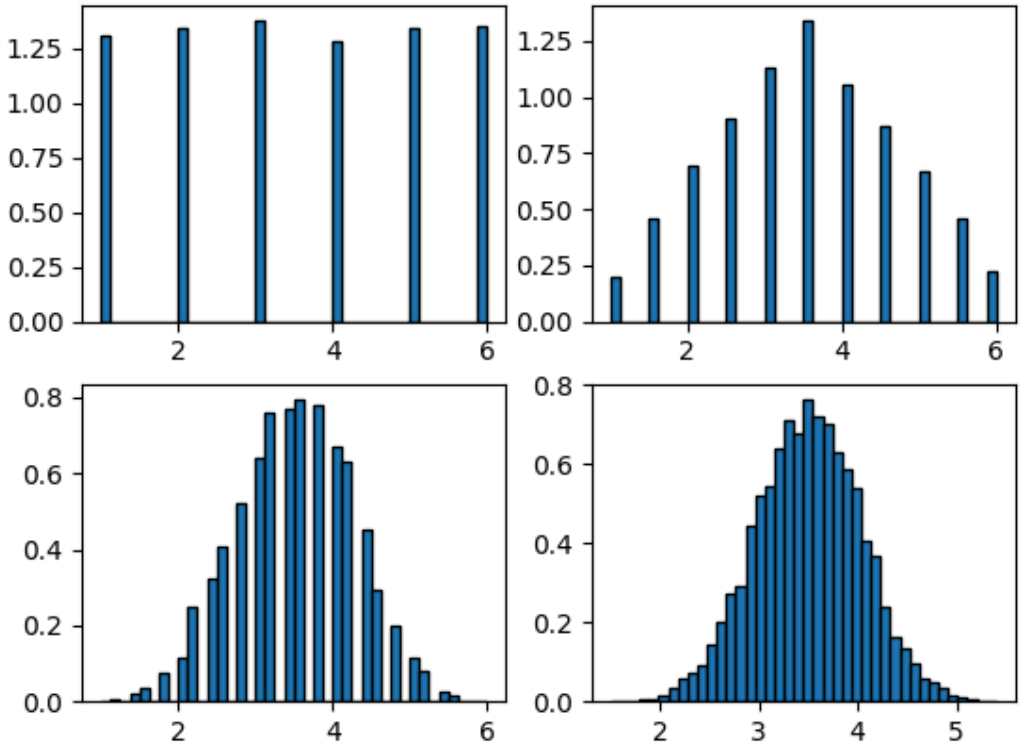
Statistical Mechanics Python Codes

Amitava Moitra

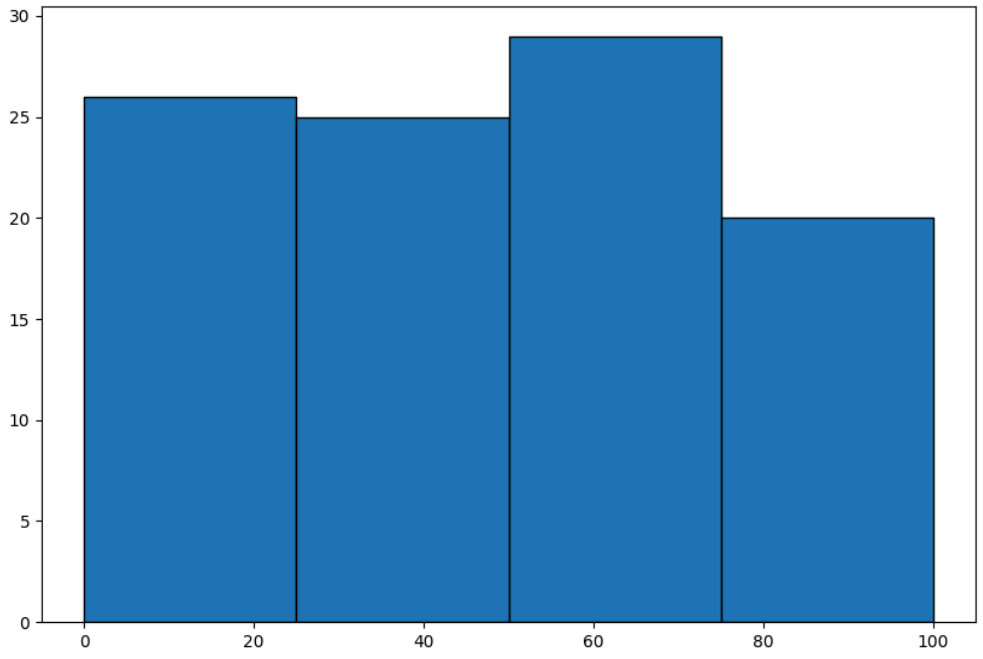
Problem of "Throw Dice"

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

x = lambda n: np.mean(np.random.randint(1,7, size=n))
k=1
for n in [1, 2, 5, 10]:
    xx = [x(n) for i in range(10000)]
    plt.subplot(2,2,k)
    k=k+1
    plt.hist(xx, bins=40, density = True, edgecolor='black')
plt.show()
```



Problem of "Student Results Histogram"

```
import numpy as np
import matplotlib.pyplot as plt
import random
a=[]
for i in range(0, 100):             #Number of students
   b=random.randint(5, 95)         #Marks Obtained by students between 5
to 95
   a.append(b)

fig, ax= plt.subplots(figsize=(10,7))
ax.hist(a, bins=[0,25, 50, 75, 100], ec='black')
plt.show()
```

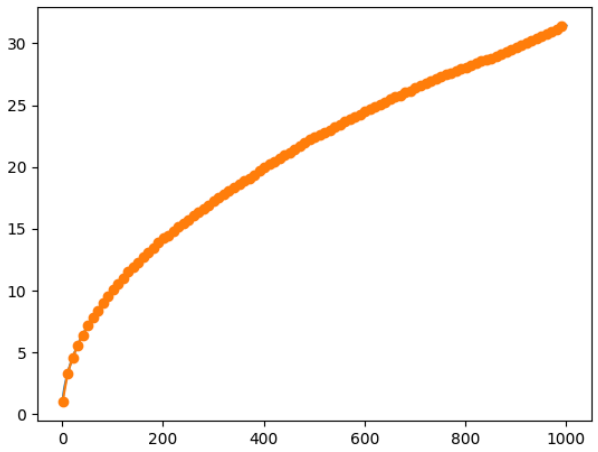


Problem of "Specific Heat Plotting"

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps
k=1.38e-23
N=6e23
Te=240
Td=343

T=np.linspace(10,2*Td, 1000)
x=Te/T
y=Td/T
def Cvdp(T): return 3*N*k*T**0
def Cve(T): return 3*N*k*(Te/T)**2*np.exp(Te/T)/(np.exp(Te/T)-1)**2

#def F(T): return (Td/T)**4*np.exp(Td/T)/(np.exp(Td/T)-1)**2
#I=simps(F(T), 0,y)
#def Cvd(T): return 9*N*k*I*(T/Td)**3

plt.plot(T, Cvdp(T), color='red')
plt.plot(T, Cve(T), color='blue')
#plt.plot(T, Cvd(y, T), color='black')
plt.show()
```

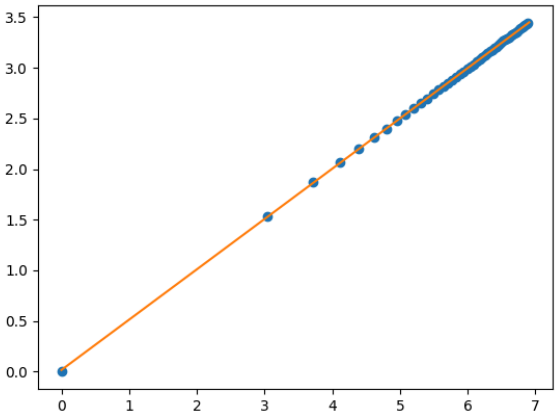
Problem of Random Walk in 1D

```
#### This is ONE DIMENSIONAL Random Walk#####x=
import numpy as np
import matplotlib.pyplot as plt
#x= 2*np.random.randint(0,2, size=10)-1
#xx=np.cumsum(x)
#plt.plot(xx)
#plt.show()
N, T =10000, 1000 #For N such walks of T times long
t=range(1, T+1)
walks=2*np.random.randint(0,2, size=(N,T))-1
pos=np.cumsum(walks, axis=1)
pos_sq=pos**2
pos_meansq=np.mean(pos_sq, axis=0)
rms=np.sqrt(pos_meansq)
#plt.plot(t, rms)
#plt.plot(t[::50], rms[::50], '-o')
t=np.log(t)
rmsl=np.log(rms)
#plt.plot(t,rmsl)
import numpy.polynomial.polynomial as poly
poly.polyfit(t, rmsl,1)
coeffs=poly.polyfit(t,rmsl, 1)
#print (coeffs)
rmsfit=poly.polyval(t, coeffs)
plt.plot(t[::20],rmsl[::20],'o')
plt.plot(t,rmsfit)
plt.show()
```

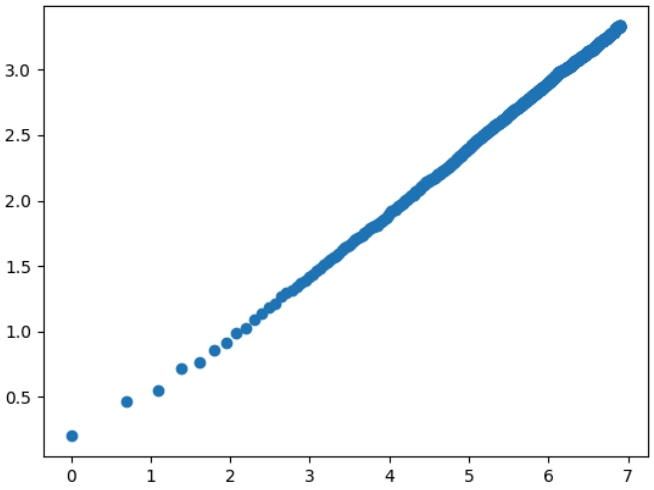


Problem of Random Walk in 2D

```
import numpy as np
import matplotlib.pyplot as plt
from random import choice

x, y =0, 0
X, Y =[], []
for time in range(5000):
    dx,dy=choice([(1,0), (-1,0),
(0,1), (0,-1)])
    x,y=x+dx, y+dy
    X.append(x)
    Y.append(y)

#plt.plot(X,Y)
#plt.show()
```

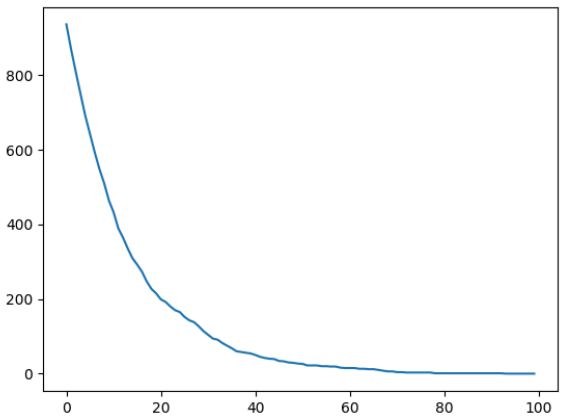


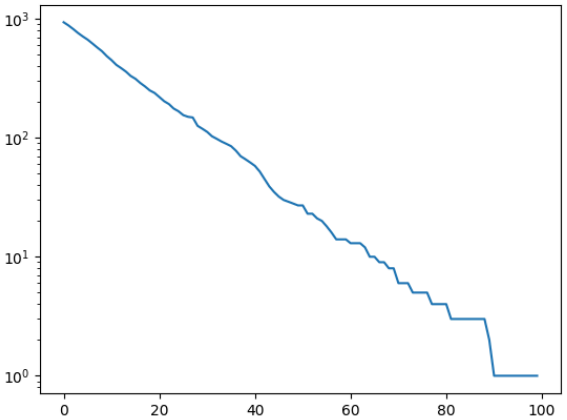

```
def walk(steps):
    x,y=0,0
    pos=[]
    for i in range(steps):
        dx,dy=choice([(1,0), (-1,0), (0,1), (0,-1)])
        x,y=x+dx, y+dy
        pos.append(np.sqrt(x*x+y*y))
    return pos
steps =1000
config =1000

walks = np.array([walk(steps) for i in range(config)])

m_walk=np.mean(walks, axis=0)  # average at each time step
t=range(steps)
#plt.plot(t,m_walk)

t=np.log(t)
r=np.log(m_walk)
#plt.plot(t,r, 'o')

import numpy.polynomial.polynomial as poly
coeffs=poly.polyfit(t,r,1)
#print(coeffs)
#rfit=poly.polyval(t[100:],
coeffs)
#plt.plot(t, rfit)

plt.show()
```

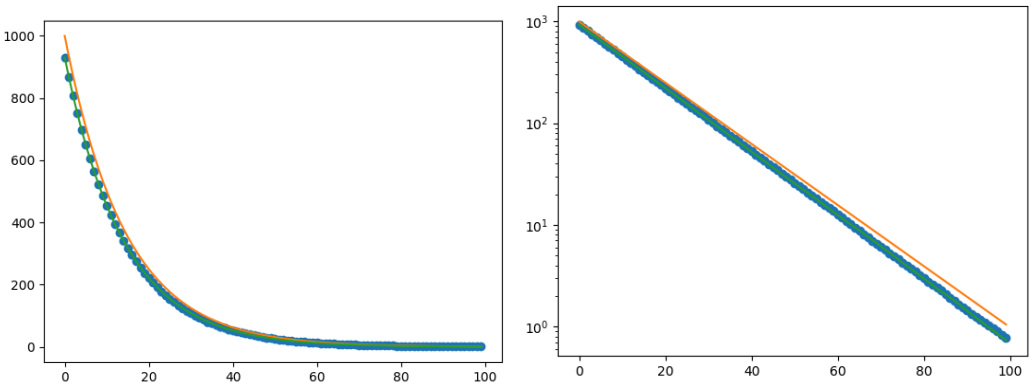


Problem of Radioactive Decay

```
import numpy as np
import matplotlib.pyplot as plt
t_half=10
lam=np.log(2)/t_half
p=lam       #decay probability
q=1-p       #survival probability
N =1000 # No of Atoms
def decay(N):
    population=[]
    for t in range(100):
        r=np.random.random(N)
        survive=np.sum(r<q)
        population.append(survive)
        N=survive
    return population
#plt.plot(range(100), decay(N))
#plt.show()

mean_decay=np.mean(np.array([decay(N) for i in range(1000)]), axis=0)
std_decay=np.mean(np.array([decay(N) for i in range (1000)]), axis =0)
plt.semilogy()
T=range(100)
plt.plot(T,mean_decay,'o')
exact=N*np.exp(-lam*T)
plt.plot(T, exact)
plt.plot(T, std_decay)
plt.show()
```





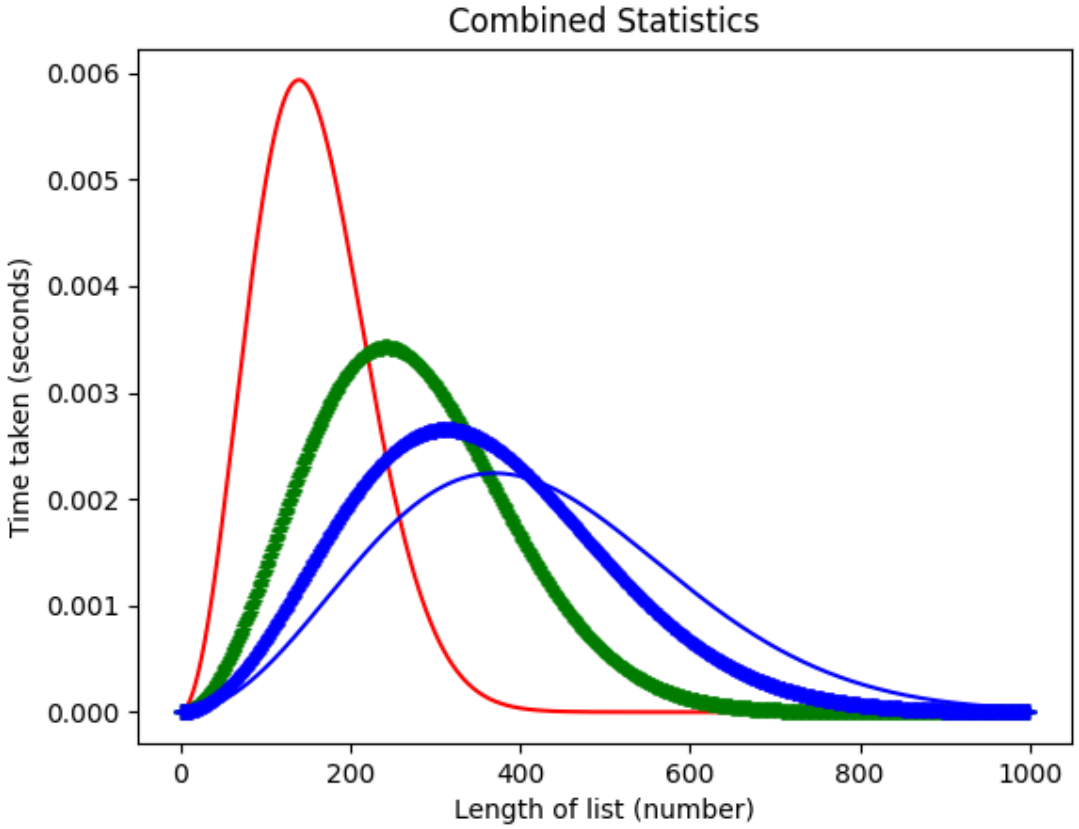


Random Walk Fundamentals

```
import numpy as np
import matplotlib.pyplot as plt
N, T = 10000, 1000
t = range(1, T+1)
walks = 2*np.random.randint(2, size =(N, T)) -1
#xx = np.cumsum(walks)
positions= np.cumsum(walks, axis=1)
pos_sq=positions**2
mean_pos_sq = np.mean(pos_sq, axis =0)
rms = np.sqrt(mean_pos_sq)
t=np.log(t)
rms= np.log(rms)

import numpy.polynomial.polynomial as poly
coeffs = poly.polyfit(t, rms, 1)
rmsfit = poly.polyval(t, coeffs)
print (coeffs)
plt.plot(t[::20], rms[::20], 'o', t, rmsfit, '-')

#plt.plot(positions[::100], '0')
plt.show()
```

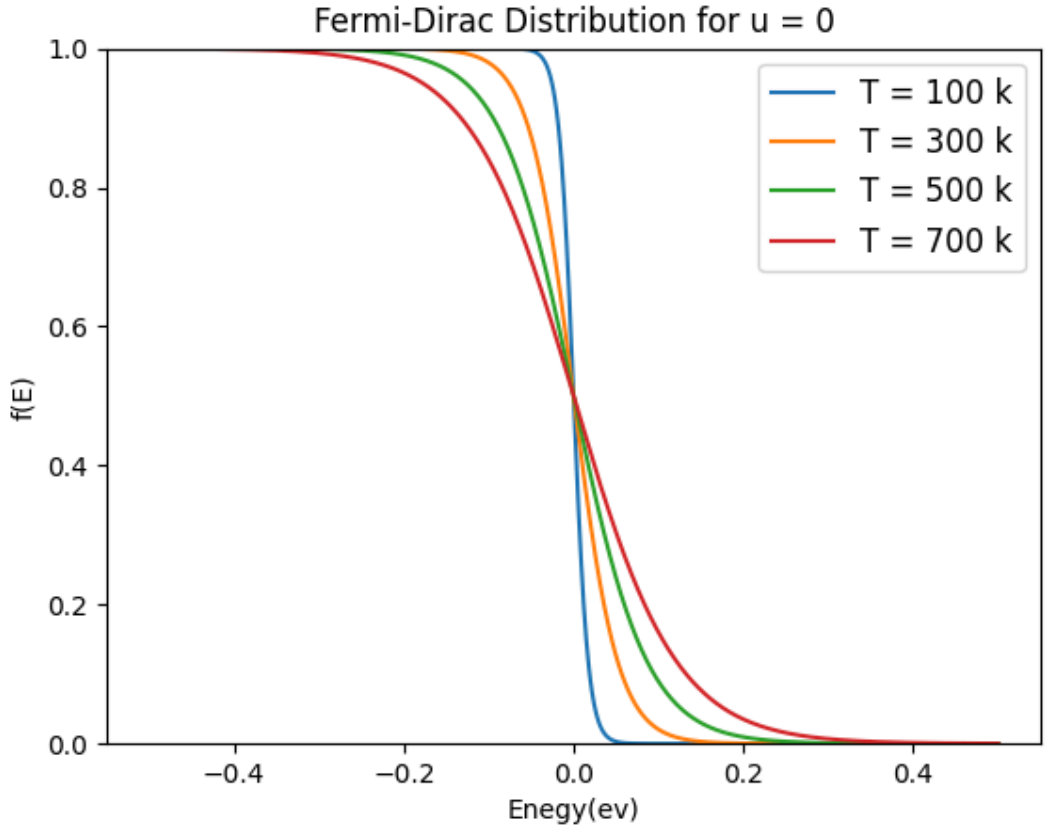
Problem of MB Distribution

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


def MB(v, m, T):
    kB = 1.38e-23
    return (m / (2 * np.pi * kB * T)) ** 1.5 * 4 * np.pi * v ** 2 *
np.exp(-m * v ** 2 / (2 * kB * T))


v = np.arange(0, 1000, 1)
amu = 1.66e-27
mass = 85 * amu
plt.plot(v, MB(v, mass, 100), "-r", label="T=100K")
plt.plot(v, MB(v, mass, 300), "g+", label="T=300K")
plt.plot(v, MB(v, mass, 500), 'b+')
plt.plot(v, MB(v, mass, 700), '-b')
plt.title("Combined Statistics")
plt.xlabel("Length of list (number)")
plt.ylabel("Time taken (seconds)")
plt.show()
```



Problem of FD Distribution

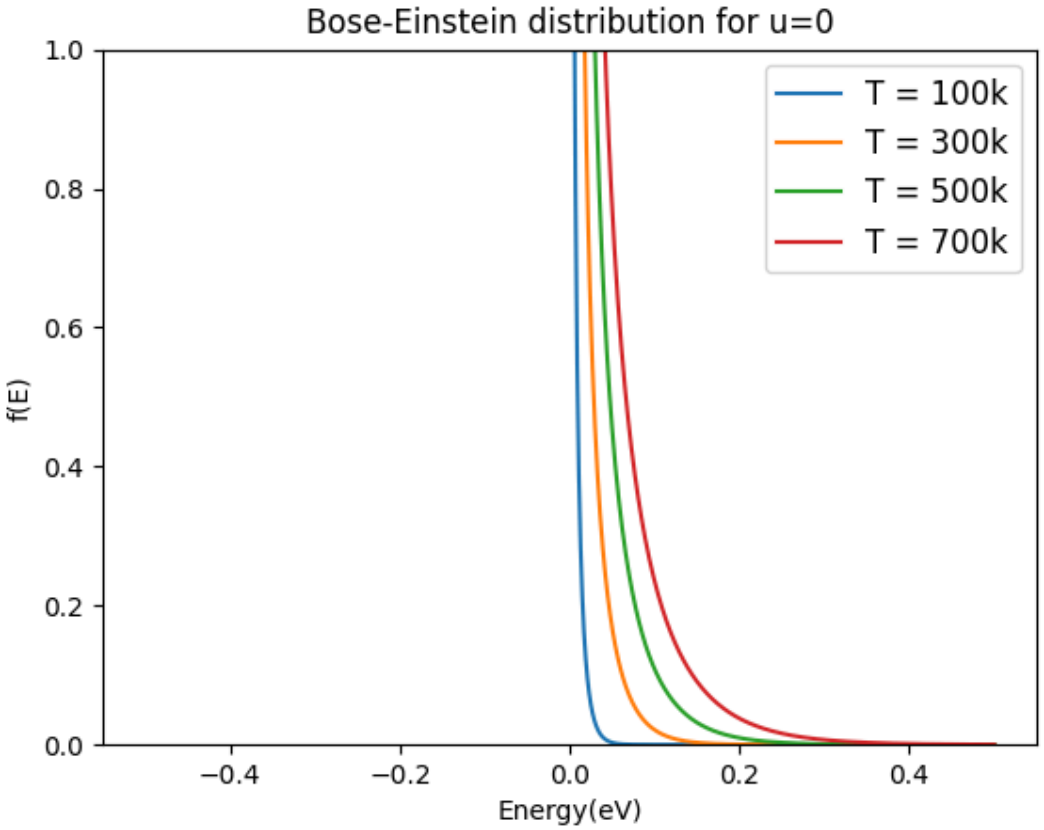
```
import numpy as np
import matplotlib.pyplot as plt

E = np. linspace(-0.5,0.5,1001) # energy range
e = 1.6e-19 # electric charge
k = 1.38e-23 # Boltzmann constant (Joule per kelvin)
u = 0 # considering chemical potential of the substance is zero
def Fn(T,a):
      return 1/((np.exp(((E-u)*e)/(k*T)))+a)

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':12})
plt.ylim(0,1)
plt.xlabel('Enegy(ev)')
plt.ylabel('f(E)')
plt.title("Fermi-Dirac Distribution for u = 0")
plt.show()
```



Problem of BE Distribution

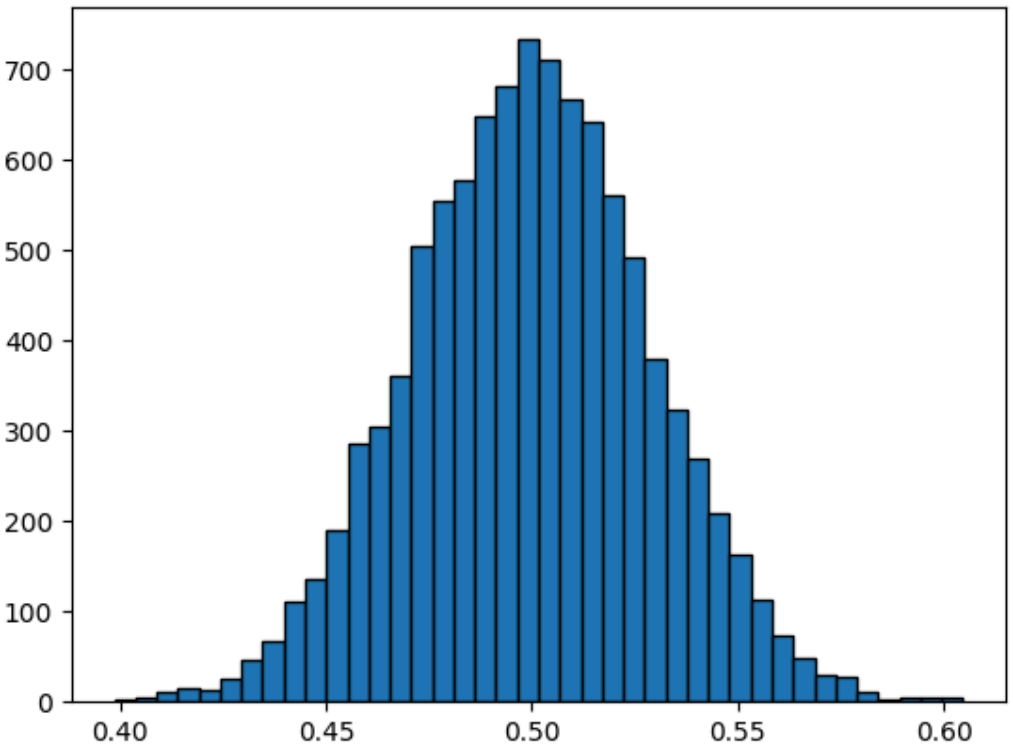
```
import numpy as np
import matplotlib.pyplot as plt

E = np.linspace(-0.5, 0.5, 1001) #energy range
e = 1.6e-19 #electric charge
k = 1.38e-23 #Boltzman constant(joule per kelvin)
u = 0 # considering chemical potential of the substance is zero
def Fn(T,a):
      return 1/((np.exp(((E-u)*e)/(k*T)))+a)
""" This is general equation,
for Maxwell-Boltzman distribution a=0,
for Bose-Einstein distribution a=-1,
and for Fermi Dirac a=+1 """
plt.plot(E,Fn(100,-1), label='T = 100k')
plt.plot(E,Fn(300,-1), label='T = 300k')
plt.plot(E,Fn(500,-1), label='T = 500k')
plt.plot(E,Fn(700,-1), label='T = 700k')
plt.legend(loc='best',prop={'size':12})
plt.ylim(0,1)
plt.xlabel('Energy(eV)')
plt.ylabel('f(E)')
plt.title('Bose-Einstein distribution for u=0')
```

```
plt.show()
```



Problem of Central Limit

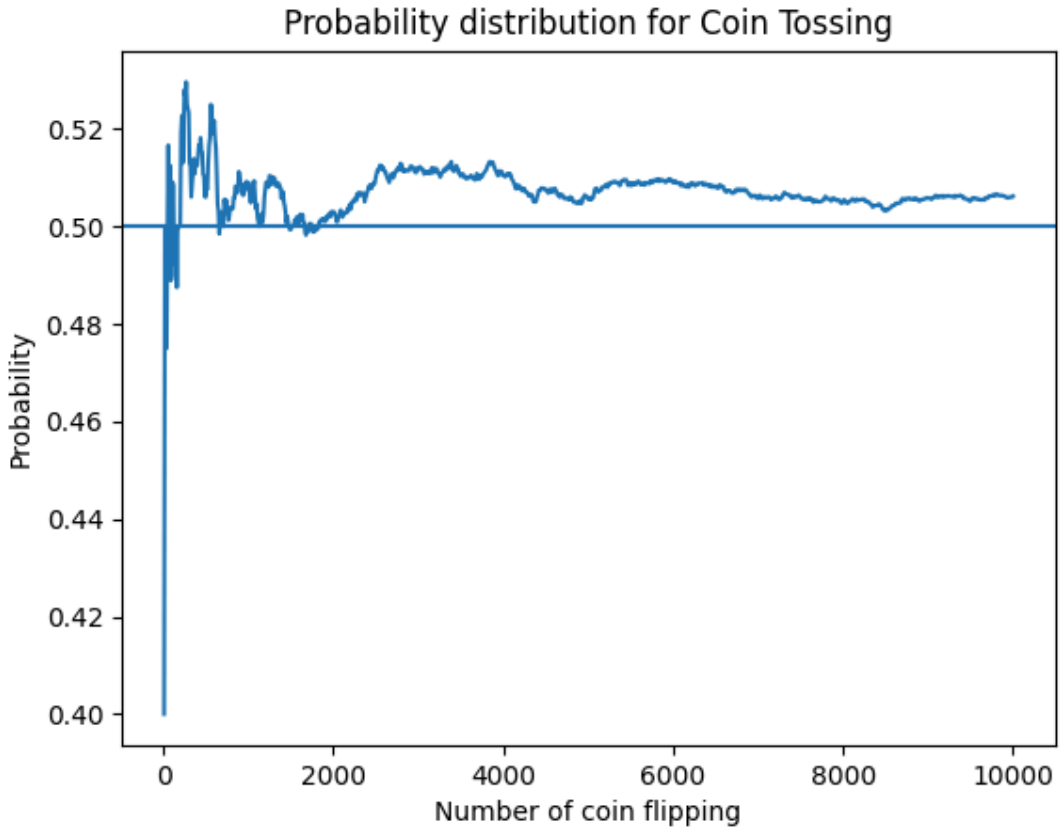
```
import numpy as np
import matplotlib.pyplot as plt
sample= lambda: np.mean(np.random.uniform(0,1, size=100))
x=[sample() for _ in range (10000)]
plt.hist(x, bins=40, edgecolor='black')
plt.show()
```



Problem of Coin Tossing

```
import numpy as np
import matplotlib.pyplot as plt
n=10000
x=np.random.randint(0,2, size=n) #where 1 is head
frac = lambda i: np.sum(x[:i])/i
N=range(10, n+10, 10)
y=[frac(i) for i in N]
plt.plot(N,y)
```

```
plt.axhline(0.5)
plt.title("Probability distribution for Coin Tossing")
plt.xlabel("Number of coin flipping")
plt.ylabel("Probability")
plt.show()
```



Exponent Variate

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

N=10000
u=np.random.uniform(0,1, size=N)
x=-np.log(1-u)
plt.hist(x,bins=50, edgecolor ='red')
```

```
plt.xlim(0,8)
plt.show()
freq, bins, extra=plt.hist(x, bins=50)
bin_mid=(bins[:-1]+bins[1:])*0.5
bin_width=bins[1]-bins[0]
freq_theo=np.exp(-bin_mid)*bin_width*N
plt.plot(bin_mid, freq_theo, '-o' )
plt.hist(x,bins=50, edgecolor ='k')
plt.xlim(0,8)
#plt.show()
```