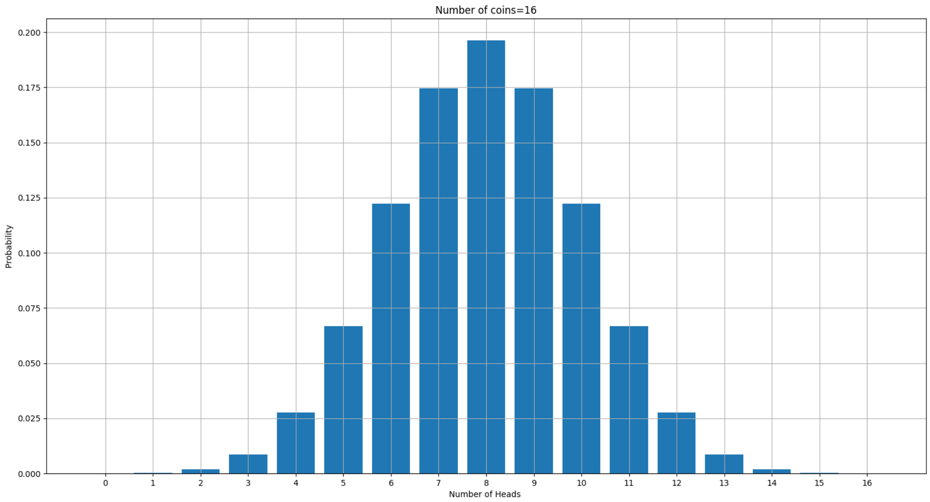
Statistical Mechanics Lab Report

Ankit Mishra , 1819061 , Sem 6

Problem 1. Plot the probability of various macrostates in coin-tossing experiment (two level system) versus number of heads with 4, 8, 16 coins etc.

```
'''
This program calulates probablity of different outcomes of N coint tosses.
'''

import math
import matplotlib.pyplot as plt


# N is total number if coins
N = 16

# total multiplicity, omegaT is toal number of microsates
# N coins can have
omegaT = 2**N


# multiplicity of n heads, is the number of ways n head
# can show up on toss of N coins. The list stores that value
# for n ranging to no head(=0) to all head(=N)
omega_n = [math.comb(N,n) for n in range(N+1)]

# TO calulate probability of n head,just divide multiplicty of
# n head by total multiplicity
probability_n = [math.comb(N,n)/omegaT for n in range(N+1)]

# Plot
plt.grid(True)
plt.bar([n for n in range(N+1)],probability_n)
plt.title('Number of coins=%i'%N)
plt.xticks([n for n in range(N+1)])

plt.xlabel("Number of Heads")
plt.ylabel(" Probability ")
plt.show()
```

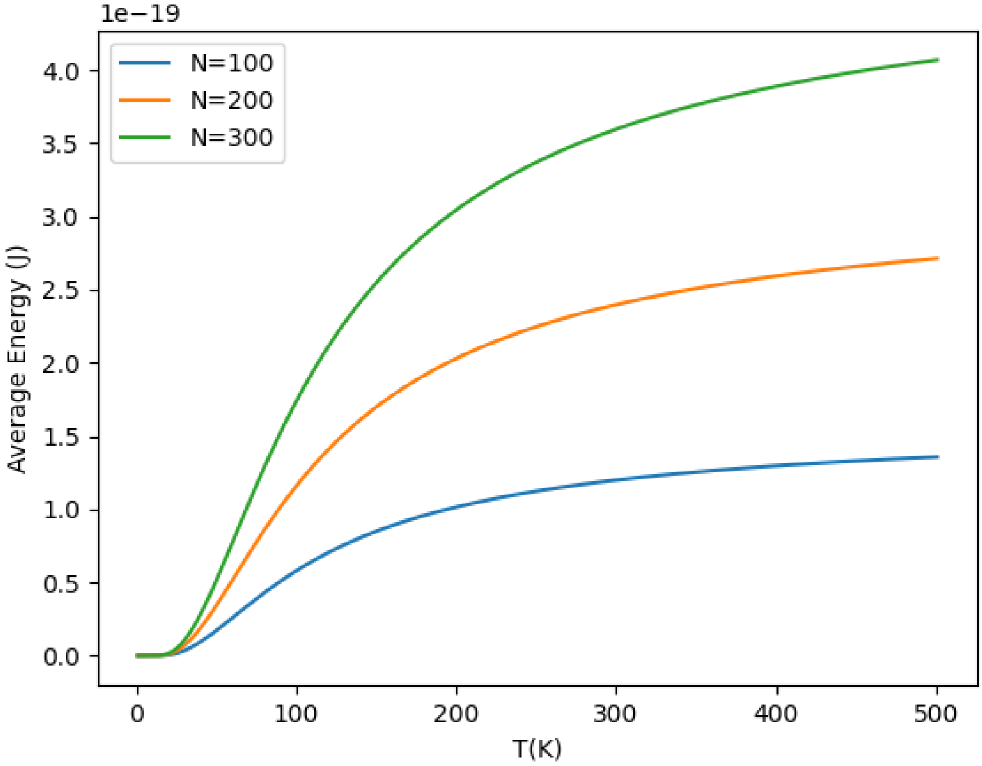


Program 2. Computation of the partition function Z(b) for the systems with a finite number of single particle levels (e.g., 2 level, 3 level etc.) and finite number of non-interacting particles N under Maxwell-Boltzmann/ Fermi-Dirac/Bose Einstein statistics: a) Study the behavior of Z(b), average energy, C v , and entropy and its dependence upon the temperature, total number of particles N and the spectrum of single particle energy states. b) Plot the probability of occupancy of all the states w.r.t. temperature.

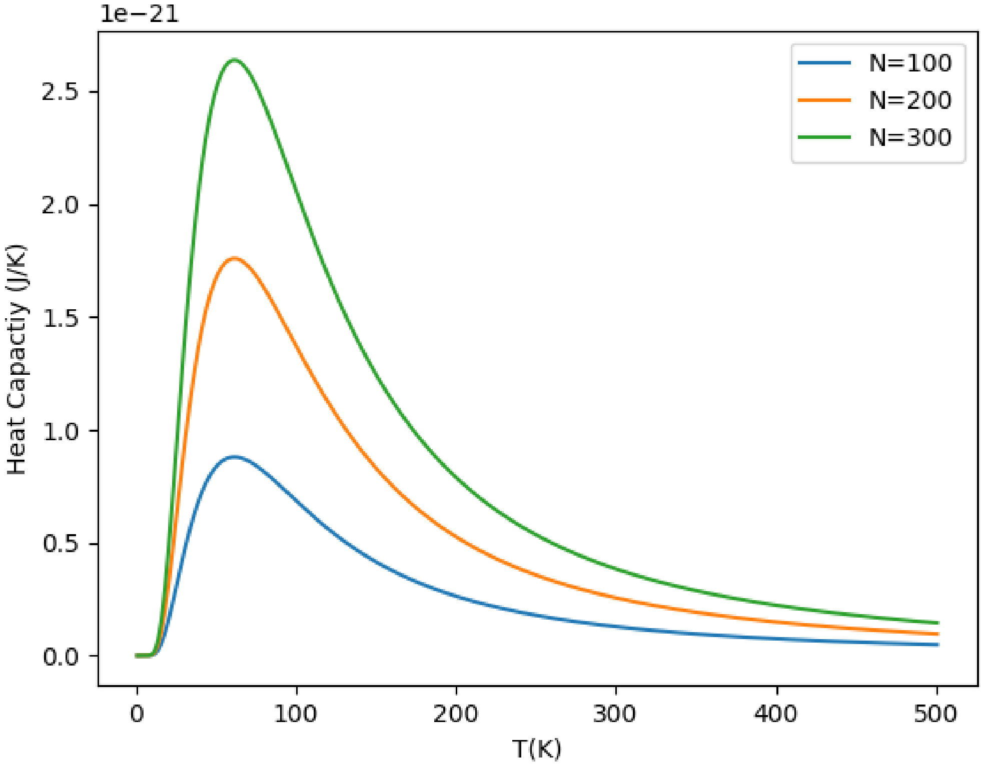
```
import numpy as np
import matplotlib.pyplot as plt

#constants , 1eV = e * (1 J)
k = 1.381*10**(-23) # J/K
e = 1.602 * 10**(-19) # coulomb

# create temperature points
Tmin = 1
Tmax = 500
dT = 0.1
T =  np.arange(Tmin,Tmax + 1,dT)

N = np.array([100,200,300]) # number of paticles

n = 3  # number of energy levels
E0 = 0 # ground state energy
dE = 0.01 # ev

# Z_list to store value of partition fucntion
# at differnt temperatures for differnt number of
# particles
```

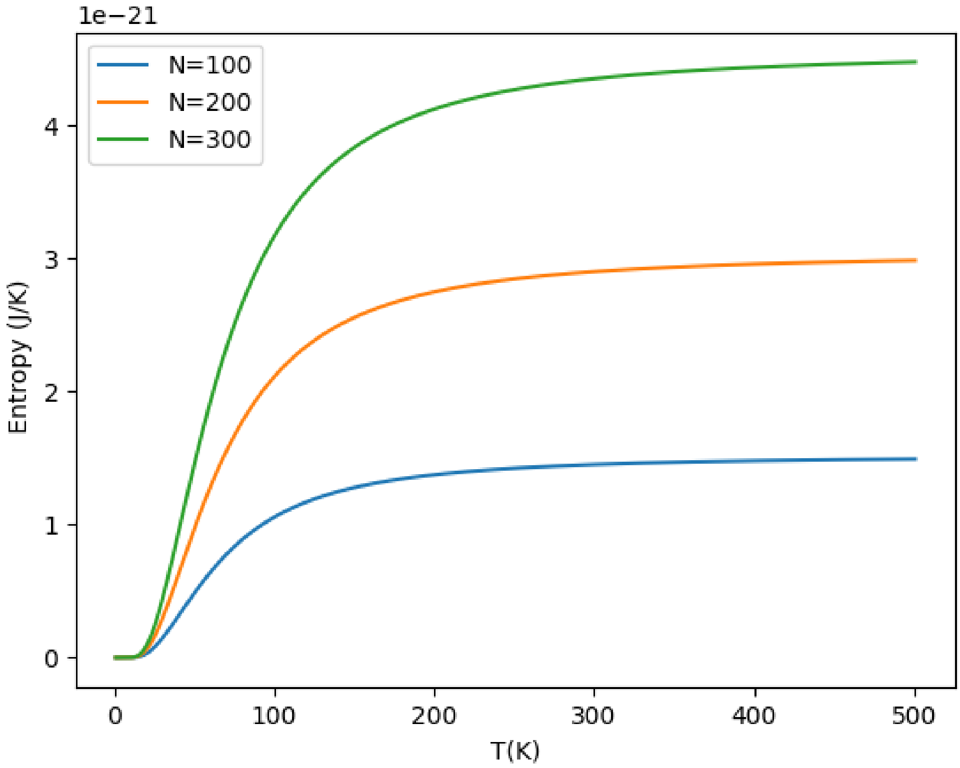
```
Z_list = np.zeros((len(N),len(T)))

# P list stoes the value of boltzmann factors of each
# state at a given temp as column, next column for next
# temp and so on. And then each colum is divided by the
# partiton function at that temp to convert those
# boltzmann factor into probabilities.

P_list = np.zeros((len(N),len(T)))

for m in range(len(N)):
    for j in range(len(T)):
        z = 0 # partition function for each particle
        for i in range(n):

            # print((e/k)/(T[j]))
            P_list[i,j] = np.e**(-(E0 + i*dE)*(e/k)/(T[j]))
            z = z + P_list[i,j]

        # the partition function for N partices is just
        Z_list[m,j] = z**N[m]
        P_list[:,j] = P_list[:,j]/z

T1 = T[:-1]
# note that multiplicatio of 2D array and 1D array
# first elemt of T is multiplied to first element
# of all rows (first colum that is). That what we need.
U = k*T1*T1*np.diff(np.log(Z_list))/dT

# specific heat capacity
T2 = T1[:-1]
Cv = np.diff(U)/dT

# Helomholtz free energy
F = -k*T*np.log(Z_list)
S = -np.diff(F)/dT

#plot
plt.figure(1)
plt.xlabel('T(K)')
plt.ylabel('Average Energy (J)')
plt.plot( T1, U[0] , label ='N='+str(N[0]))
plt.plot( T1, U[1] , label ='N='+str(N[1]))
plt.plot( T1, U[2] , label ='N='+str(N[2]))
plt.legend()

plt.figure(2)
plt.xlabel('T(K)')
plt.ylabel('Heat Capactiy (J/K)')
```

```
plt.plot(T2,Cv[0] , label ='N='+str(N[0]))
plt.plot(T2,Cv[1] , label ='N='+str(N[1]))
plt.plot(T2,Cv[2] , label ='N='+str(N[2]))
plt.legend()


plt.figure(3)
plt.xlabel('T(K)')
plt.ylabel(' Entropy (J/K)')
plt.plot(T1,S[0] , label ='N='+str(N[0]))
plt.plot(T1,S[1] , label ='N='+str(N[1]))
plt.plot(T1,S[2] , label ='N='+str(N[2]))
plt.legend()

plt.figure(4)
plt.xlabel('T(K)')
plt.ylabel('Probability of occupancy (J)')
plt.plot(T,P_list[0] , label ='State 0')
plt.plot(T,P_list[1] , label ='State 1')
plt.plot(T,P_list[2] , label ='State 2')

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

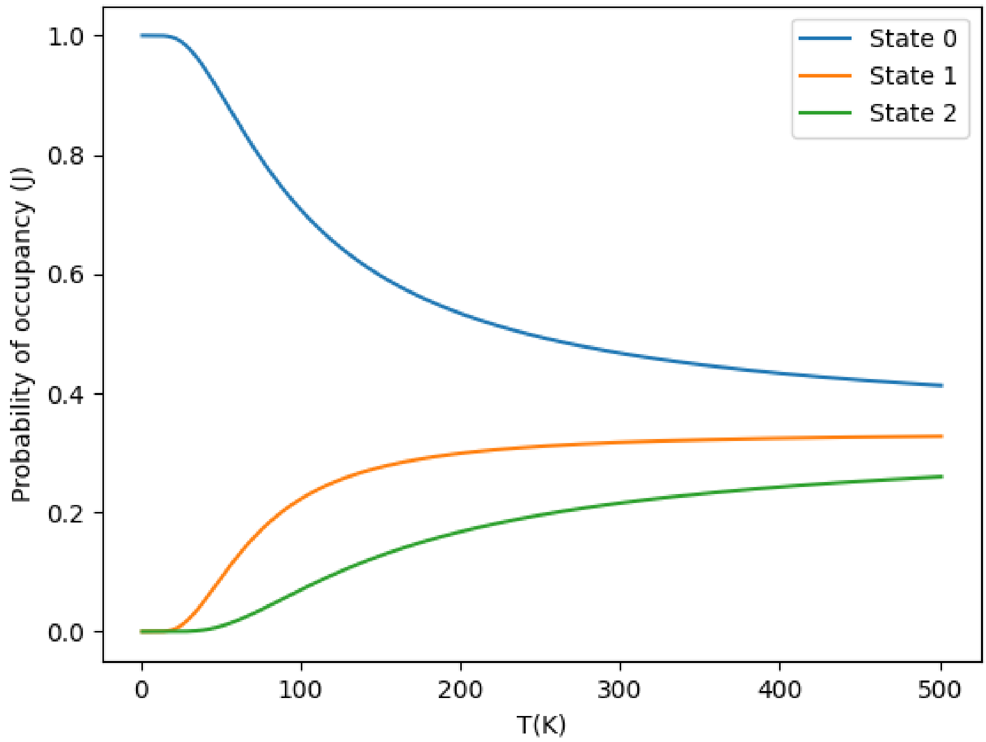


1e−19
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Average Energy (J)
N=100
N=200
N=300
0
100
200
300
400
500
T(K)


1e−21
Heat Capactiy (J/K)
T(K)
N=100
N=200
N=300
2.5
2.0
1.5
1.0
0.5
0.0
0
100
200
300
400
500



1e−21
N=100
N=200
N=300
Entropy (J/K)
T(K)
0
1
2
3
4
0
100
200
300
400
500



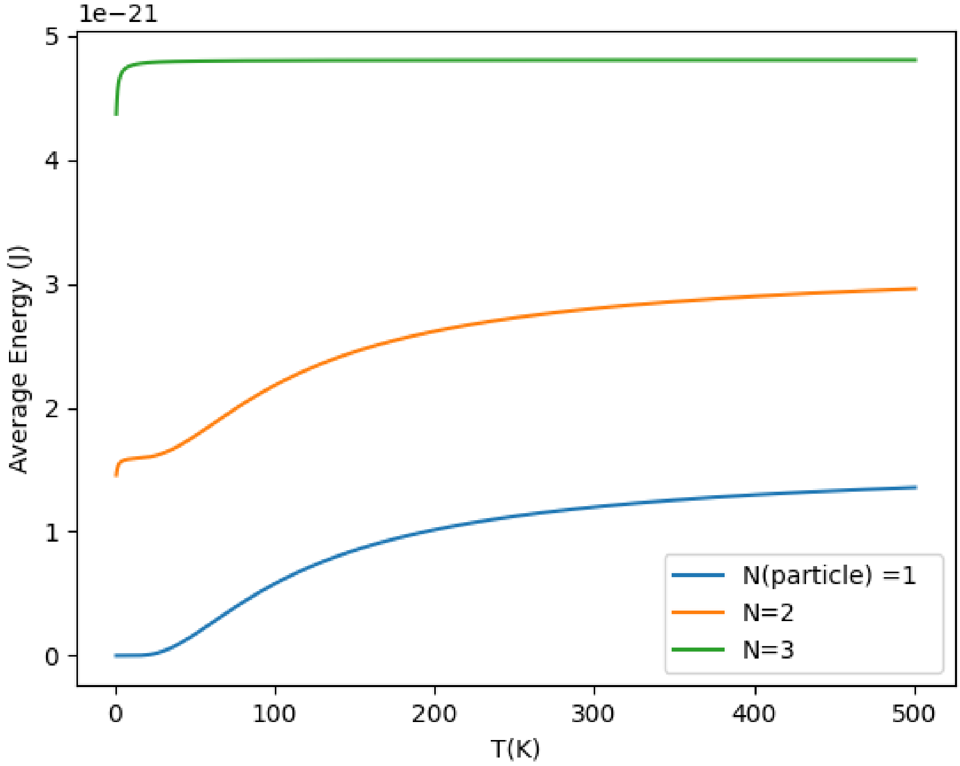

```
# using BE statistics
import numpy as np
import matplotlib.pyplot as plt

#constants , 1eV = e * (1 J)
k = 1.381*10**(-23) # J/K
e = 1.602 * 10**(-19) # coulomb

# create temperature points
Tmin = 1
Tmax = 500
dT = 0.1
T =  np.arange(Tmin,Tmax + 1,dT)

N = np.array([1,2,3]) # number of paticles

n = 3  # number of energy levels
dE = 0.01 # ev
E0 = 0 # ground state energy
E1 = E0 + dE # first state nergy
E2 = E0 + 2*dE # second state nergy
```

```python
# Z_list to store value of partition fucntion
# at differnt temperatures for differnt number of
# particles

Z_list = np.zeros((len(N),len(T))) # row,col


def Z1(t):
    # partition funciton for 1 particle and 3 levels
    # t is temp
    b = (e/k)/t
    return np.e**(-E0*b) + np.e**(-E1*b) + np.e**(-E2*b)

def Z2(t):
    # partitio function for 2 partilces and 3 levels
    # t is temp
    b = (e/k)/t
    return (np.e**(-2*E0*b) + np.e**(-2*E1*b) + np.e**(-2*E2*b)+
            np.e**(-(E0 + E1)*b) + np.e**(-(E0 + E2)*b) +
            np.e**(-(E1 + E2)*b) )

def Z3(t):
    # partitio function for 3 partilces and 3 levels
    # t is temp
    b = (e/k)/t
    return (np.e**(-3*E0*b) + np.e**(-3*E1*b) + np.e**(-3*E2*b) +
            np.e**(-(2*E0 + E1)*b) + np.e**(-(2*E0 + E2)*b)    +
            np.e**(-(E0 + 2*E1)*b) + np.e**(-(2*E1 + E2)*b)    +
            np.e**(-(E0 + 2*E2)*b) + np.e**(-(E1    + 2*E2)*b) +
            np.e**(-(E0 + E1 + E2 )*b) )

for j in range(len(T)):

    # row0 stores partition function for 1 particle
    # row1 for 2 paticles and row2 for 3 .
    Z_list[0,j] = Z1(T[j])
    Z_list[1,j] = Z2(T[j])
    Z_list[2,j] = Z3(T[j])


# 3n3rgy
T1 = T[:-1]
U = k*T1*T1*np.diff(np.log(Z_list))/dT

# specific heat capacity
T2 = T1[:-1]
Cv = np.diff(U)/dT

# Helomholtz free energy
```

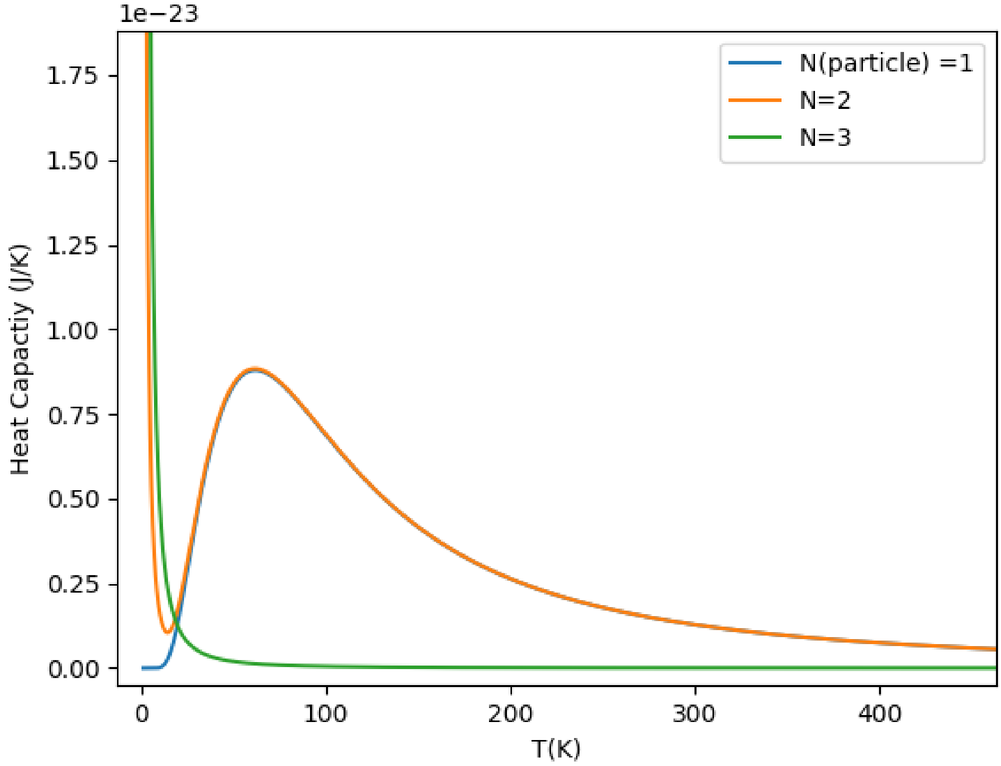
```python
F = -k*T*np.log(Z_list)

#enrtopy
S = -np.diff(F)/dT

#plot
plt.figure(1)
plt.xlabel('T(K)')
plt.ylabel('Average Energy (J)')
plt.plot( T1, U[0] , label ='N(particle) =1  ')
plt.plot( T1, U[1] , label ='N=2')
plt.plot( T1, U[2] , label ='N=3')
plt.legend()

plt.figure(2)
plt.xlabel('T(K)')
plt.ylabel('Heat Capactiy (J/K)')
plt.plot(T2,Cv[0] , label ='N(particle) =1')
plt.plot(T2,Cv[1] , label ='N=2')
plt.plot(T2,Cv[2] , label ='N=3')
plt.legend()


plt.figure(3)
plt.xlabel('T(K)')
plt.ylabel(' Entropy (J/K)')
plt.plot(T1,S[0] , label ='N(particle) =1')
plt.plot(T1,S[1] , label ='N=2')
plt.plot(T1,S[2] , label ='N=3')
plt.legend()

plt.figure(5)
plt.xlabel('T(K)')
plt.ylabel(' Partiton  Function Z')
plt.plot(T,Z_list[0], label = 'N(particle) =1')
plt.plot(T,Z_list[1], label = 'N =2')
plt.plot(T,Z_list[2], label = 'N =3')
plt.legend()
plt.show()

# using fermi dirac statistic
import numpy as np
import matplotlib.pyplot as plt

#constants , 1eV = e * (1 J)
k = 1.381*10**(-23) # J/K
e = 1.602 * 10**(-19) # coulomb

# create temperature points
Tmin = 1
```

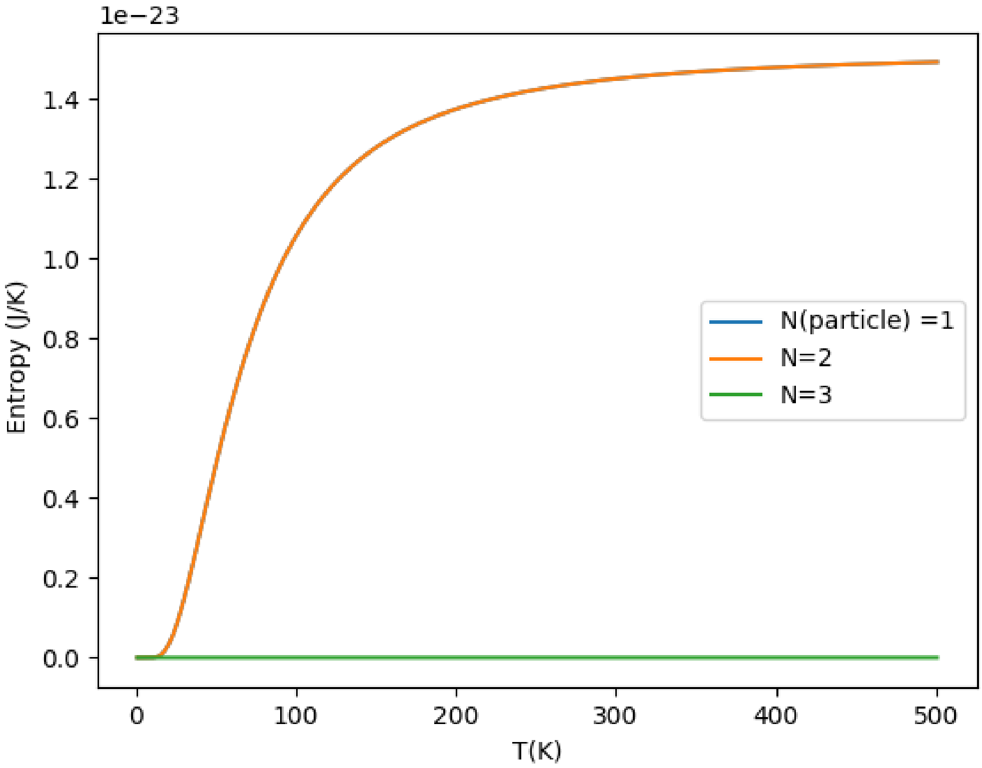
```
Tmax = 500
dT = 0.1
T =  np.arange(Tmin,Tmax + 1,dT)

N = np.array([1,2,3]) # number of paticles

n = 3  # number of energy levels
dE = 0.01 # ev
E0 = 0 # ground state energy
E1 = E0 + dE # first state nergy
E2 = E0 + 2*dE # second state nergy

# Z_list to store value of partition fucntion
# at differnt temperatures for differnt number of
# particles

Z_list = np.zeros((len(N),len(T))) # row,col


def Z1(t):
    # partition funciton for 1 particle and 3 levels
    # t is temp
    b = (e/k)/t
    return np.e**(-E0*b) + np.e**(-E1*b) + np.e**(-E2*b)

def Z2(t):
    # partitio function for 2 partilces and 3 levels
    # t is temp
    b = (e/k)/t
    return (np.e**(-(E0 + E1)*b) + np.e**(-(E0 + E2)*b) +
            np.e**(-(E1 + E2)*b) )

def Z3(t):
    # partitio function for 3 partilces and 3 levels
    # t is temp
    b = (e/k)/t
    return np.e**(-(E0 + E1 + E2 )*b)

for j in range(len(T)):

    # row0 stores partition function for 1 particle
    # row1 for 2 paticles and row2 for 3 .
    Z_list[0,j] = Z1(T[j])
    Z_list[1,j] = Z2(T[j])
    Z_list[2,j] = Z3(T[j])


# Energy
T1 = T[:-1]
```

```
U = k*T1*T1*np.diff(np.log(Z_list))/dT

# specific heat capacity
T2 = T1[:-1]
Cv = np.diff(U)/dT

# Helomholtz free energy
F = -k*T*np.log(Z_list)

# enrtopy
S = -np.diff(F)/dT

# plot
plt.figure(1)
plt.xlabel('T(K)')
plt.ylabel('Average Energy (J)')
plt.plot( T1, U[0] , label ='N(particle) =1  ')
plt.plot( T1, U[1] , label ='N=2')
plt.plot( T1, U[2] , label ='N=3')
plt.legend()

plt.figure(2)
plt.xlabel('T(K)')
plt.ylabel('Heat Capactiy (J/K)')
plt.plot(T2,Cv[0] , label ='N(particle) =1')
plt.plot(T2,Cv[1] , label ='N=2')
plt.plot(T2,Cv[2] , label ='N=3')
plt.legend()


plt.figure(3)
plt.xlabel('T(K)')
plt.ylabel(' Entropy (J/K)')
plt.plot(T1,S[0] , label ='N(particle) =1')
plt.plot(T1,S[1] , label ='N=2')
plt.plot(T1,S[2] , label ='N=3')
plt.legend()

plt.figure(5)
plt.xlabel('T(K)')
plt.ylabel(' Partiton  Function Z')
plt.plot(T,Z_list[0], label = 'N(particle) =1')
plt.plot(T,Z_list[1], label = 'N =2')
plt.plot(T,Z_list[2], label = 'N =3')
plt.legend()
plt.show()
```

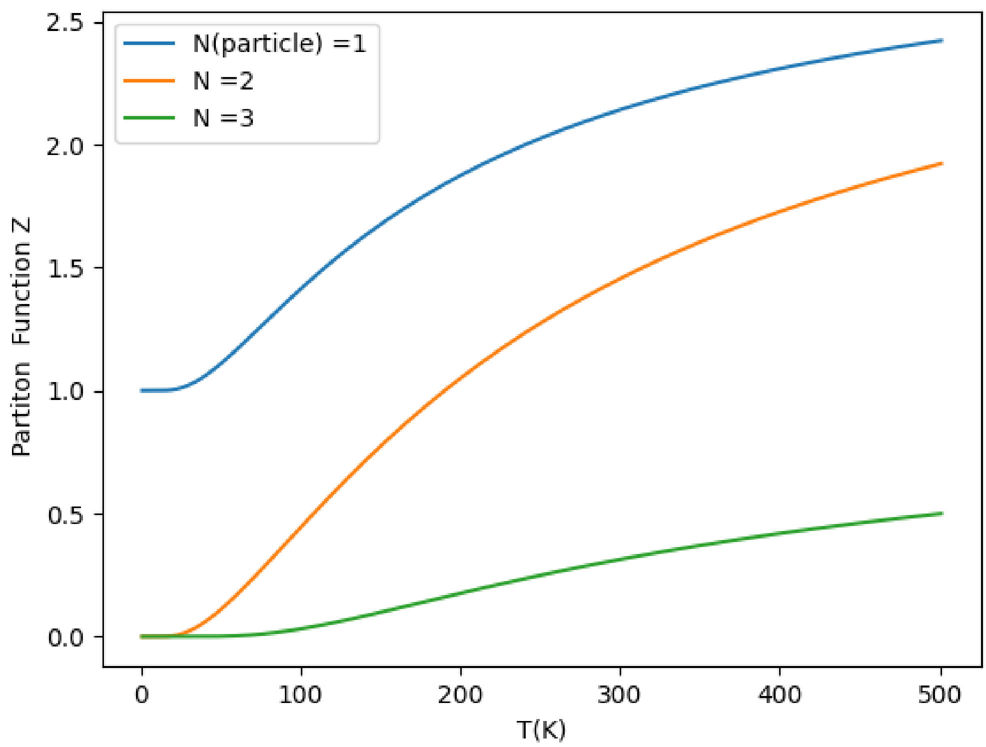


1e−21
5
4
3
2
1
0
Average Energy (J)
0
100
200
300
400
500
T(K)
N(particle) =1
N=2
N=3


1e−23
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Heat Capactiy (J/K)
0
100
200
300
400
T(K)
N(particle) =1
N=2
N=3



1e−23
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Entropy (J/K)
N(particle) =1
N=2
N=3
0
100
200
300
400
500
T(K)

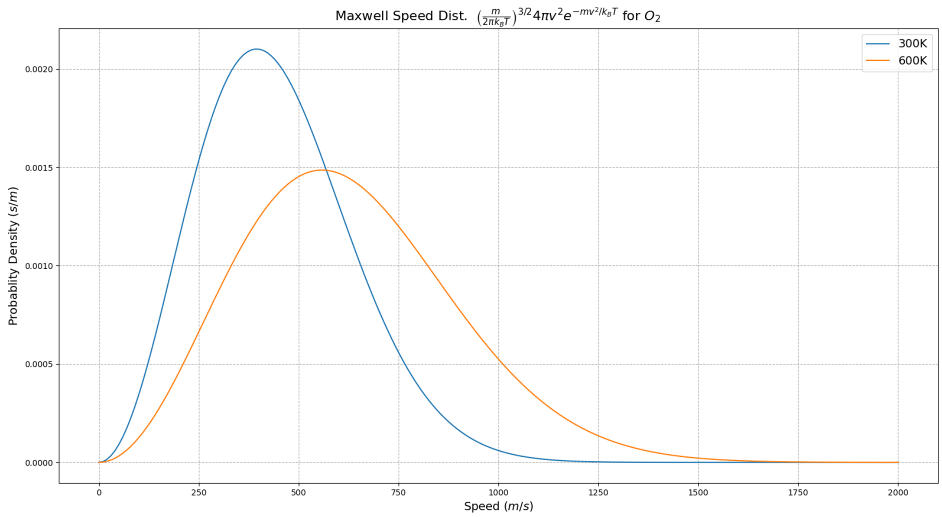


Program 3. Plot the Maxwell speed distribution function at different temperatures in a 3-dimension system. Calculate the average speed, root mean square and most probable speed.

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

#constants actual

h = 6.626*(10**(-34)) # J-s
k = 1.381*10**(-23) # J/K
m = 32*1.66*10**(-27) # kg
pi= np.pi
e = np.e
T1 = 200 # K
T2 = 300 # K
T3 = 600 # K

def maxwell_speed_dist(v,T):
    '''
    v : speed
    T: temprature
```

```
    '''
    v0 = 4*pi*np.power( (m / (2*pi*k*T) ) ,3/2)
    return v0 * (v**2) *  e**( -m*(v**2)/(2*k*T) )

def integrate_n_sort(x,y):
    '''
    Intergrate and sort
    x : array of x values, not used here
    y : array of y values, to be integrated

    '''
    dx = x[1]-x[0]
    y_avg = 0
    y_sq_avg = 0
    j=0
    y_new = 0
    y_max = 0
    max_at = 0 # store the index at whihc y is max
    for i in y:
        y_avg = y_avg + x[j]*i*dx
        y_sq_avg = y_sq_avg + (x[j]**2)*i*dx

        y_new=  i
        if(y_new>y_max):
            y_max=y_new
            max_at = j

        j += 1
    v_max = x[max_at]
    y_rms = np.sqrt(y_sq_avg)

    return y_avg,y_rms,v_max,max_at


# Calculate arrays
v_array = np.linspace(0,2000,200)
f_2 = maxwell_speed_dist( v_array,T2) # dist of speed
f_3 = maxwell_speed_dist( v_array,T3) # dist of speed

# find avg, rms and max velocity
v_avg , v_rms ,v_max , i = integrate_n_sort(v_array,f_2)
print('At T = 300K')
print('v_avg = ', round(v_avg,3),'v_rms=',round(v_rms,3),'v_max=',round(v_max,3))

v_avg , v_rms ,v_max , i = integrate_n_sort(v_array,f_3)
print('At T = 600K')
print('v_avg = ', round(v_avg,3),'v_rms=',round(v_rms,3),'v_max=',round(v_max,3))

eqn = r'$ \left( \frac{m}{2\pi k_B T} \right)^{3/2} 4 \pi v^2 e^{-mv^2/k_B T} $'
```

```
# plt.text()
#plot
plt.plot( v_array, f_2  ,label='300K')
plt.plot( v_array, f_3  ,label='600K')

plt.xlabel(r'Speed $(m/s)$',fontsize=14)
plt.ylabel(r'Probablity Density $(s/m)$',fontsize=14)
plt.title(r'Maxwell Speed Dist.  $\left( \frac{m}{2\pi k_B T} \right)^{3/2} 4 \pi v^2 e^{-mv^2/k_
plt.grid(True,ls='--')
plt.legend(fontsize=14)
plt.show()
```

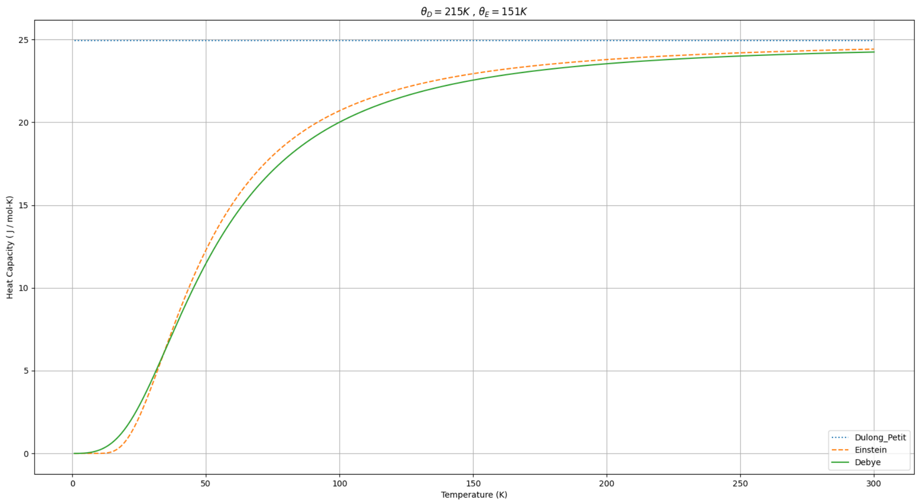


Program 4. Plot Specific Heat of Solids w.r.t temperature a) Dulong-Petit law, b) Einstein distribution function c) Debye distribution function

```
from cProfile import label
from scipy.integrate import quad
import numpy as np
import matplotlib.pyplot as plt

# constants
e = np.e
R = 8.314 # J/mol-K
T_D = 215 # debye temp of copper
T_E = 151 # einsten temp of copper


def Dulong(T):
    return np.array(len(T)*[3*R])
```

```python
def Einstein(T):
    '''
    Einstein heat capacity for a mole of any solid
    T: temprature at which to evaluate heat capacity
    '''
    C = 3*R*(T_E/T)**2
    C = C * ( e**( T_E/T ) ) / (e**( T_E/T ) - 1)**2
    return C

def Debye(T):
    '''
    Debye heat capacity for a mole of any solid
    T: temprature at which to evaluate heat capacity
    uses quadrature to eavluate integral
    '''
    f = lambda x :(x**4)*(e**x)/(e**x - 1)**2
    c_array = np.array([])
    # extract the value of integral
    for t in T:
        C = 9*R*(t/T_D)**3
        value = C*quad( f , 0.1 , T_D/t)[0]
        c_array = np.append(c_array,value)

    return c_array


# define temprature array
T_array = np.linspace(0.8,300,1000)

# store heat capacity

# Dulong_Petit value
C_DP  = Dulong(T_array)

# Einstein  Value
C_E = Einstein(T_array)

# Debye value
C_D = Debye(T_array)

plt.plot(T_array,C_DP,label='Dulong_Petit',ls=':')
plt.plot(T_array,C_E,label='Einstein',ls='--')
plt.plot(T_array,C_D,label='Debye')
plt.legend()
plt.title(r'$\theta_D = 215 K $ , $\theta_E = 151 K $ ')
plt.xlabel('Temperature (K)')
plt.ylabel('Heat Capacity ( J / mol-K)')
plt.grid(True)
plt.show()
```

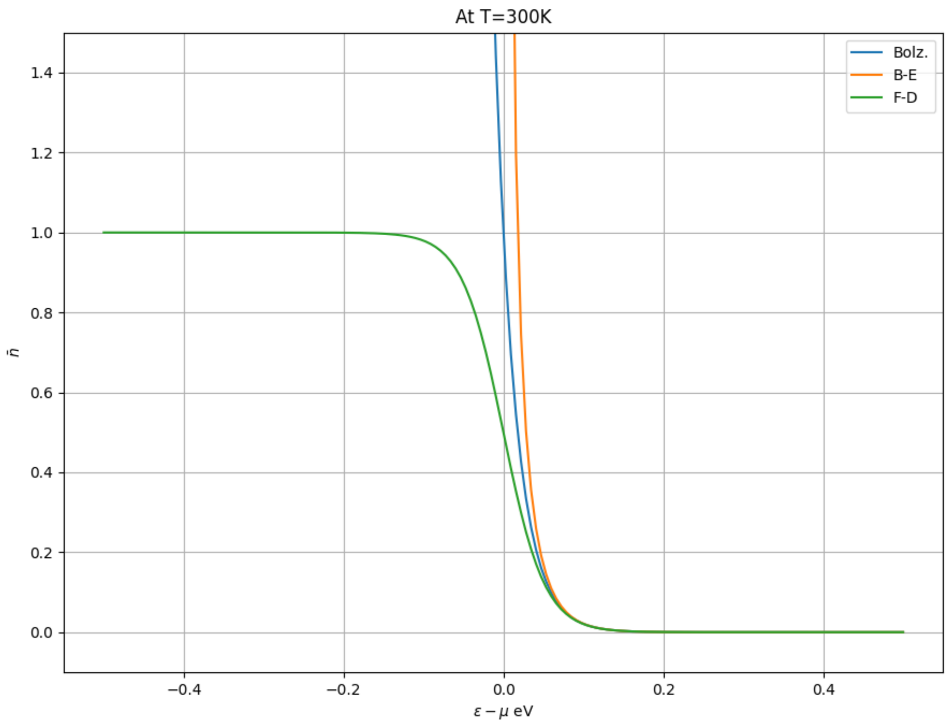
```
# print(Einstein(1))
# print(C_E)
# print(C_DP)
# print(C_E)
```

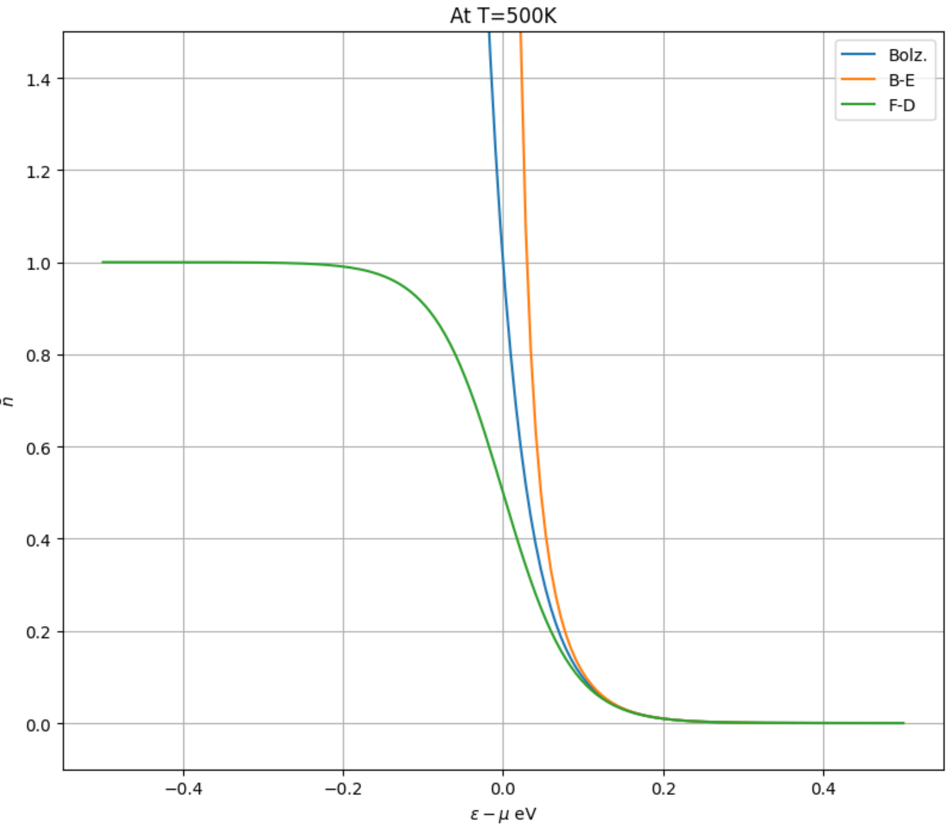


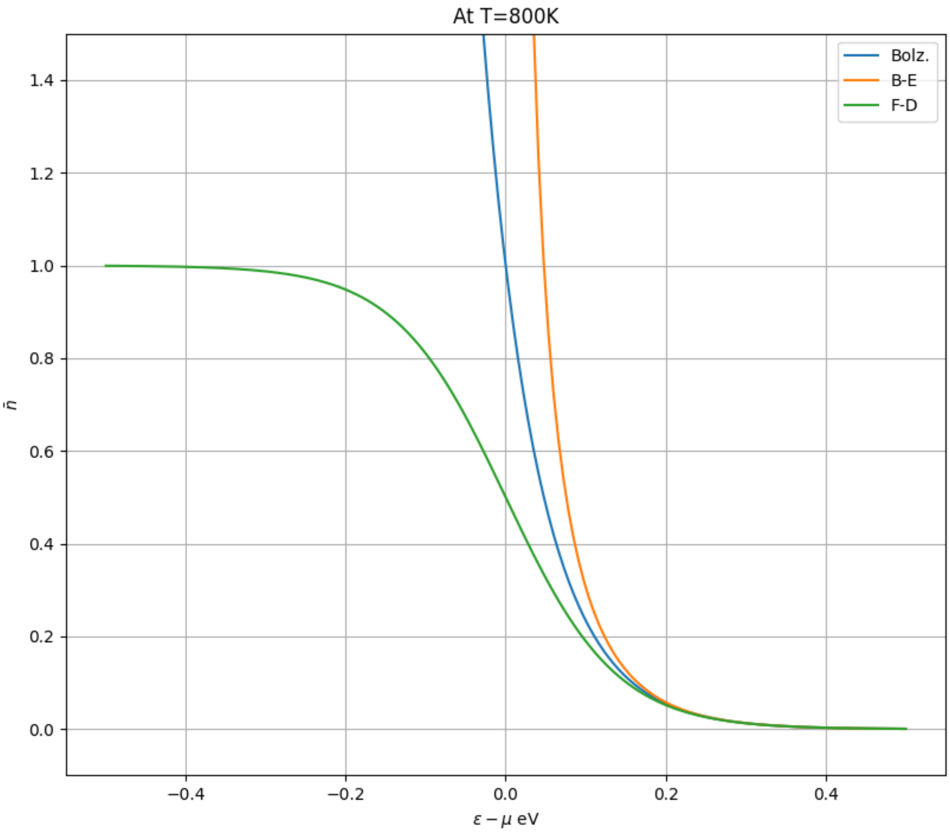
Program 5. Plot the following functions with energy at different temperatures a) Maxwell-Boltzmann distribution b) Fermi-Dirac distribution c) Bose-Einstein distribution

```
import numpy as np
import matplotlib.pyplot as plt
e = np.e
k = 8.617*10**(-5)

T1 = 300    # K
T2 = 500   # K
T3 = 800 # K

def Boltzmann_stat(x,T):
    a=k*T
    return e**(-x/a)

def Bose_Einstein_stat(x,T):
    a=k*T
    return 1/(e**(x/a) - 1)

def Fermi_Dirac_stat(x,T):
    a=k*T
    return 1/(e**(x/a) + 1)
```

```python
# x = enegy minus chemical potential
len = 160
x = np.linspace(-0.5,0.5,len)

plt.figure(0)
plt.title('At T=300K')
plt.plot(x,Boltzmann_stat(x,T1),label='Bolz.')
plt.plot(x[len//2 + 1 : ],Bose_Einstein_stat(x[len//2 + 1 : ],T1),label='B-E')
plt.plot(x,Fermi_Dirac_stat(x,T1),label='F-D')
plt.ylim(-0.1,1.5)
plt.grid(True)
plt.xlabel(r'$\epsilon - \mu$ eV')
plt.ylabel(r'$\bar{n}$')
plt.legend()


plt.figure(1)
plt.title('At T=500K')
plt.plot(x,Boltzmann_stat(x,T2),label='Bolz.')
plt.plot(x[len//2 + 1 : ],Bose_Einstein_stat(x[len//2 + 1 : ],T2),label='B-E')
plt.plot(x,Fermi_Dirac_stat(x,T2),label='F-D')
plt.ylim(-0.1,1.5)
plt.grid(True)
plt.legend()
plt.xlabel(r'$\epsilon - \mu$ eV')
plt.ylabel(r'$\bar{n}$')



plt.figure(2)
plt.title('At T=800K')
plt.plot(x,Boltzmann_stat(x,T3),label='Bolz.')
plt.plot(x[len//2 + 1 : ],Bose_Einstein_stat(x[len//2 + 1 : ],T3),label='B-E')
plt.plot(x,Fermi_Dirac_stat(x,T3),label='F-D')
plt.ylim(-0.1,1.5)
plt.grid(True)
plt.legend()


plt.xlabel(r'$\epsilon - \mu$ eV')
plt.ylabel(r'$\bar{n}$')
plt.show()
```


At T=300K
Bolz.
B-E
F-D
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
$\bar{n}$
−0.4
−0.2
0.0
0.2
0.4
$\varepsilon - \mu$ eV


At T=500K
Bolz.
B-E
F-D
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
$\bar{n}$
−0.4
−0.2
0.0
0.2
0.4
$\varepsilon - \mu$ eV



Program 6. Plot Planck's law of Black body radiation w.r.t. wavelength/frequency at different temperatures. Compare it with Rayleigh-Jeans Law and Wien's distribution law for a given temperature..

```
#To do:
# Plot enrgy/vol-frq
# Plot energy/vol

from cProfile import label
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp

#constants actual
c = 3*10**8 # m/s
h = 6.626*(10**(-34)) # J-s
k = 1.381*10**(-23) #J/K
lamb_max = 2000 # nm
```

```python
nu_max = 10
beta = h*c/k
T1 = 2000 # K
T2 = 4000 # K
T3 = 6000 # K



def  planck_dist_lamb(lamb,T):
    cons = 8*np.pi*h
    return cons/((lamb**3)*(np.exp(h*c/(lamb*k*T))-1))

def  planck_dist_nu(nu,T):
    cons = 8*np.pi*h
    return (cons*(nu/c)**3)/(np.exp(h*nu/(k*T))-1)

def r_j_dist_nu(nu,T):
    return (8*np.pi*k*T*nu**2)/c**3

def wein_dist_nu(nu,T):
    cons = 8*np.pi*h
    return (cons*(nu/c)**3)/np.exp(h*nu/(k*T))
# lamb_array = np.linspace(10**-2,lamb_max,1000)


#------------Placnck--------------------------------------------------
nu_array = 10**14*np.linspace(0.001,nu_max,100)
plt.xlim((0,nu_max))
plt.ylim((0,2))

#----------------------Planck's law--------------
# plt.plot(nu_array/10**14,10**15*planck_dist_nu(nu_array,T1))
# plt.plot(nu_array/10**14,10**15*planck_dist_nu(nu_array,T2))
plt.plot(nu_array/10**14,10**15*planck_dist_nu(nu_array,T3),label='Planck \' s Law')
# -----------------------------------------------

#-----------Rayleigh_jeans----------------------
# plt.plot(nu_array/10**14,10**15*r_j_dist_nu(nu_array,T1),ls='--')
# plt.plot(nu_array/10**14,10**15*r_j_dist_nu(nu_array,T2),ls='--')
plt.plot(nu_array/10**14,10**15*r_j_dist_nu(nu_array,T3),ls='--',label='Rayleigh-Jeans Law')

#----------Wein-------------------------------------
# plt.plot(nu_array/10**14,10**15*wein_dist_nu(nu_array,T1),ls=':')
# plt.plot(nu_array/10**14,10**15*wein_dist_nu(nu_array,T2),ls=':')
plt.plot(nu_array/10**14,10**15*wein_dist_nu(nu_array,T3),ls=':',label='Weins\' Law')
#-------------------------------------------------


# plt.rcParams['text.usetex'] = True
```

```
plt.rcParams['font.size'] = '13'
plt.grid(True)
plt.xlabel(r'$\nu 10^{-14}  $ frequency',fontsize=14)
plt.ylabel(r'$u(\nu ,T) \cdot 10^{15}  \ \ J / {m^3 s} $  ',fontsize=14)
plt.legend()
plt.show()
```