='T = 700 K')
plt.xlim(0,1)
plt.ylim(0,2)
plt.xlabel('Energy(eV)',fontsize=20)
plt.ylabel('f(E)',fontsize=20)
plt.legend(loc='best',prop={'size':20})
plt.title("Bose-Einstein Distribution (for u=0)",fontsize=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

#3. Fermi-Dirac Distribution
plt.figure(figsize=(15,20))
plt.subplot(4,1,3)
plt.plot(E,Fn(100,+1),label='T = 100 K')
plt.plot(E,Fn(300,+1),label='T = 300 K')
plt.plot(E,Fn(500,+1),label='T = 500 K')
plt.plot(E,Fn(700,+1),label='T = 700 K')
plt.legend(loc='best',prop={'size':20})
plt.ylim(0,1)
plt.xlabel('Energy(eV)',fontsize=20)
plt.ylabel('f(E)',fontsize=20)
plt.title("Fermi-Dirac Distribution (for u=0)",fontsize=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

#4. Comparing all 3 distributions at specific temperature (T=500K)
plt.figure(figsize=(15,20))
plt.subplot(4,1,4)
plt.plot(E,Fn(500,0),label='M-B distribution')
plt.plot(E,Fn(500,-1),label='B-E distribution')
plt.plot(E,Fn(500,+1),label='F-D distribution')
plt.legend(loc='best',prop={'size':20})
plt.ylim(0,4)
plt.xlabel('Energy(eV)',fontsize=20)
plt.ylabel('f(E)',fontsize=20)
plt.title("Comparing all 3 Distributions (At Temperature = 500 K)",fontsize=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

#Showing the plot
plt.show()
```

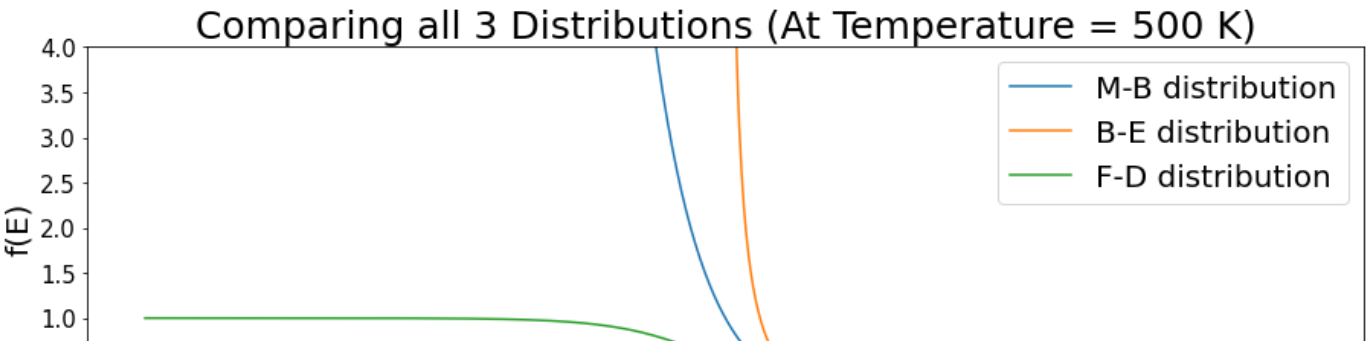
```
/tmp/ipykernel_4786/1138838089.py:2: RuntimeWarning: divide by zero encountered in true
```

```
_divide
  return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system in e
V
/tmp/ipykernel_4786/1138838089.py:2: RuntimeWarning: divide by zero encountered in true
_divide
  return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system in e
V
/tmp/ipykernel_4786/1138838089.py:2: RuntimeWarning: divide by zero encountered in true
_divide
  return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system in e
V
/tmp/ipykernel_4786/1138838089.py:2: RuntimeWarning: divide by zero encountered in true
_divide
  return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system in e
V
/tmp/ipykernel_4786/1138838089.py:2: RuntimeWarning: divide by zero encountered in true
_divide
  return 1/(np.exp(((E-u)*e)/(k*T)) + a) #e is multiplied to make the whole system in e
V
```








Comparing all 3 Distributions (At Temperature = 500 K)
f(E)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
M-B distribution
B-E distribution
F-D distribution

AIM:

Plot the distribution of particles w.r.t. Energy (dN/dE vs E) in 3D for :-

1. Relativistic & Non-Relativistic Bosons both at low & high temperatures.

2. Relativistic & Non-Relativistic Fermions both at low & high temperatures.

THEORY:

(Source : Practical Hope Youtube Channel)

The density of particles is the product of the density of states and the average number of particles in each state,

$$\frac{dN}{dE} = g(E).\bar{n}(E) = \frac{C_n\sqrt{E}}{e^{\beta(E-\mu)} - 1} \quad \text{(Non-relativistic Bosons)}$$

$$\frac{dN}{dE} = g(E).\bar{n}(E) = \frac{C_r E^2}{e^{\beta(E-\mu)} - 1} \quad \text{(Relativistic Bosons)}$$

$$\frac{dN}{dE} = g(E).\bar{n}(E) = \frac{C_n\sqrt{E}}{e^{\beta(E-\mu)} + 1} \quad \text{(Non-relativistic Fermions)}$$

$$\frac{dN}{dE} = g(E).\bar{n}(E) = \frac{C_r E^2}{e^{\beta(E-\mu)} + 1} \quad \text{(Relativistic Fermions)}$$

$$C_n = (2s+1)\frac{2\pi V(2m)^{3/2}}{h^3} \qquad C_r = 2s\frac{4\pi V}{h^3c^3}$$

1. Non-Relativistic Fermions

Step-1 : Import necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

Step-2 : Define required Constants

```
e = 1.6e-19 #Electronic Charge
kb = 1.38e-23 #Boltzmann Constant
h = 6.626e-34 #Planck's Constant
s = 0.5 #Spin
u = 1 #Chemical Potential in eV
V = 1 #Volume
m = 9.1e-31 #Mass of electron
```

Step-3 : Define Energy & Temperature Range

```
E = np.arange(0,2,0.001) #Energy in eV
T = np.array([100,1000]) #Temperature in Kelvin (100K & 1000K)
```

Step-4 : Evaluate Cn using above mentioned formula

```
Cn = (2*s + 1)*(2*3.14*V*(2*m)**1.5)/(h**3)
```

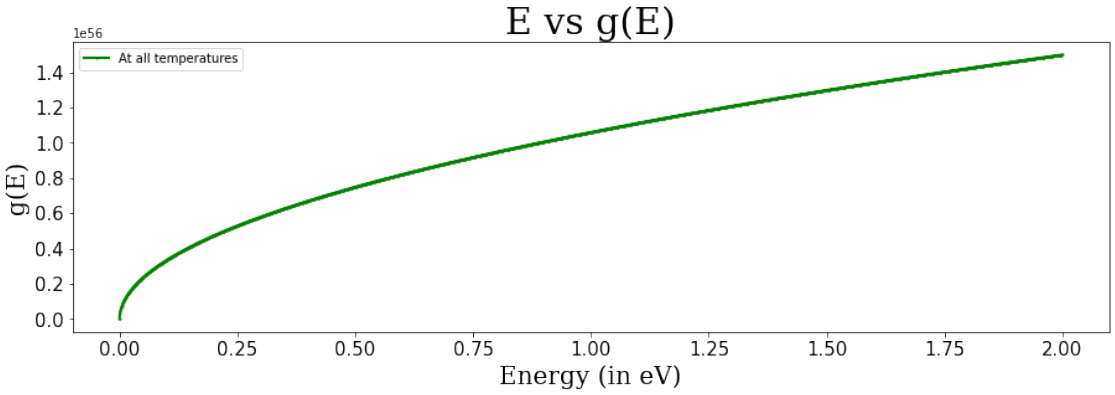
Step-5 : Evaluate g(E), n(E) & dN/dE using above mentioned formulaes

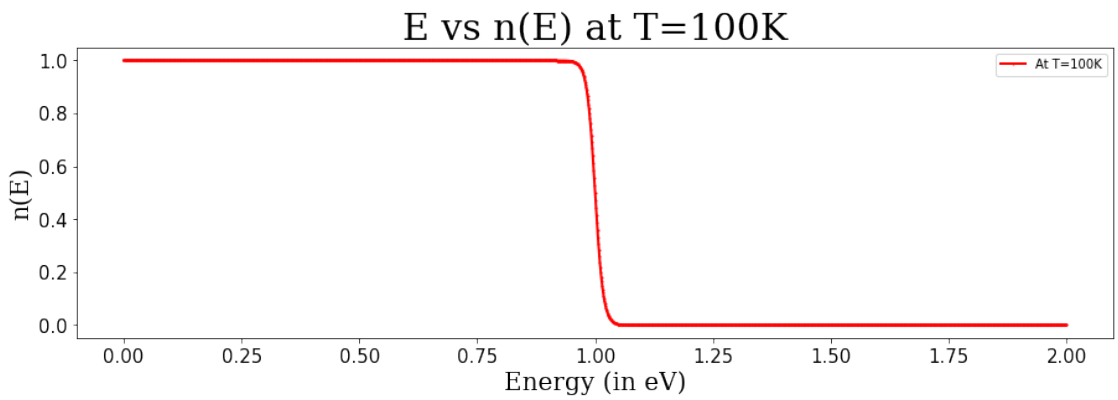
```
b = 1/(kb*T)
g = Cn * np.sqrt(E) #Density of States
n100 = 1/(np.exp((E-u)*e*b[0])+1) #At 100K
n1000 = 1/(np.exp((E-u)*e*b[1])+1) #At 1000K
f100 = n100*g #dN/dE at 100K
f1000 = n1000*g #dN/dE at 1000K
```

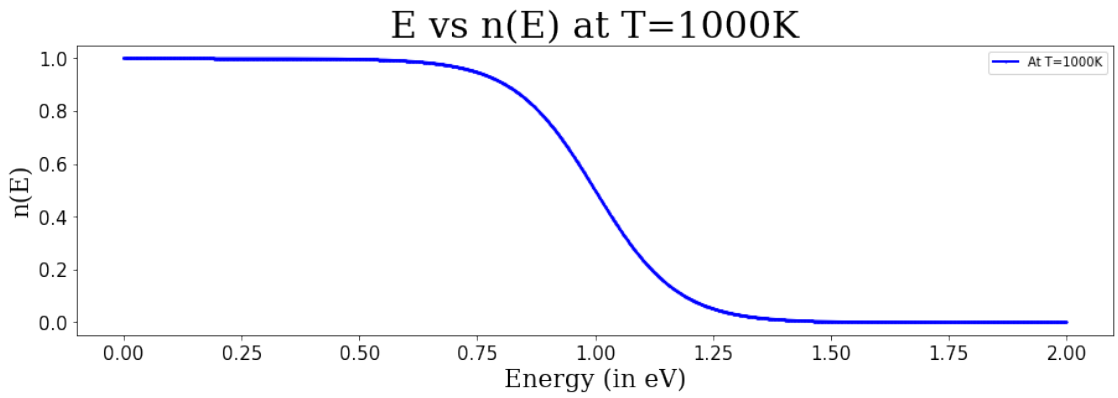
Step-6 : Plot required graphs

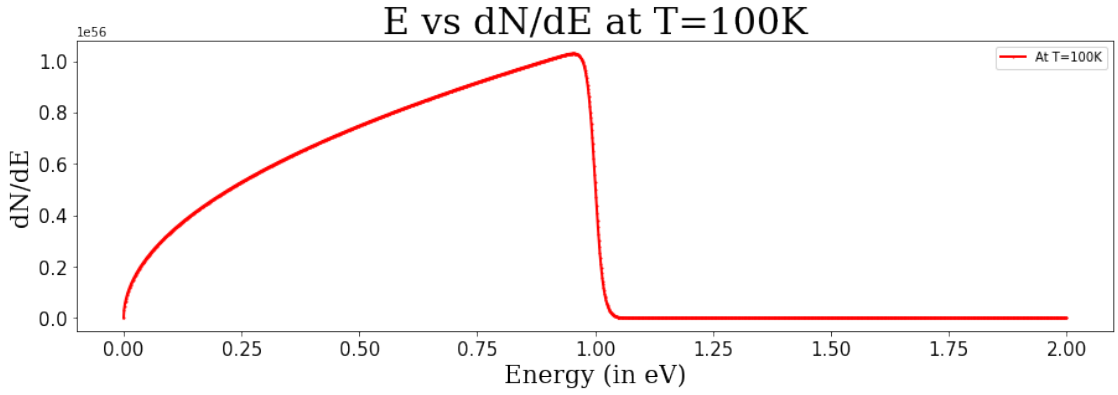
```
### Step-6 : Plot required graphs

fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

#plt.suptitle("Non-Relativistic Fermions")
plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,1)
plt.plot(E,g,"o-g",lw="2",ms="1",label="At all temperatures")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("g(E)",fontdict=fontji)
plt.title("E vs g(E)",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,2)
plt.plot(E,n100,"o-r",lw="2",ms="1",label="At T=100K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E) at T=100K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,3)
plt.plot(E,n1000,"o-b",lw="2",ms="1",label="At T=1000K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
```

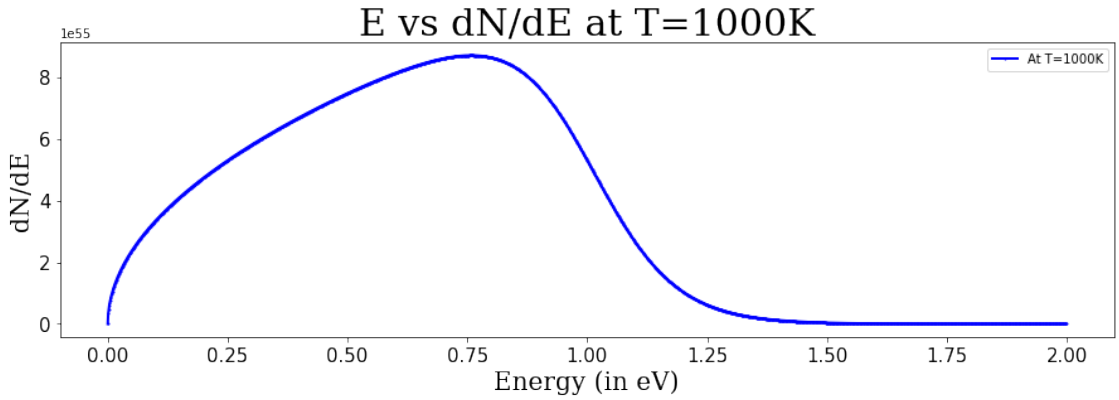
```
plt.title("E vs n(E) at T=1000K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,4)
plt.plot(E,f100,"o-r",lw="2",ms="1",label="At T=100K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T=100K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,5)
plt.plot(E,f1000,"o-b",lw="2",ms="1",label="At T=1000K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T=1000K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```




E vs n(E) at T=100K
At T=100K
n(E)
Energy (in eV)


E vs n(E) at T=1000K
At T=1000K
n(E)
Energy (in eV)

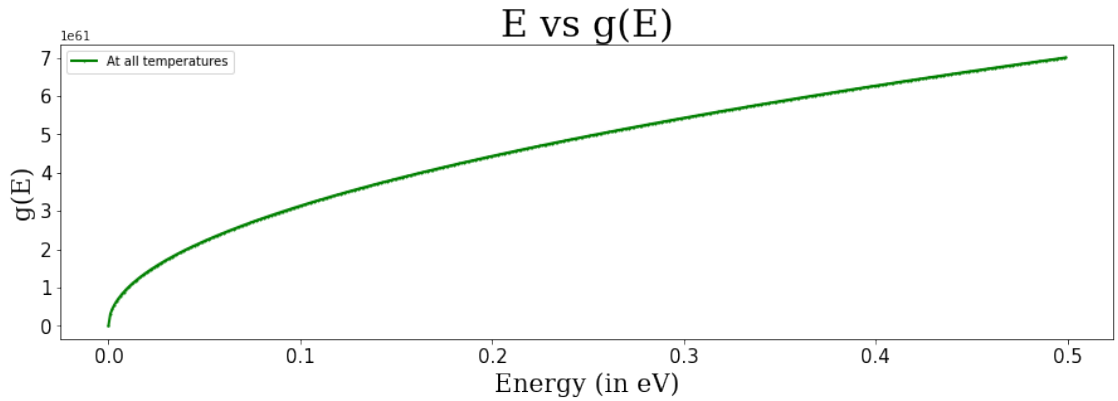

E vs dN/dE at T=100K
1e56
At T=100K
dN/dE
Energy (in eV)

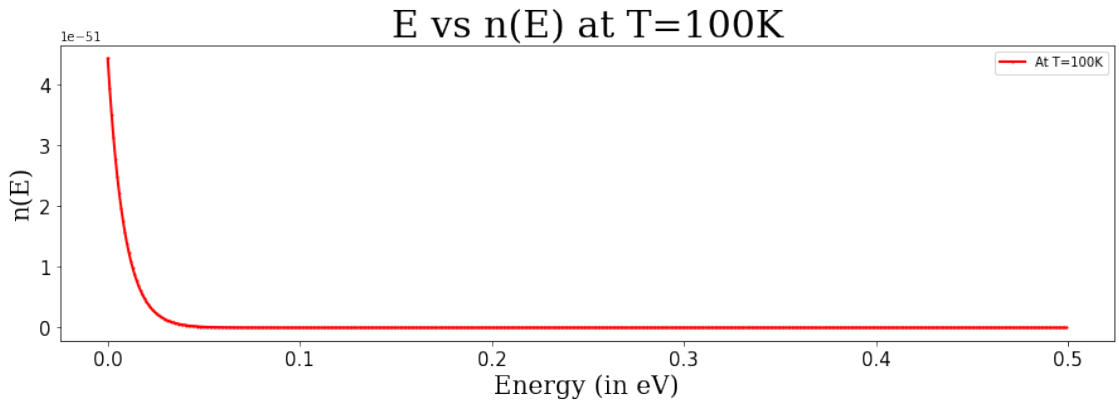

E vs dN/dE at T=1000K
1e55
At T=1000K
dN/dE
Energy (in eV)

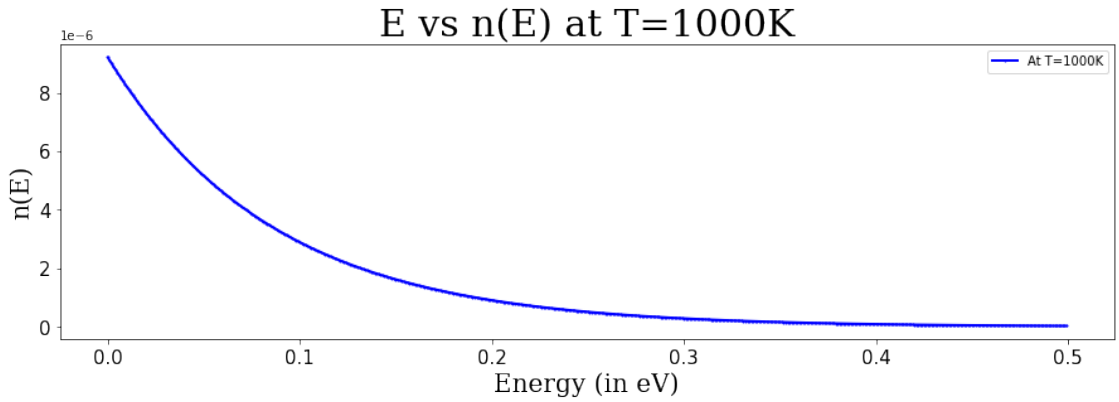
2. Non-Relativistic Bosons (same as above with few changes)

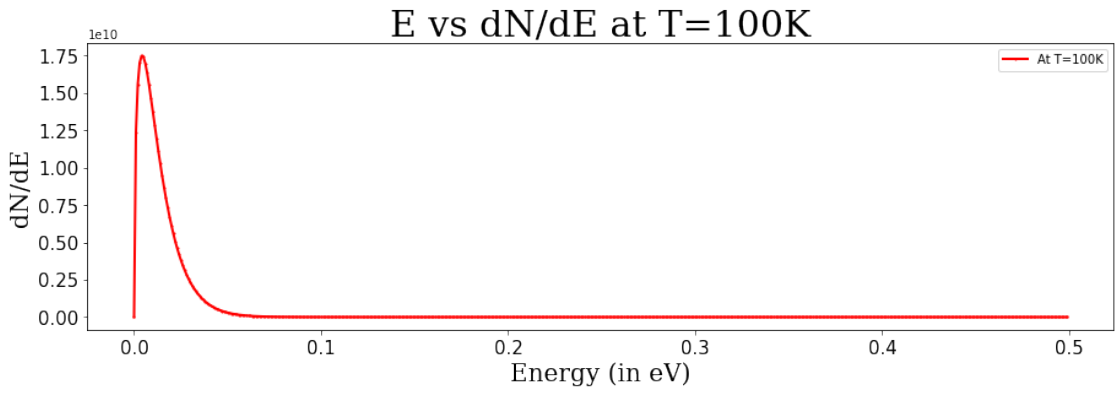
```
### Step-1 : Import necessary libraries

import numpy as np
import matplotlib.pyplot as plt

### Step-2 : Define required Constants

e = 1.6e-19 #Electronic Charge
kb = 1.38e-23 #Boltzmann Constant
h = 6.626e-34 #Planck's Constant
s = 1 #Spin
u = -1 #Chemical Potential in eV
V = 1 #Volume
m = 4*1.66e-27 #Mass = 4amu (1amu = 1.66e-27)

### Step-3 : Define Energy & Temperature Range

E = np.arange(0,0.5,0.001) #Energy in eV
T = np.array([100,1000]) #Temperature in Kelvin (100K & 1000K)

### Step-4 : Evaluate Cn using above mentioned formula

Cn = (2*s + 1)*(2*3.14*V*(2*m)**1.5)/(h**3)

### Step-5 : Evaluate g(E), n(E) & dN/dE using above mentioned
formulaes

b = 1/(kb*T)
g = Cn * np.sqrt(E) #Density of States
n100 = 1/(np.exp((E-u)*e*b[0])-1) #At 100K
n1000 = 1/(np.exp((E-u)*e*b[1])-1) #At 1000K
f100 = n100*g #dN/dE at 100K
f1000 = n1000*g #dN/dE at 1000K

### Step-6 : Plot required graphs

fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

#plt.suptitle("Non-Relativistic Bosons")
plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,1)
plt.plot(E,g,"o-g",lw="2",ms="1",label="At all temperatures")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("g(E)",fontdict=fontji)
plt.title("E vs g(E)",fontdict=fontji2)
plt.xticks(fontsize=15)
```

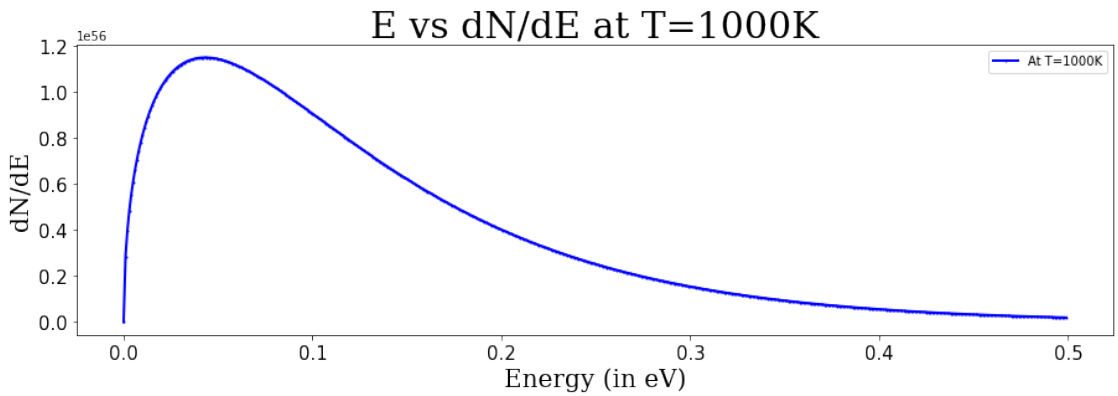
```
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,2)
plt.plot(E,n100,"o-r",lw="2",ms="1",label="At T=100K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E) at T=100K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,3)
plt.plot(E,n1000,"o-b",lw="2",ms="1",label="At T=1000K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E) at T=1000K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,4)
plt.plot(E,f100,"o-r",lw="2",ms="1",label="At T=100K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T=100K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,5)
plt.plot(E,f1000,"o-b",lw="2",ms="1",label="At T=1000K")
plt.legend(loc="best")
plt.xlabel("Energy (in eV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T=1000K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```


E vs g(E)
1e61
At all temperatures
g(E)
Energy (in eV)


E vs n(E) at T=100K
1e−51
At T=100K
n(E)
Energy (in eV)


E vs n(E) at T=1000K
1e−6
At T=1000K
n(E)
Energy (in eV)

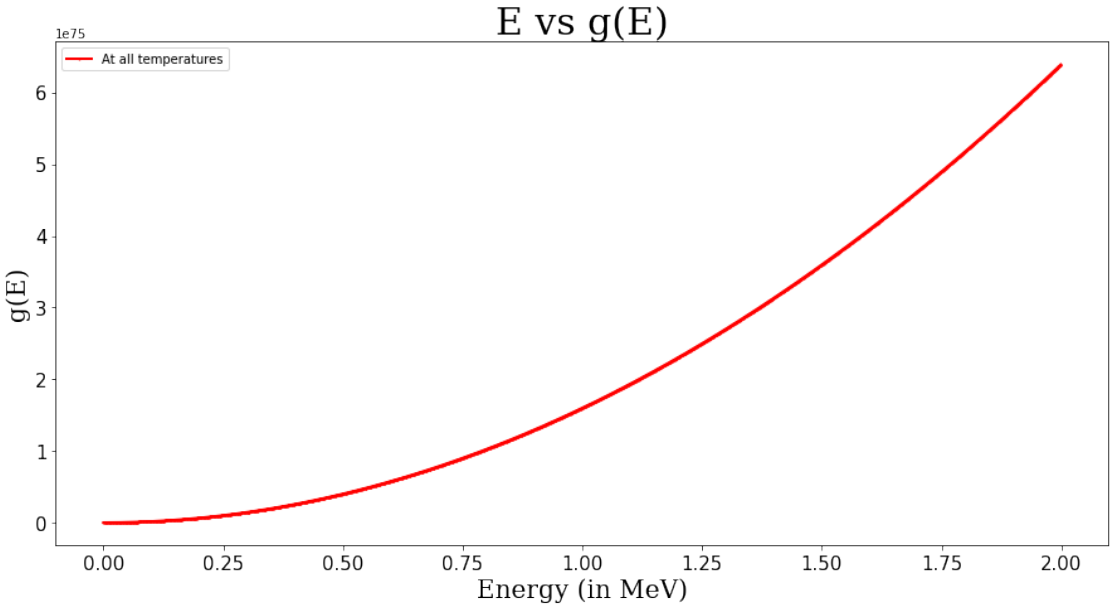

E vs dN/dE at T=100K
1e10
At T=100K
dN/dE
Energy (in eV)

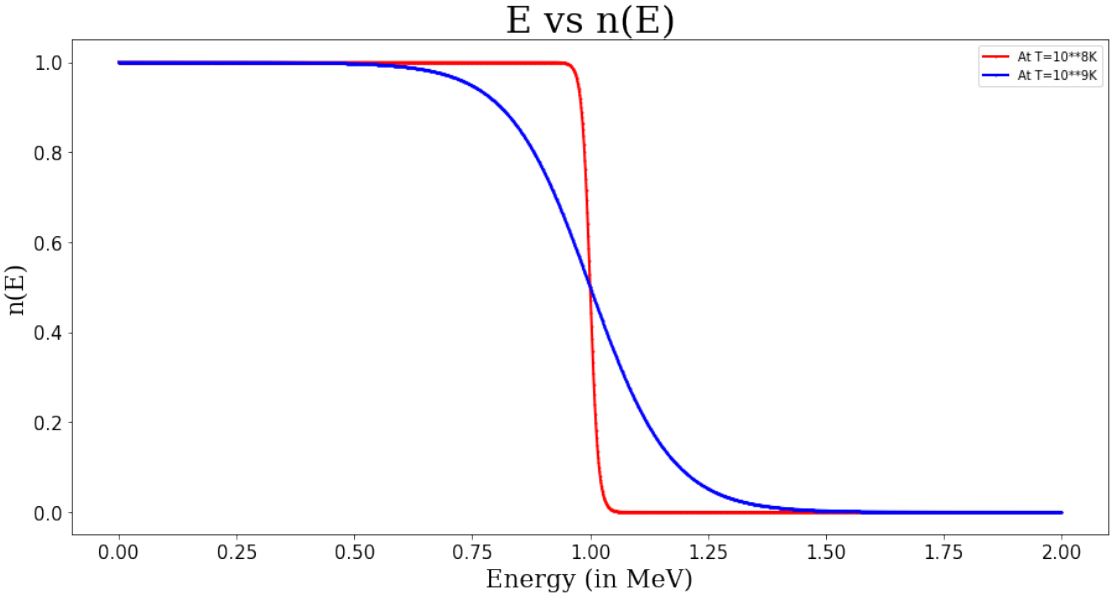


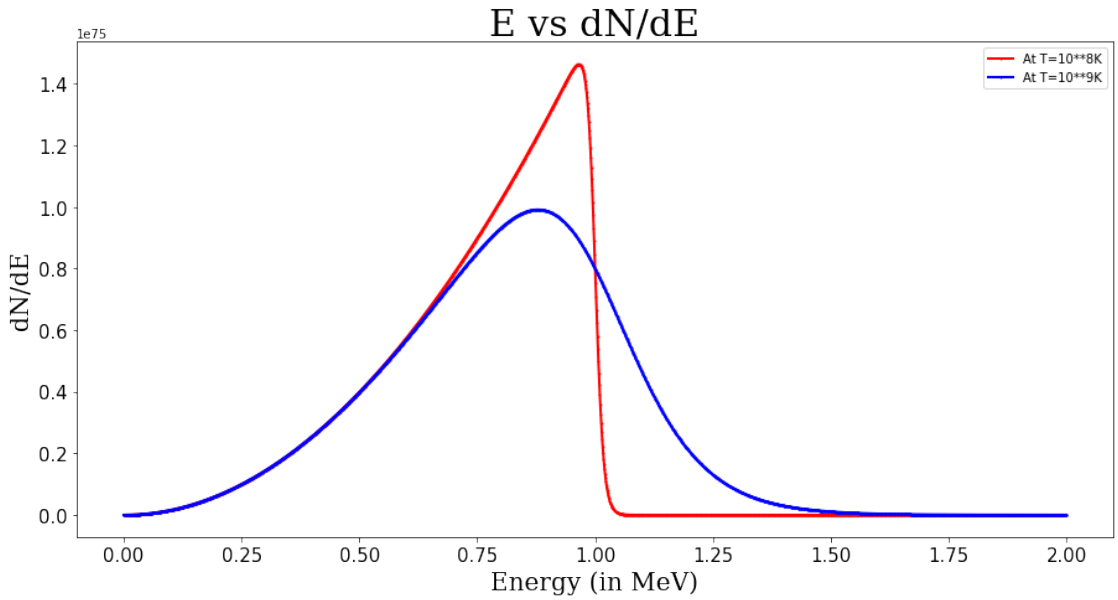
3. Relativistic Fermions (same as above with few changes)

```
### Step-1 : Import necessary libraries

import numpy as np
import matplotlib.pyplot as plt

### Step-2 : Define required Constants

e = 1.6e-19 #Electronic Charge
kb = 1.38e-23 #Boltzmann Constant
h = 6.626e-34 #Planck's Constant
s = 0.5 #Spin
u = 1 #Chemical Potential in MeV
V = 1 #Volume
#m = 9.1e-31 #Mass of electron
c = 3e8 #Speed of Light

### Step-3 : Define Energy & Temperature Range

E = np.arange(0,2,0.001) #Energy in MeV
T = np.array([10**8,10**9]) #Temperature in Kelvin

### Step-4 : Evaluate Cr using above mentioned formula

Cr = (2*s*4*3.14*V)/((h**3)*(c**3))

### Step-5 : Evaluate g(E), n(E) & dN/dE using above mentioned
formulaes

b = 1/(kb*T)
g = Cr * (E)**2 #Density of States
n100 = 1/(np.exp((E-u)*e*(10**6)*b[0])+1) #At 10**8K
n1000 = 1/(np.exp((E-u)*e*(10**6)*b[1])+1) #At 10**9K
f100 = n100*g #dN/dE at 10**8K
f1000 = n1000*g #dN/dE at 10**9K
```

```
### Step-6 : Plot required graphs

fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

#plt.suptitle("Relativistic Fermions")
plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(3,1,1)
plt.plot(E,g,"o-r",lw="2",ms="1",label="At all temperatures")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("g(E)",fontdict=fontji)
plt.title("E vs g(E)",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(3,1,2)
plt.plot(E,n100,"o-r",lw="2",ms="1",label="At T=10**8K")
plt.plot(E,n1000,"o-b",lw="2",ms="1",label="At T=10**9K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E)",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(3,1,3)
plt.plot(E,f100,"o-r",lw="2",ms="1",label="At T=10**8K")
plt.plot(E,f1000,"o-b",lw="2",ms="1",label="At T=10**9K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```


E vs g(E)
1e75
At all temperatures
g(E)
Energy (in MeV)


E vs n(E)
At T=10**8K
At T=10**9K
n(E)
Energy (in MeV)

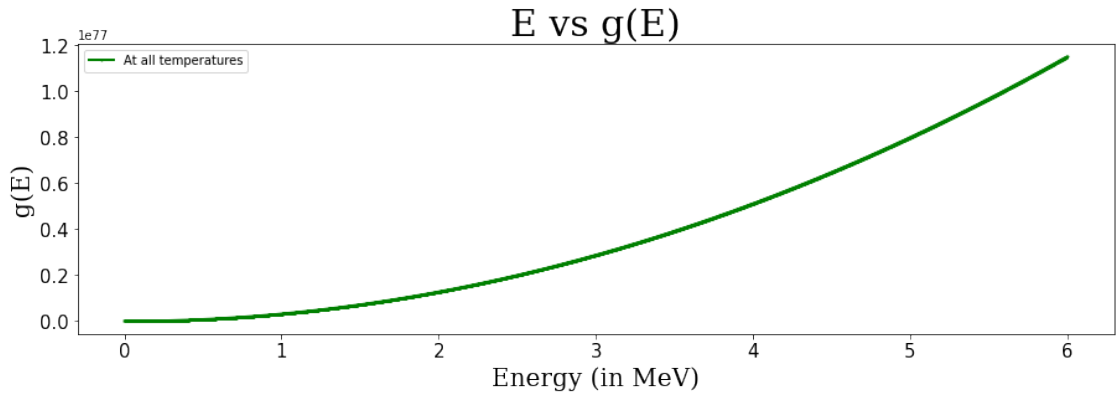


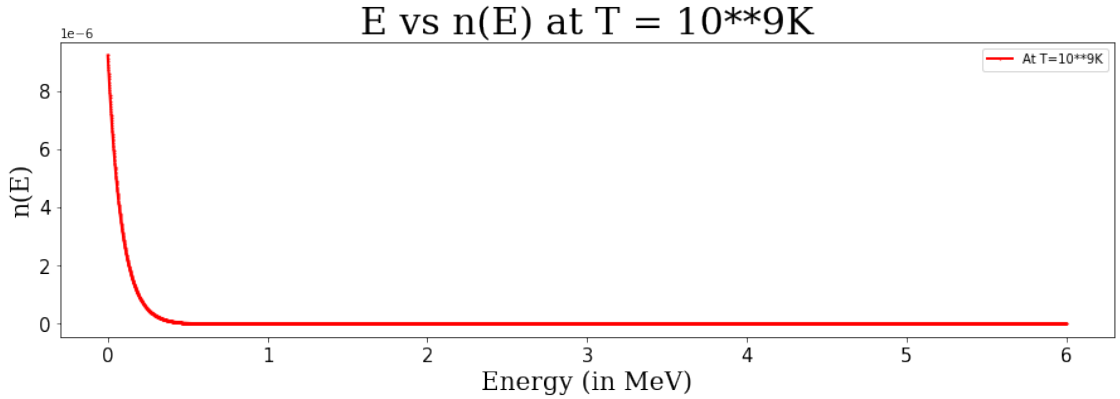
4. Relativistic Bosons (same as above with few changes)

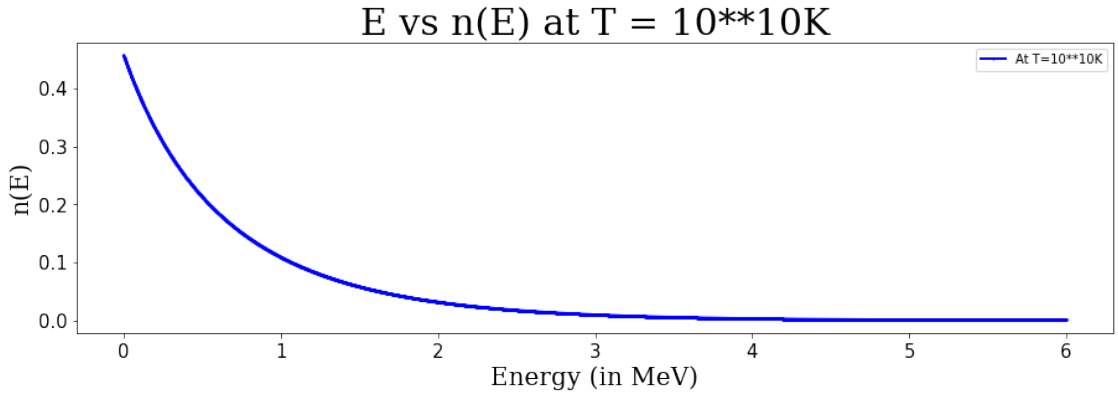
```
### Step-1 : Import necessary libraries

import numpy as np
import matplotlib.pyplot as plt

### Step-2 : Define required Constants

e = 1.6e-19 #Electronic Charge
kb = 1.38e-23 #Boltzmann Constant
h = 6.626e-34 #Planck's Constant
s = 1 #Spin
u = -1 #Chemical Potential in MeV
V = 1 #Volume
#m = 9.1e-31 #Mass of electron
c = 3e8 #Speed of Light

### Step-3 : Define Energy & Temperature Range

E = np.arange(0,6,0.001) #Energy in MeV
T = np.array([10**9,10**10]) #Temperature in Kelvin

### Step-4 : Evaluate Cr using above mentioned formula

Cr = (2*s*4*3.14*V)/((h**3)*(c**3))

### Step-5 : Evaluate g(E), n(E) & dN/dE using above mentioned
formulaes

b = 1/(kb*T)
```

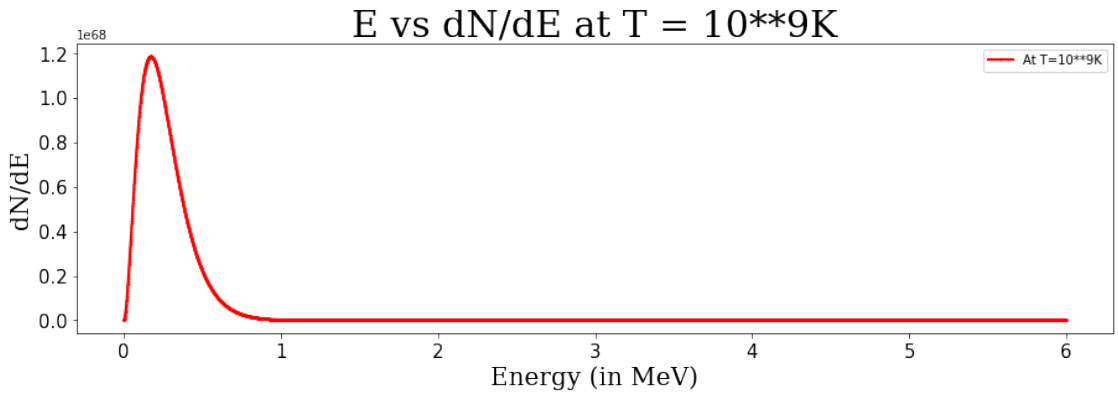
```
g = Cr * (E)**2 #Density of States
n100 = 1/(np.exp((E-u)*e*(10**6)*b[0])-1) #At 10**9K
n1000 = 1/(np.exp((E-u)*e*(10**6)*b[1])-1) #At 10**10K
f100 = n100*g #dN/dE at 10**9K
f1000 = n1000*g #dN/dE at 10**10K

### Step-6 : Plot required graphs

fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

#plt.suptitle("Relativistic Bosons")
plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,1)
plt.plot(E,g,"o-g",lw="2",ms="1",label="At all temperatures")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("g(E)",fontdict=fontji)
plt.title("E vs g(E)",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,2)
plt.plot(E,n100,"o-r",lw="2",ms="1",label="At T=10**9K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E) at T = 10**9K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,3)
plt.plot(E,n1000,"o-b",lw="2",ms="1",label="At T=10**10K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("n(E)",fontdict=fontji)
plt.title("E vs n(E) at T = 10**10K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,4)
plt.plot(E,f100,"o-r",lw="2",ms="1",label="At T=10**9K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
```

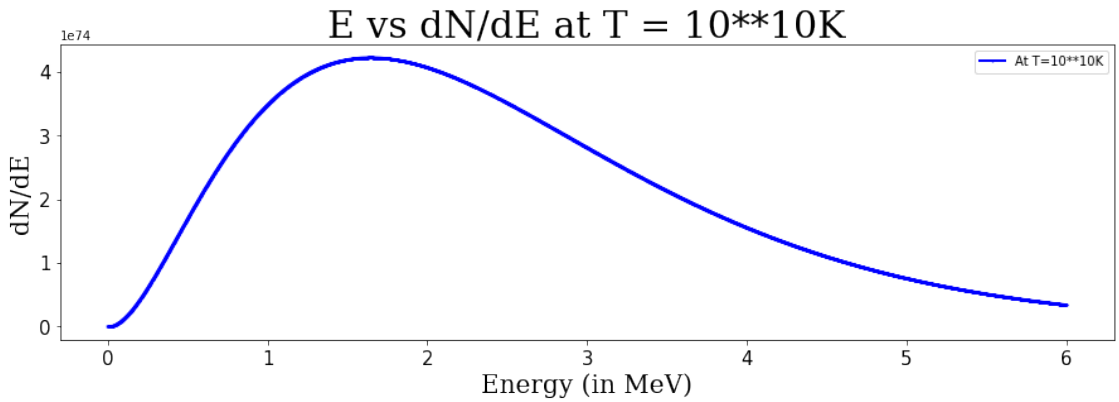
```
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T = 10**9K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()

plt.figure(figsize=(15,25)) #Setting size of the figure
plt.subplot(5,1,5)
plt.plot(E,f1000,"o-b",lw="2",ms="1",label="At T=10**10K")
plt.legend(loc="best")
plt.xlabel("Energy (in MeV)",fontdict=fontji)
plt.ylabel("dN/dE",fontdict=fontji)
plt.title("E vs dN/dE at T = 10**10K",fontdict=fontji2)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```






E vs n(E) at T = 10**10K
At T=10**10K
n(E)
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
Energy (in MeV)


E vs dN/dE at T = 10**9K
1e68
At T=10**9K
dN/dE
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4
5
6
Energy (in MeV)


E vs dN/dE at T = 10**10K
1e74
At T=10**10K
dN/dE
0
1
2
3
4
0
1
2
3
4
5
6
Energy (in MeV)

AIM

1. Plot Planck's law & Rayleigh-Jean's Law of Black body radiation w.r.t. wavelength at different temperatures.

2. Compare both at high & low temperatures.

3. Verify Weins-Displacement Law

Breif about BlackBody Radiation

- "Blackbody radiation" or "Cavity radiation" refers to an object or system which absorbs all radiation incident upon it and re-radiates energy which is characteristic of this radiating system only, not dependent upon the type of radiation which is incident upon it.
- The radiated energy can be considered to be produced by standing wave or resonant modes of the cavity which is radiating.

Step-1 : Importing necessary libraries

In [140…

```
import numpy as np
from scipy.constants import h,c,k,pi
import matplotlib.pyplot as plt
```

Step-2 : Define an array for wavelength in micrometers & then convert it in meters

In [141…

```
L = (np.arange(0.1,30,0.005))*(1e-6) #0.1 um to 30 um with step size 0.005um
```

Step-3 : Define function planck_lamda for Plancks Law of Black Body Radiation

Plancks Radiation Formula in terms of Wavelength :

Energy per unit volume per unit wavelength $S_\lambda = \frac{8\pi hc}{\lambda^5}\frac{1}{e^{hc/\lambda kT}-1}$

Image Source : BlackBody Radiation.

In [142…

```
def planck_lamda(L,T):
    a = (8*pi*h*c)/(L**5)
    b = (h*c)/(L*k*T)
    c1 = np.exp(b)-1
    d = a/c1
    return d
```

Step-4 : Find Intensity at 4 different temperatures (ex: 500K, 700K, 900K & 1100K)

In [143…

```
T500 = planck_lamda(L , 500)
T700 = planck_lamda(L , 700)
T900 = planck_lamda(L , 900)
T1100 = planck_lamda(L , 1100)
```

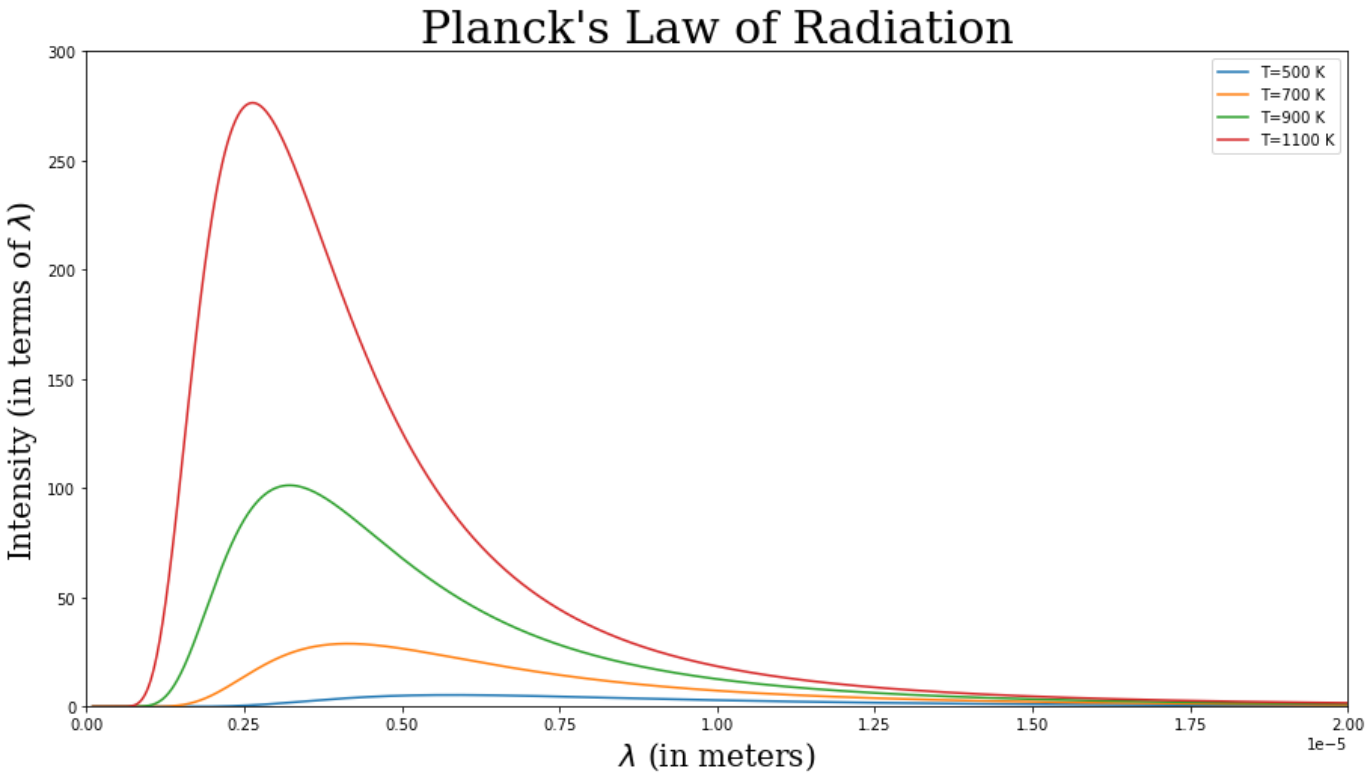
Step-5 : Plotting Planck's Law of Radiation at different temperatures

In [144…

```
plt.figure(figsize=(15, 8)) #Changing Figure Size
fontji = {'family':'serif','size':20}
fontji2 = {'family':'serif','size':30}

plt.plot(L, T500,label='T=500 K')
plt.plot(L, T700 ,label='T=700 K')
plt.plot(L, T900 ,label='T=900 K')
plt.plot(L, T1100 ,label='T=1100 K')
plt.legend()
plt.xlabel(r"$\lambda$ (in meters)",fontdict=fontji)
plt.ylabel(r"Intensity (in terms of $\lambda$)",fontdict=fontji)
plt.title("Planck's Law of Radiation",fontdict=fontji2)
plt.ylim(0,300)
plt.xlim(0,0.00002)
```

Out[144…

```
(0.0, 2e-05)
```



Step-6 : Define function rayleigh_lamda for Rayleigh Jeans Formula

In [145…

```
def r_lamda(L,T):
    i = 8*pi*k*T/(L**4)
    return i
```

Step-7 : Finding Intensity at different temperatures using r_lamda

In [146…

```
Tr500 = r_lamda(L , 500)
Tr700 = r_lamda(L , 700)
Tr900 = r_lamda(L , 900)
Tr1100 = r_lamda(L , 1100)
```

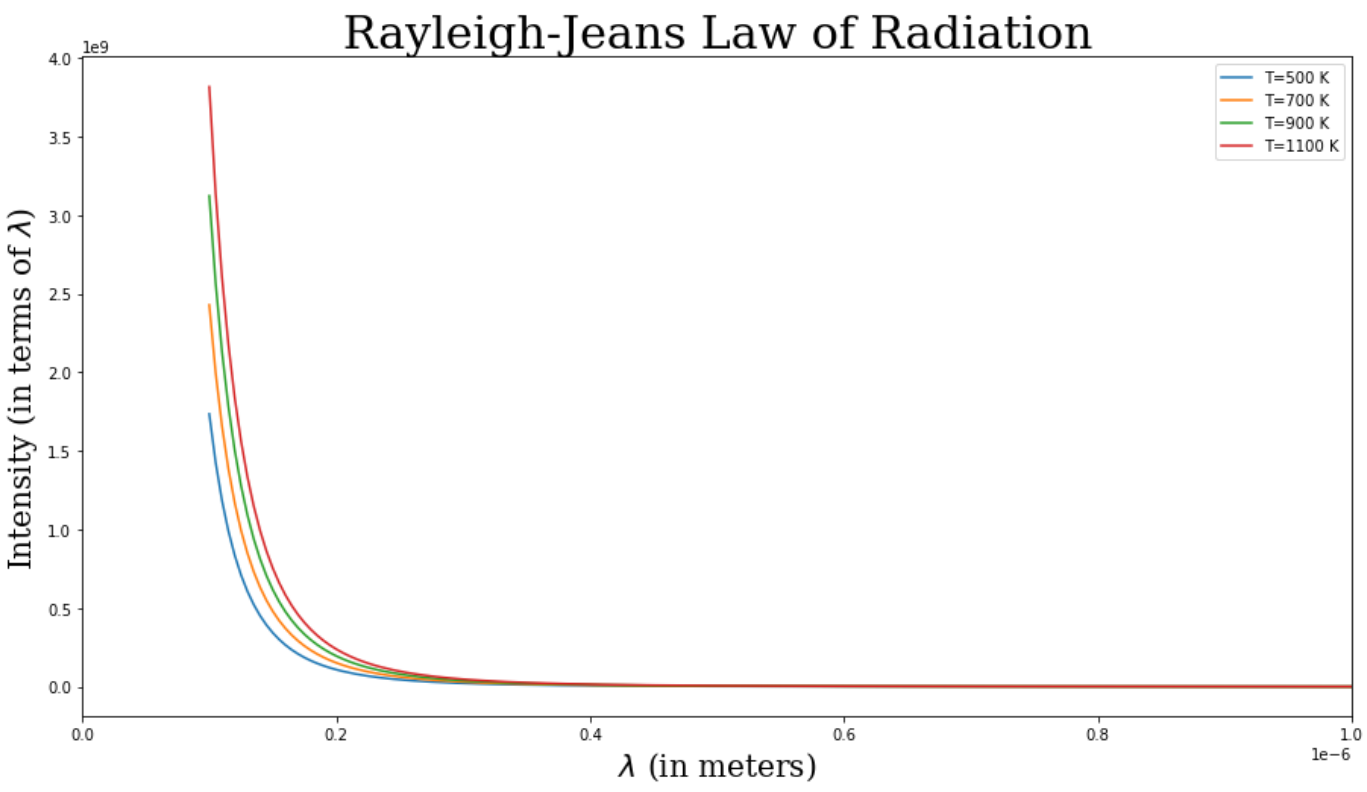
Step-8 : Plotting Rayleigh Jeans formula for different temperatures

In [147…

```
plt.figure(figsize=(15, 8)) #Changing Figure Size

plt.plot(L, Tr500,label='T=500 K')
plt.plot(L, Tr700 ,label='T=700 K')
plt.plot(L, Tr900 ,label='T=900 K')
plt.plot(L, Tr1100 ,label='T=1100 K')
plt.legend()
plt.xlabel(r"$\lambda$ (in meters)",fontdict=fontji)
plt.ylabel(r"Intensity (in terms of $\lambda$)",fontdict=fontji)
plt.title("Rayleigh-Jeans Law of Radiation",fontdict=fontji2)
#plt.ylim(0,1.2)
plt.xlim(0,0.000001)
```

Out[147…

```
(0.0, 1e-06)
```



Step-9 : Comparing Rayleigh Jeans & Plancks Formula at low & high temperatures

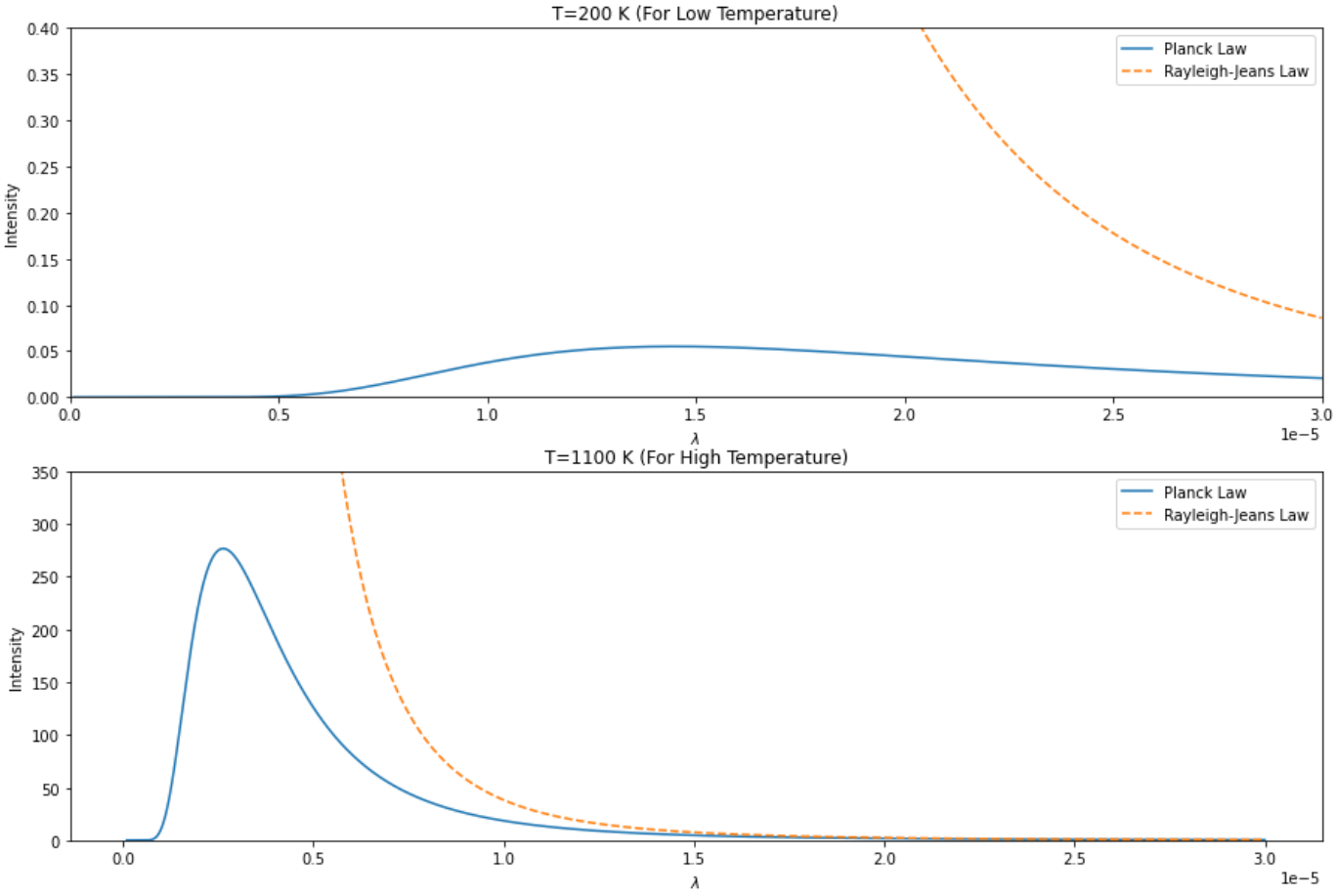
In [148…

```
plt.suptitle("Comparing Rayleigh-Jeans & Plancks Law for BBR at low & high temperatures
plt.figure(figsize=(15, 10)) #Changing Figure Size

plt.subplot(2,1,1)
plt.plot(L, (planck_lamda(L,200)),label='Planck Law')
plt.plot(L, (r_lamda(L,200)) , "--" , label="Rayleigh-Jeans Law")
plt.legend(loc="best")
plt.xlabel(r"$\lambda$ ")
plt.ylabel("Intensity")
plt.title("T=200 K (For Low Temperature)")
plt.ylim(0,0.4)
plt.xlim(0,0.00003)

plt.subplot(2,1,2)
plt.plot(L, T1100 ,label='Planck Law')
plt.plot(L, Tr1100 , "--" , label="Rayleigh-Jeans Law")
plt.legend(loc="best")
plt.xlabel(r"$\lambda$ ")
plt.ylabel("Intensity")
plt.title("T=1100 K (For High Temperature)")
plt.ylim(0,350)
```

```
/tmp/ipykernel_3710/3404406587.py:4: RuntimeWarning: overflow encountered in exp
  c1 = np.exp(b)-1
```

Out[148…

```
(0.0, 350.0)

<Figure size 432x288 with 0 Axes>
```



> **Conclusion** : The Rayleigh-Jeans curve agrees with the Planck radiation formula for long wavelengths or low frequencies.

Step-10 : Verifying Weins Displacement Law

- When the temperature of a blackbody radiator increases, the overall radiated energy increases and the peak of the radiation curve moves to shorter wavelengths.

- When the maximum is evaluated from the Planck radiation formula, the product of the peak wavelength and the temperature is found to be a constant.
- Formula : $\lambda_{peak} T = 2.898 \times 10^{-3} \, m \cdot K$
- This relationship is called **Wien's displacement law**.

> **Note** : It should be noted that the peak of the radiation curve in the Wien relationship is the peak only because the intensity is plotted as a function of wavelength. If frequency or some other variable is used on the horizontal axis, the peak will be at a different wavelength.

- Source : Weins Displacement Law