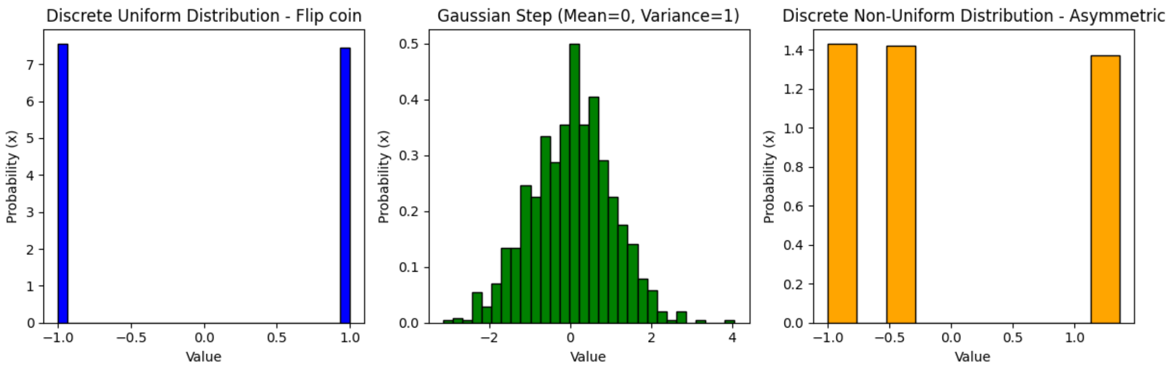
In [ ]:

5.1. Universality of random walks

5.1.a

In [ ]:

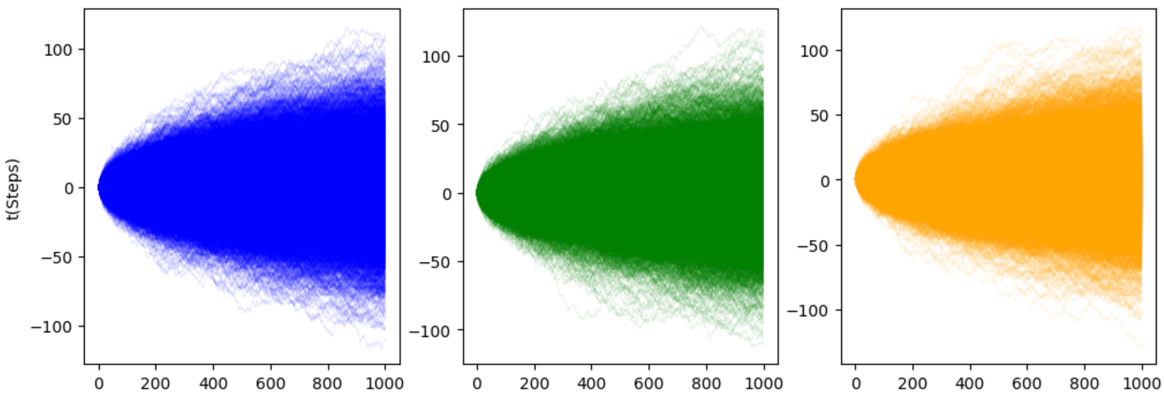
```
import numpy as np
import random
from matplotlib import pyplot as plt

sample_size = 1000

discrete_uniform_distribution = np.random.choice([-1, 1], size=s
gaus_distribution = np.random.normal(0, 1, size=sample_size)
discrete_non_uniform = np.random.choice([-1,
                                         (1 - np.sqrt(3)) / 2,
                                           (1 + np.sqrt(3)) / 2]

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
hist, bins, _ = plt.hist(discrete_uniform_distribution,
                         bins=30, color='blue', edgecolor='black
plt.title('Discrete Uniform Distribution - Flip coin')
plt.xlabel('Value')
plt.ylabel('Probability (x)')

plt.subplot(1, 3, 2)
plt.hist(gaus_distribution, bins=30, color='green', edgecolor='b
plt.title('Gaussian Step (Mean=0, Variance=1)')
plt.xlabel('Value')
plt.ylabel('Probability (x)')

plt.subplot(1, 3, 3)
plt.hist(discrete_non_uniform, color='orange', edgecolor='black
plt.title('Discrete Non-Uniform Distribution - Asymmetric')
plt.xlabel('Value')
plt.ylabel('Probability (x)')

plt.tight_layout()

plt.show()
```

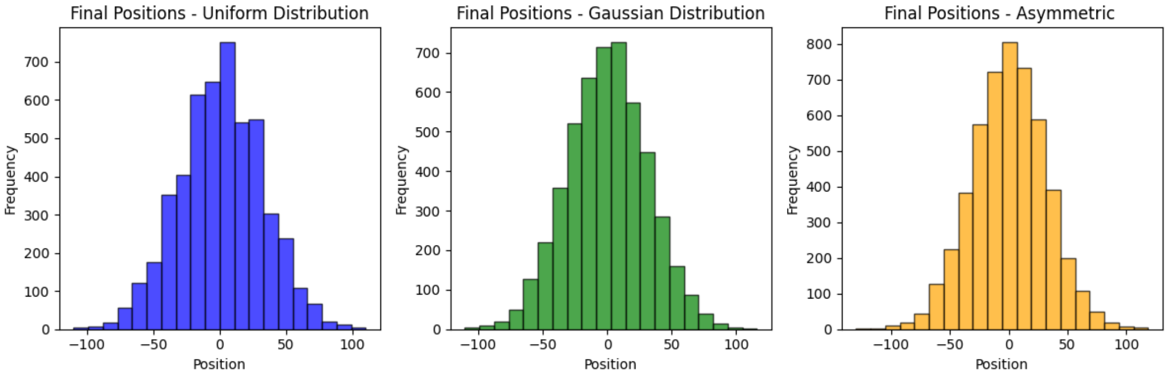


5.1.B

In [ ]:
```
steps = 5000
path_UD=np.zeros((steps,sample_size))
path_GD = np.zeros((steps,sample_size))
path_AS = np.zeros((steps,sample_size))



final_positions_UD = np.zeros(steps)
final_positions_GD = np.zeros(steps)
final_positions_AS = np.zeros(steps)
plt.figure(figsize=(12, 4))
for j in range(steps):
    path_UD[j] =np.random.choice([-1,1], size=sample_size)
    path_GD[j] = np.random.normal(0,1,size=sample_size)
    path_AS[j] = np.random.choice([-1,(1-np.sqrt(3))/2,
                                   (1+np.sqrt(3))/2], size=samp
    position_UD = 0
    position_list_UD =[]
    position_list_UD.append(position_UD)



    position_GD = 0
    position_list_GD =[]
    position_list_GD.append(position_GD)

    position_AS = 0
    position_list_AS =[]
    position_list_AS.append(position_GD)

    for i in range(sample_size):
        position_UD = position_UD + path_UD[j][i]
        position_list_UD.append(position_UD)

        position_GD= position_GD + path_GD[j][i]
        position_list_GD.append(position_GD)

        position_AS= position_AS + path_AS[j][i]
        position_list_AS.append(position_AS)
```

```
    final_positions_UD[j] = np.sum(path_UD[j])
    final_positions_GD[j] = np.sum(path_GD[j])
    final_positions_AS[j] = np.sum(path_AS[j])

    plt.subplot(1, 3, 1)
    plt.plot(position_list_UD, color ='blue', linewidth=0.5, al
    plt.ylabel('t(Steps)')
    plt.subplot(1, 3, 2)
    plt.plot(position_list_GD, color ='green', linewidth=0.5, a
    plt.subplot(1, 3, 3)
    plt.plot(position_list_AS, color ='orange', linewidth=0.5, a

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)
plt.hist(final_positions_UD, bins=20, color='blue',
         alpha=0.7, edgecolor='black')
plt.title('Final Positions - Uniform Distribution')
plt.xlabel('Position')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
plt.hist(final_positions_GD, bins=20, color='green',
         alpha=0.7, edgecolor='black')
plt.title('Final Positions - Gaussian Distribution')
plt.xlabel('Position')
plt.ylabel('Frequency')

plt.subplot(1, 3, 3)
plt.hist(final_positions_AS, bins=20, color='orange',
         alpha=0.7, edgecolor='black')
plt.title('Final Positions - Asymmetric ')
plt.xlabel('Position')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

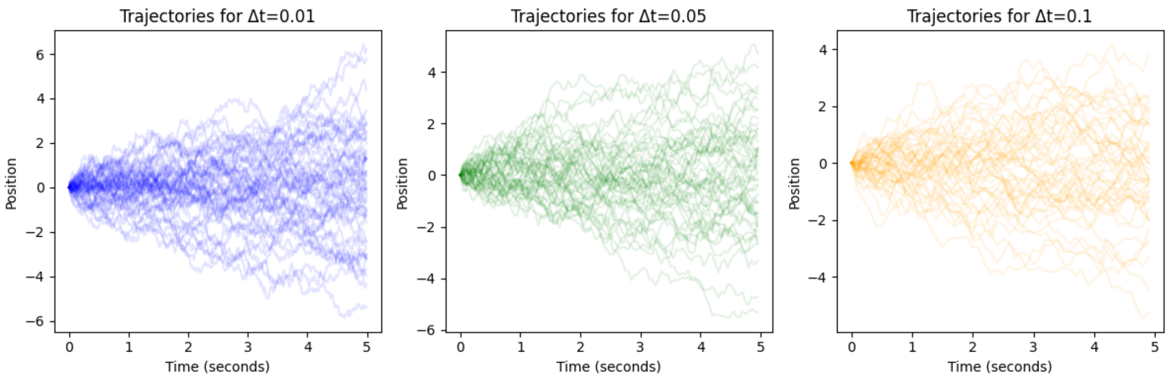




Final Positions - Uniform Distribution
Final Positions - Gaussian Distribution
Final Positions - Asymmetric
Frequency
Position

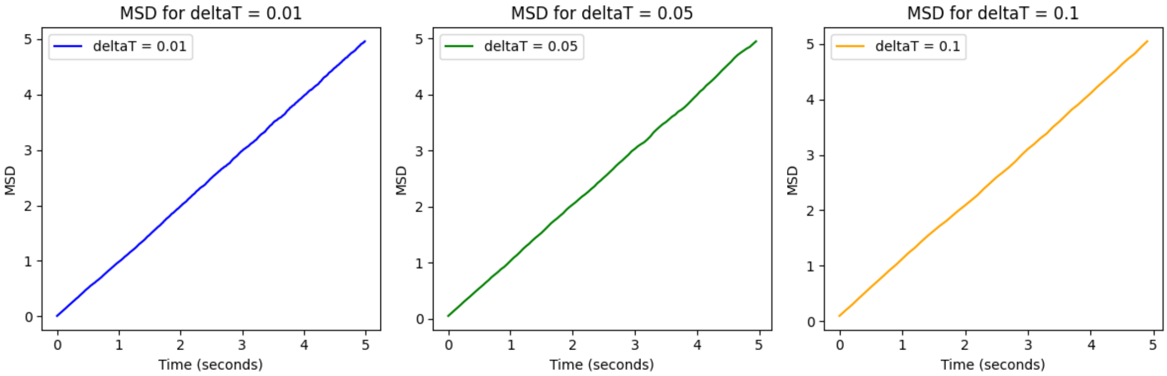
5.2.A

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


sample_size = 50
DeltaT_values = [0.01, 0.05, 0.1]
colors = ['blue', 'green', 'orange']


def simulate_diffusion(sample_size, DeltaT):
    steps = int(5 / DeltaT)
    trajectories = np.zeros((sample_size, steps))
    values =np.zeros((sample_size, steps))

    for j in range(sample_size):
        position = 0

        for i in range(1, steps):
            w= np.random.normal(0, 1)
            position = position + w*np.sqrt(DeltaT)
            trajectories[j, i] = position
            values[j,i]= w


    return trajectories

plt.figure(figsize=(12, 4))

for i, dt in enumerate(DeltaT_values):
    plt.subplot(1, 3, i + 1)
    trajectories = simulate_diffusion(sample_size, dt)


    for traj in trajectories:
        # print(traj.shape[0])
        plt.plot(np.arange(traj.shape[0]) * dt, traj, alpha=0.1

    plt.title(f'Trajectories for Δt={dt}')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Position')

plt.tight_layout()
plt.show()
```



In [ ]:

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

N = 10000
t = 5  # No of timesteps
dt_values = [0.01, 0.05, 0.1]
cases = ['deltaT = 0.01', 'deltaT = 0.05', 'deltaT = 0.1']
colors = ['blue', 'green', 'orange']

# Generate Trajectories and Calculate MSD
msd_values = []

for dt in dt_values:
    T = int(t / dt)
    r = np.sqrt(dt) * np.random.randn(T, N)
    r = np.sqrt(dt) * np.random.normal(0,1, size=(T, N))
    x = np.cumsum(r, axis=0)
    msd = np.mean(x**2, axis=1)
    msd_values.append(msd)

# Plotting MSD for each case in a single row
plt.figure(figsize=(12, 4))

for i, (msd, case) in enumerate(zip(msd_values, cases), 1):
    plt.subplot(1, len(dt_values), i)
    plt.plot(np.arange(0, t, dt_values[i-1])[:len(msd)], msd,
             label=case, color= colors[i-1])
    plt.title(f'MSD for {case}')
    plt.xlabel('Time (seconds)')
    plt.ylabel('MSD')
    plt.legend()

plt.tight_layout()
plt.show()
```


MSD for deltaT = 0.01
deltaT = 0.01
MSD
Time (seconds)
MSD for deltaT = 0.05
deltaT = 0.05
MSD
Time (seconds)
MSD for deltaT = 0.1
deltaT = 0.1
MSD
Time (seconds)

In [ ]:

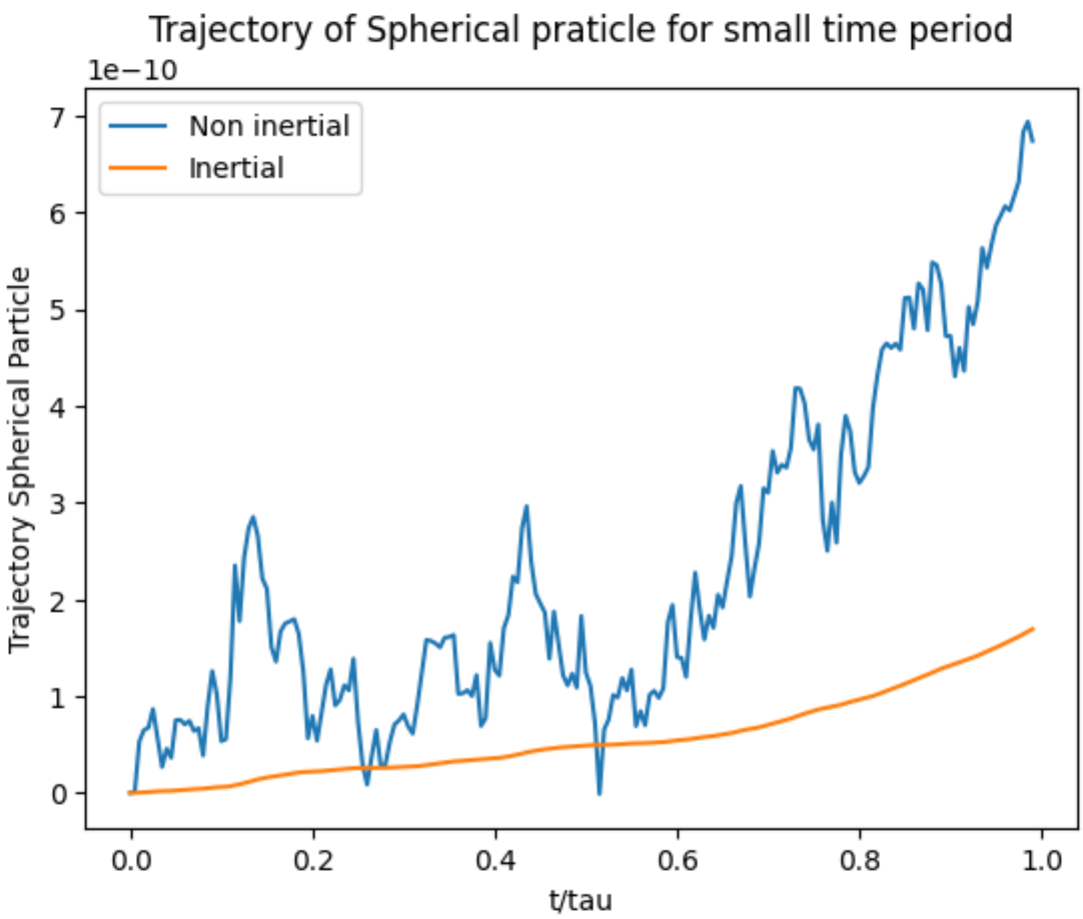
5.3.A

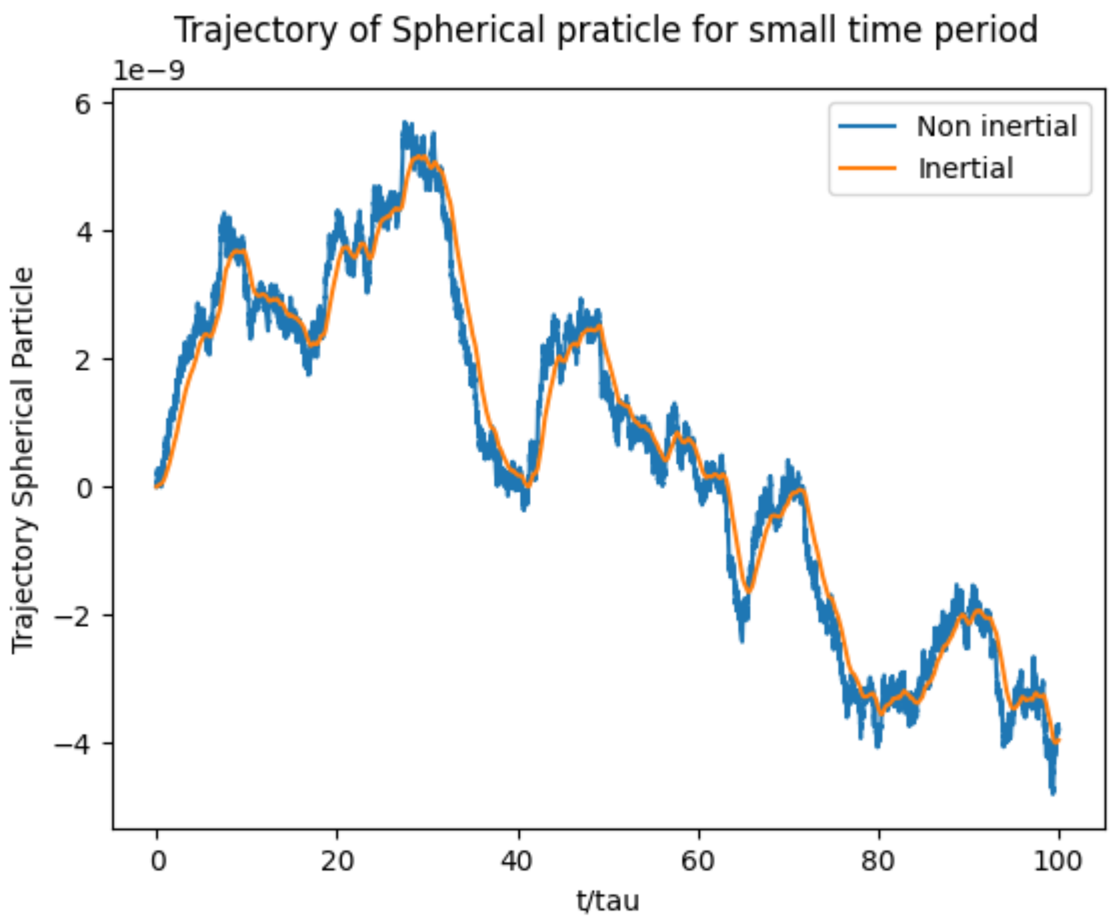
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

N = 1  # number of trajectory
radius = 1e-6
m = 1.11*10**(-14)
eta = 0.001
gamma = 6*np.pi*eta*radius
Temperature= 300
kb = 1.380649*10**(-23)
tau = m/gamma #0.588 # value from book

dt = 0.005*tau  #dt should be smaller than 0.1tau so divided by
length = 100*tau
T=int(length/dt) #time step
print(T,dt)


w = np.random.normal(0,1,size=(T,1))
w=w.flatten()
x=0
y=0
path_wo_mass=[]
path_mass=[]
path_wo_mass.append(x)
path_wo_mass.append(x)
path_mass.append(y)
path_mass.append(y)
temp_time=0
time_list=[]
time_list.append(temp_time)
time_list.append(temp_time)


def next_position_without_mass (previous_pos,wi):

    next_pos = previous_pos + np.sqrt(2*kb*
                                  Temperature*dt/gamma)*wi

    return next_pos

def next_position_with_mass (before_pre_position,
                             previous_pos, wi):

    denominator_1 = 1+ (dt*gamma/m)
    term_1 = (2+ (dt*gamma/m)) / denominator_1
    term_2 = 1/denominator_1
    term_3 =  np.sqrt(2*kb*Temperature*gamma)*dt**(3/2)*wi/(m*
```

```
                                                                    denc
    next_post = term_1*previous_pos - term_2 *before_pre_positic


    return next_post

for i in range(T):
    x = next_position_without_mass(path_wo_mass[-1], w[i])
    path_wo_mass.append(x)

    y = next_position_with_mass (path_mass[-2],path_mass[-1], w
    path_mass.append(y)

    temp_time = temp_time+ dt/tau
    time_list.append(temp_time)




# plt.figure(figsize = (20,6))

plt.subplot(1,1,1)
plt.plot(time_list[0:200], path_wo_mass[0:200], label= 'Non iner
plt.plot(time_list[0:200], path_mass[0:200], label= 'Inertial')
plt.xlabel('t/tau')
plt.ylabel('Trajectory Spherical Particle')
plt.title("Trajectory of Spherical praticle for small time peric
plt.legend()
plt.show()
plt.subplot(1,1,1)
plt.plot(time_list, path_wo_mass, label= 'Non inertial')
plt.plot(time_list, path_mass, label= 'Inertial')
plt.xlabel('t/tau')
plt.ylabel('Trajectory Spherical Particle')
plt.title("Trajectory of Spherical praticle for small time peric
plt.legend()
plt.show()

def compute_time_averaged_msd(path):
    timestep= len(path)
    time_averaged_msd = np.zeros(timestep)

    for t in range(timestep):
        squared_displacements = [(path[t + x] - path[x])**2
                                 for x in range(timestep - t)]
        time_averaged_msd[t] = np.mean(squared_displacements)
    # for t in range(1, timestep):
    #     squared_displacements = (path[t:] - path[:-t])**2
    #     time_averaged_msd[t] = np.mean(squared_displacements)
    return time_averaged_msd
```

```python
path_wo_mass= np.array(path_wo_mass)
path_mass = np.array(path_mass)

msd_wo_mass = compute_time_averaged_msd(path_wo_mass)
# print(msd_wo_mass)

msd_mass =  compute_time_averaged_msd(path_mass)

# plt.loglog(time_list, msd_wo_mass, label= 'Non inertial')
# plt.loglog(time_list, msd_mass, label= 'inertial')
# plt.legend()
# plt.show()

average_msd_wo_mass = np.mean(msd_wo_mass)
average_msd_mass = np.mean(msd_mass)

print('Ensemble averged MSD', average_msd_wo_mass)
print('Ensemble averged MSD', average_msd_wo_mass)
```

```
20000 2.9443664472000647e-09
```



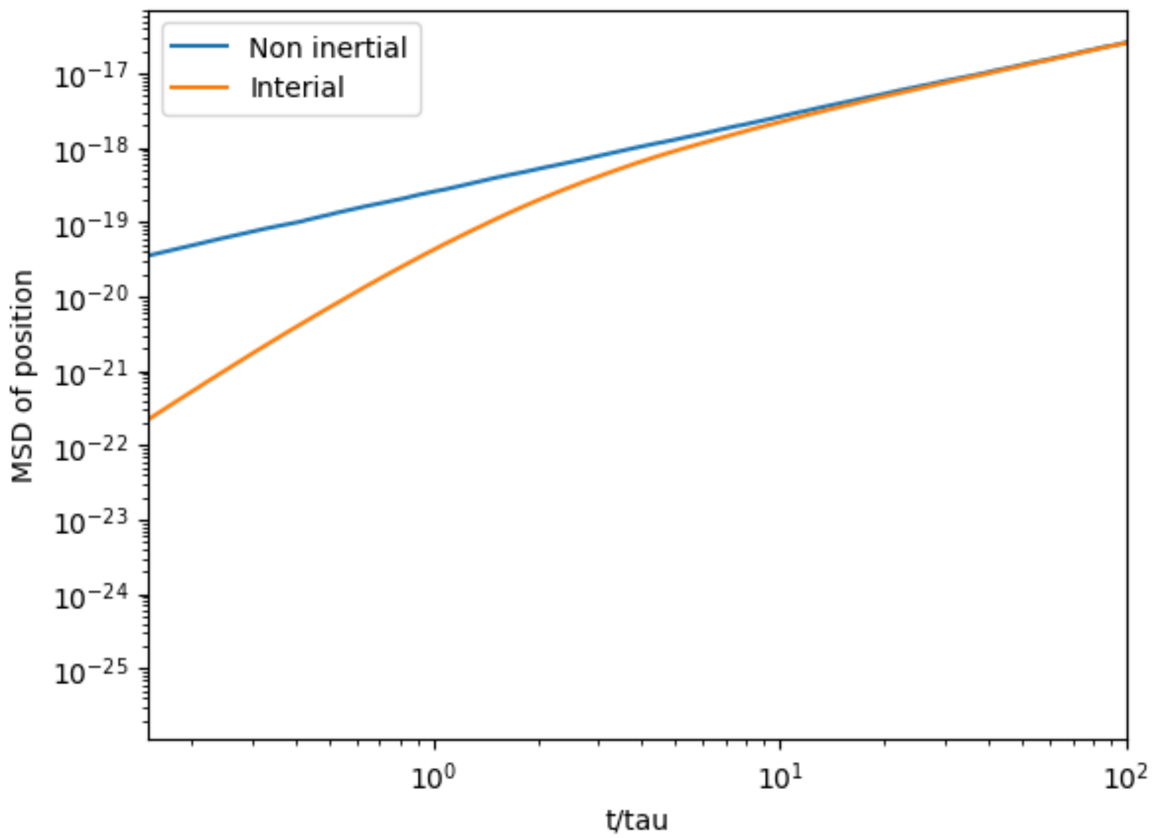



```
Ensemble averged MSD 1.987842982059189e-17
Ensemble averged MSD 1.987842982059189e-17
```

5.3A

In [ ]:
```
#b)
N=10000

mass_matrix =[]
wo_mass_matrix=[]

for k in range(N):
    w2 = np.random.normal(0,1,size=(T,1))
    w2=w2.flatten()
    path_mass2=[0,0]
    path_wo_mass2=[0,0]

    for i in range (T):
        x = next_position_without_mass(path_wo_mass2[-1], w2[i]
        path_wo_mass2.append(x)

        y = next_position_with_mass (path_mass2[-2],
                                     path_mass2[-1], w2[i])
        path_mass2.append(y)

    wo_mass_matrix.append(path_wo_mass2)
    mass_matrix.append(path_mass2)
```

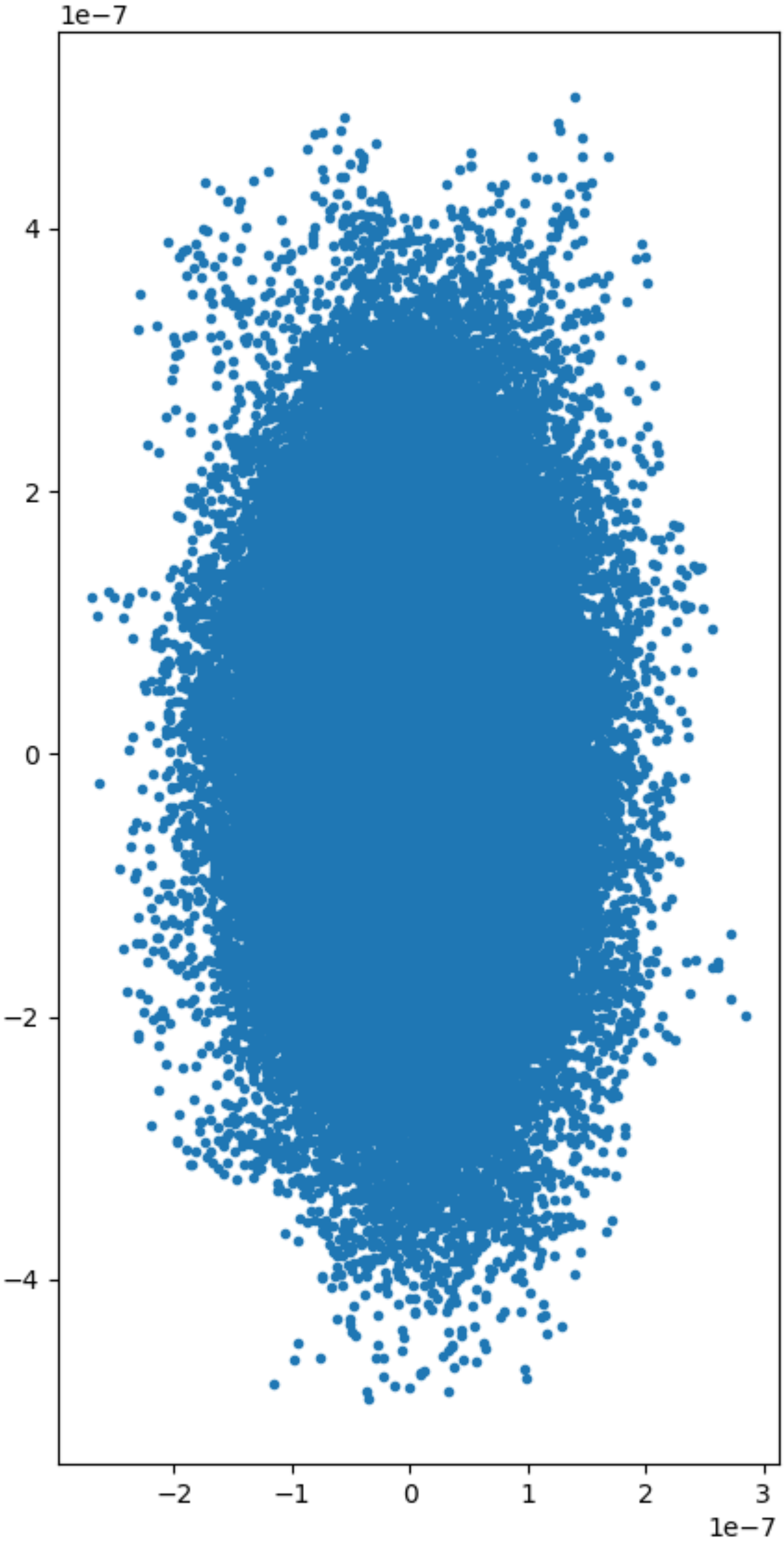
```python
wo_mass_matrix = (np.array(wo_mass_matrix))**2
mass_matrix = (np.array(mass_matrix))**2


msd_wo_mass =[0,0]
msd_mass = [0,0]
for i in range(T):
    mean_column1 = np.mean(wo_mass_matrix[:,i])
    msd_wo_mass.append(mean_column1)

    mean_column2 = np.mean(mass_matrix[:,i])
    msd_mass.append(mean_column2)

plt.loglog(time_list, msd_wo_mass, label='Non inertial')
plt.loglog(time_list, msd_mass, label="Interial")
plt.legend()
plt.xlim(0.15,100)
plt.xlabel('t/tau')
plt.ylabel('MSD of position')
plt.show()
time_average_wo_mass = np.mean(msd_wo_mass)
time_average_mass = np.mean(msd_mass)

print('Time Averaged MSD non inertial', time_average_wo_mass )
print('Time averaged MSD inertial', time_average_mass )
```




```
Time Averaged MSD non inertial 1.303159007900478e-17
Time averaged MSD inertial 1.2646616585881241e-17
```

5.4.a

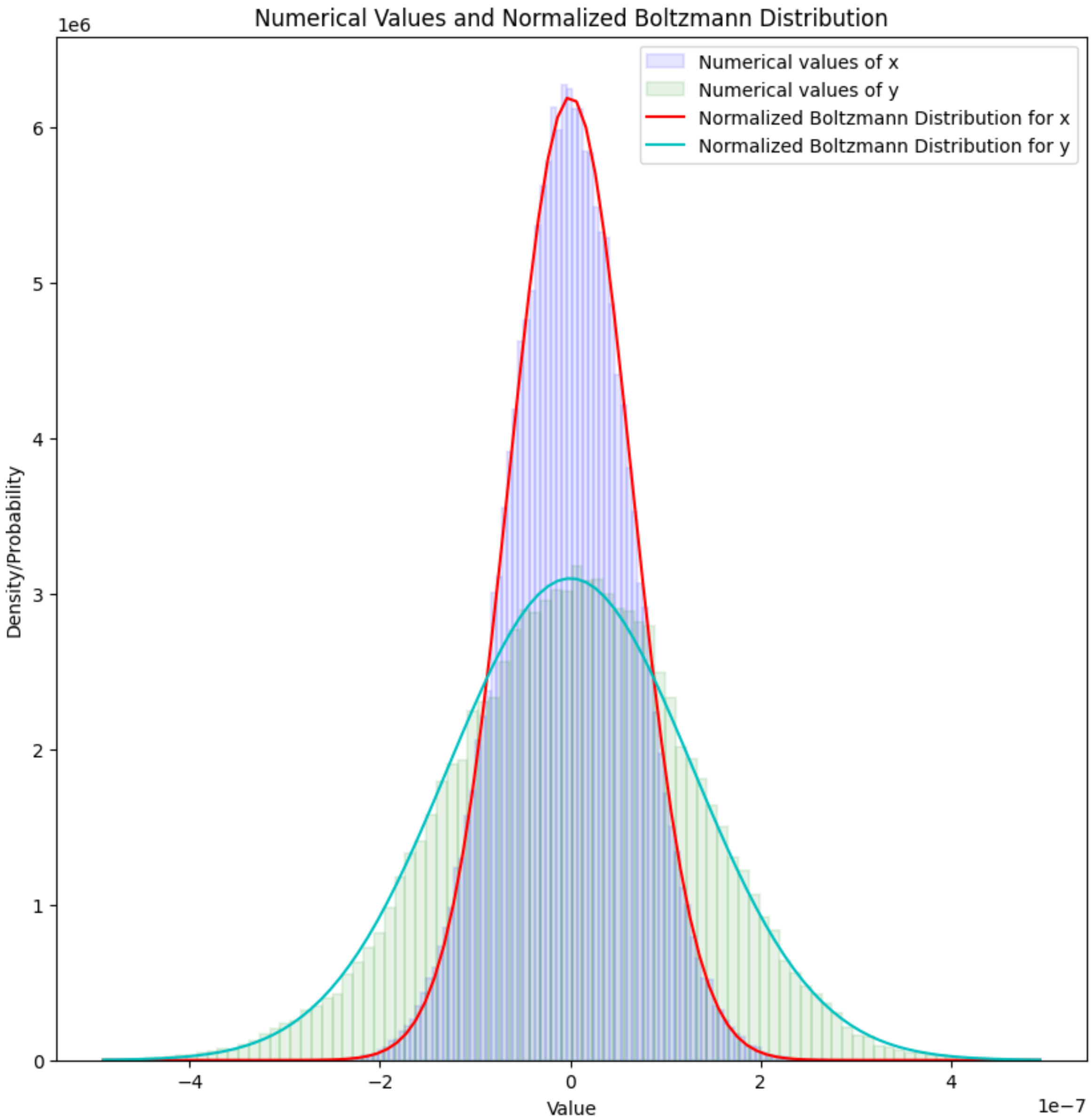
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt



radius =10**(-6)
eta= 0.001
gamma = 6*np.pi*eta*radius
temperature = 300
kx= 1*10**(-6)
ky = 0.25*10**(-6)
kb = 1.380649*10**(-23)

dt= 0.001

t=100000
position = np.zeros((t,2))
wx = np.random.normal(0,1,size=(t,1))
wy = np.random.normal(0,1,size=(t,1))
for i in range (t-1):
    position[i+1][0] = position[i][0] - kx*position[i][0]*dt/gar
    position[i+1][1] = position[i][1] - ky*position[i][1]*dt/gar

plt.figure(figsize =(5,10))
plt.plot(position[:,0], position[:,1],'.')
plt.show()
```



5.4.B

In [ ]:
```
#probability distribution
x = position[:,0]
y = position[:,1]

def potential (x,k):
    return 0.5*k*x**2
```

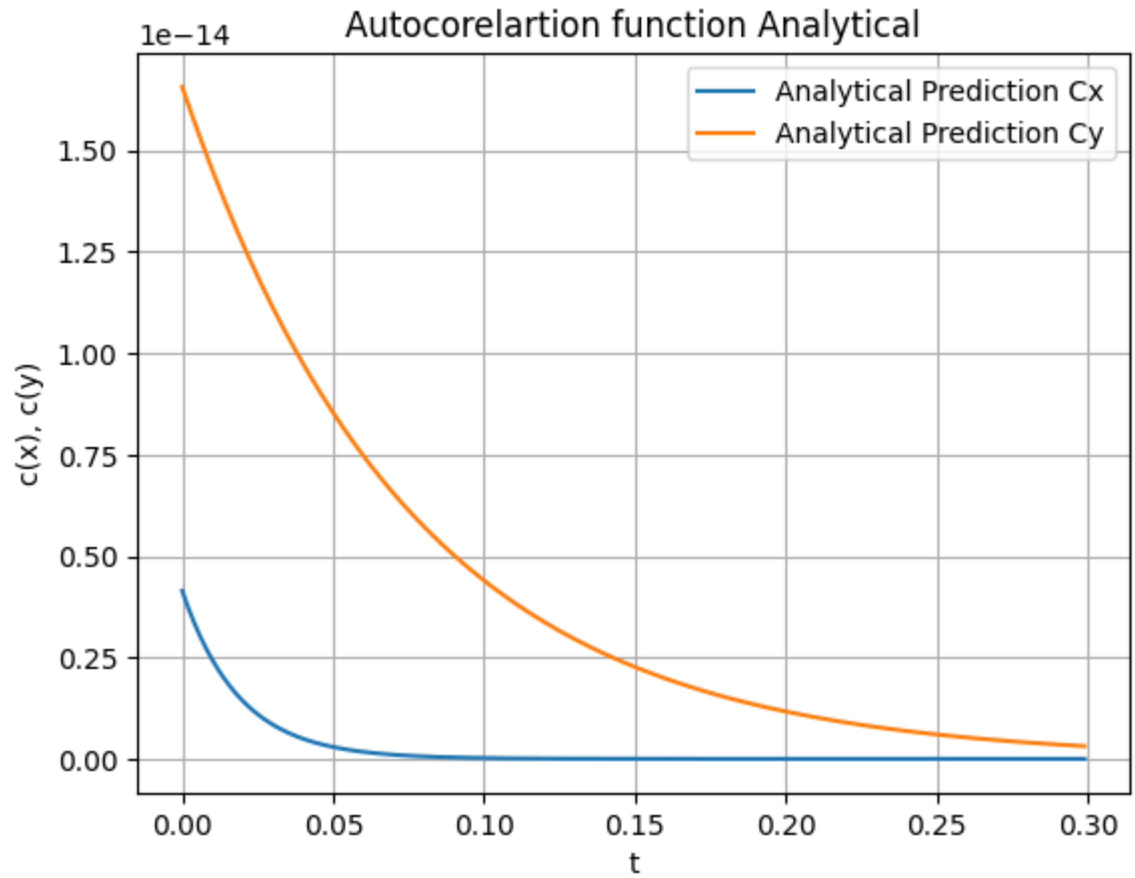
```
def boltzmann_distribution_x(x,k):
    scaled_potential = potential(x, k) - np.min(potential(x, k)
    return np.exp(-scaled_potential / (kb * temperature))

x_values = np.linspace(min(min(x), min(y)), max(max(x), max(y))
# print(x_values)

probabilities_x = boltzmann_distribution_x(x_values,kx)
probabilities_y = boltzmann_distribution_x(x_values,ky)

probabilities_x /= np.trapz(probabilities_x, x=x_values)
probabilities_y /= np.trapz(probabilities_y, x=x_values)

# probabilities_x /= np.sum(probabilities_x)
# probabilities_y /= np.sum(probabilities_y)


plt.figure(figsize =(10,10))

plt.hist(x, bins=90, density=True, color="b", alpha=0.1, label='
         edgecolor= 'blue', linewidth=1.2)
plt.hist(y, bins=90, density=True, color="g", alpha=0.1, label='
         edgecolor = 'green', linewidth=1.2)
plt.plot(x_values, probabilities_x, color="r", label="Normalized
plt.plot(x_values, probabilities_y, color="c", label="Normalized

plt.title('Numerical Values and Normalized Boltzmann Distributio
plt.xlabel('Value')
plt.ylabel('Density/Probability')
plt.legend()
plt.show()
```

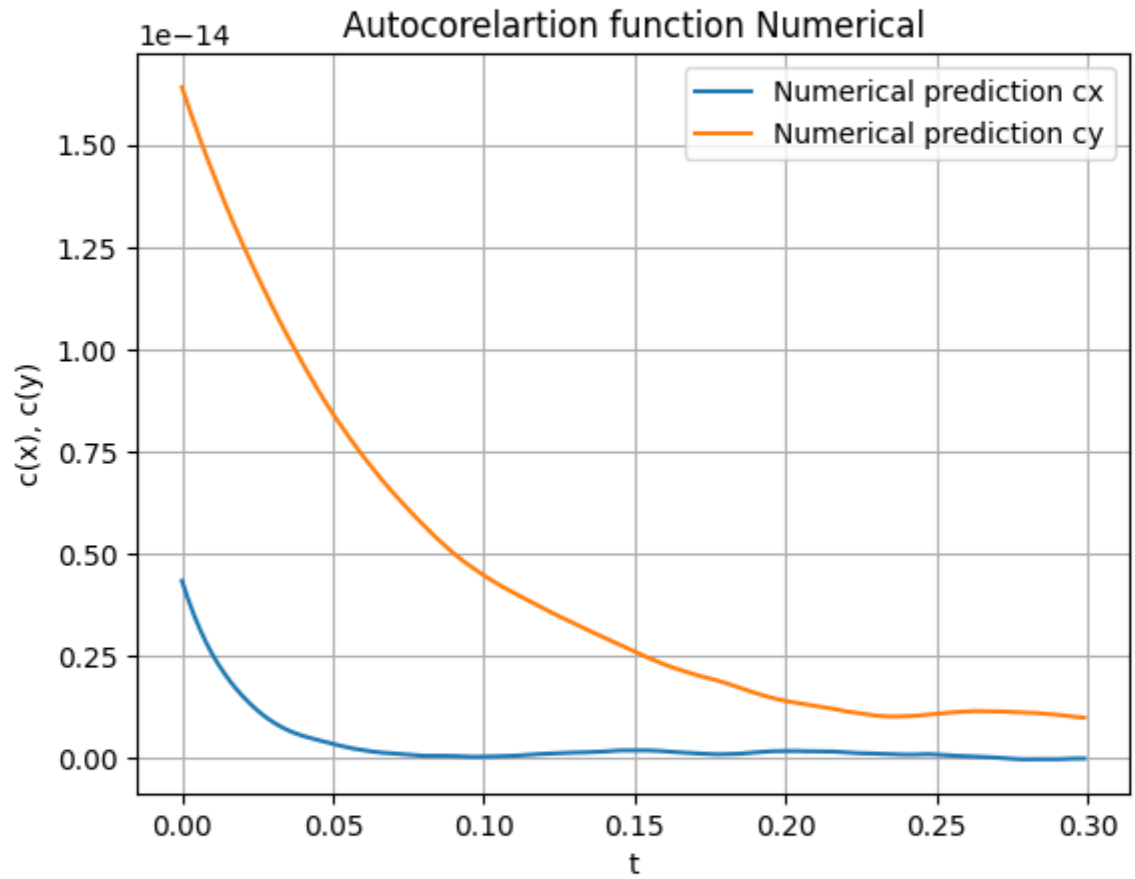


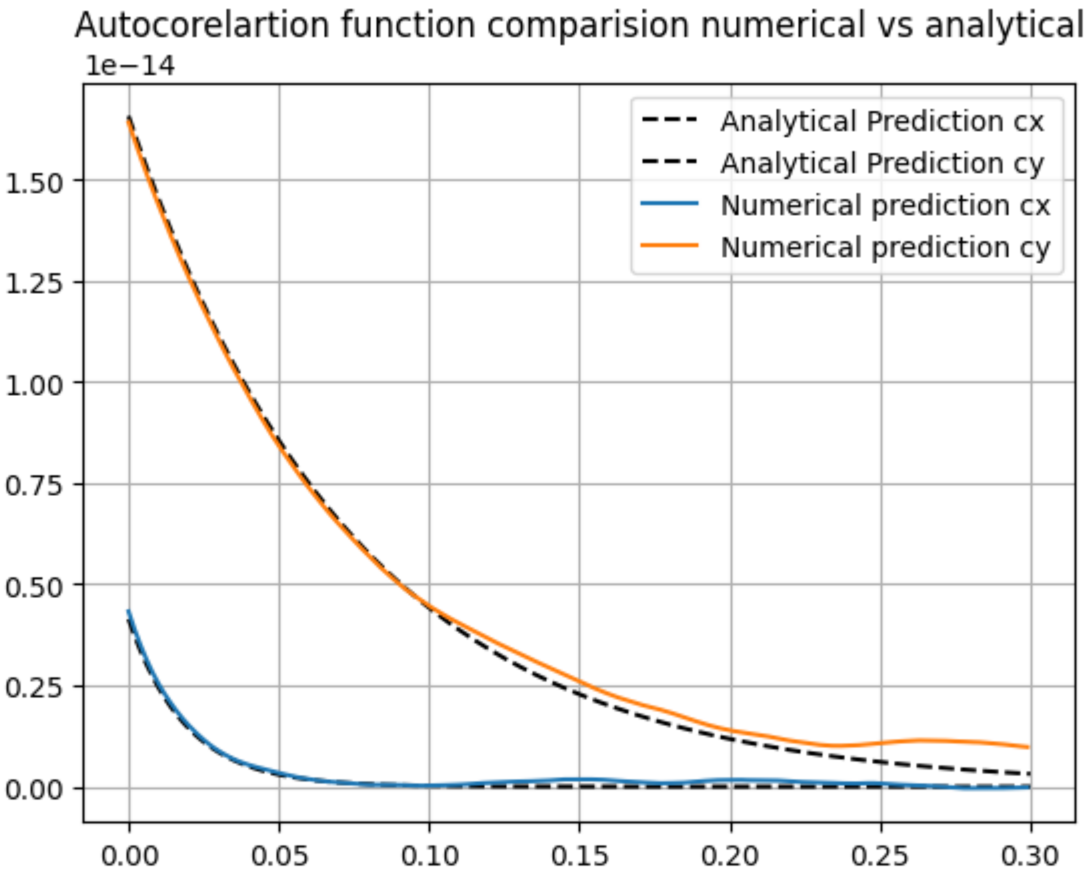
5.4.C

```
In [ ]: #positional autocorrelation function
position= np.array(position)
# print(traj)
cx=[]
cy=[]
time_list=[]
ttemp=0
for i in range(300):
    cxtemp=0
    cytemp=0
    for j in range(len(position)-i):
        cxtemp+= (position[i+j][0])*position[j][0]
        cytemp += (position[i+j][1])*position[j][1]

    time_list.append(dt*i)
    cx.append(cxtemp / (len(position) - i))
    cy.append(cytemp / (len(position) - i))
```

```
time_list=np.array(time_list)
t_values = np.linspace(0, 0.3, 300)

cx_values = kb * temperature / kx * np.exp(-kx * time_list / gar
cy_values = kb * temperature / ky * np.exp(-ky * time_list / gar



plt.plot(time_list, cx_values, label='Analytical Prediction Cx']
plt.plot(time_list, cy_values, label='Analytical Prediction Cy']
plt.xlabel('t')
plt.ylabel('c(x), c(y)')
plt.title('Autocorelartion function Analytical')
plt.grid(True)
plt.legend()
plt.show()



plt.plot(time_list, cx, label= "Numerical prediction cx")
plt.plot(time_list, cy, label= "Numerical prediction cy")
plt.xlabel('t')
plt.ylabel('c(x), c(y)')
plt.title('Autocorelartion function Numerical')
plt.grid(True)
plt.legend()
plt.show()



plt.plot(t_values, cx_values, label='Analytical Prediction cx',
          linestyle ='--' , color='black')
plt.plot(t_values, cy_values, label='Analytical Prediction cy',
          linestyle ='--',color = 'black')
plt.plot(time_list, cx, label= "Numerical prediction cx")
plt.plot(time_list, cy, label= "Numerical prediction cy")
plt.title('Autocorelartion function comparision numerical vs ana
plt.grid(True)
plt.legend()
plt.show()
```


Autocorelartion function Analytical
1e−14
Analytical Prediction Cx
Analytical Prediction Cy
c(x), c(y)
t
0.00
0.25
0.50
0.75
1.00
1.25
1.50
0.00
0.05
0.10
0.15
0.20
0.25
0.30


Autocorelartion function Numerical
1e−14
Numerical prediction cx
Numerical prediction cy
c(x), c(y)
t
0.00
0.25
0.50
0.75
1.00
1.25
1.50
0.00
0.05
0.10
0.15
0.20
0.25
0.30



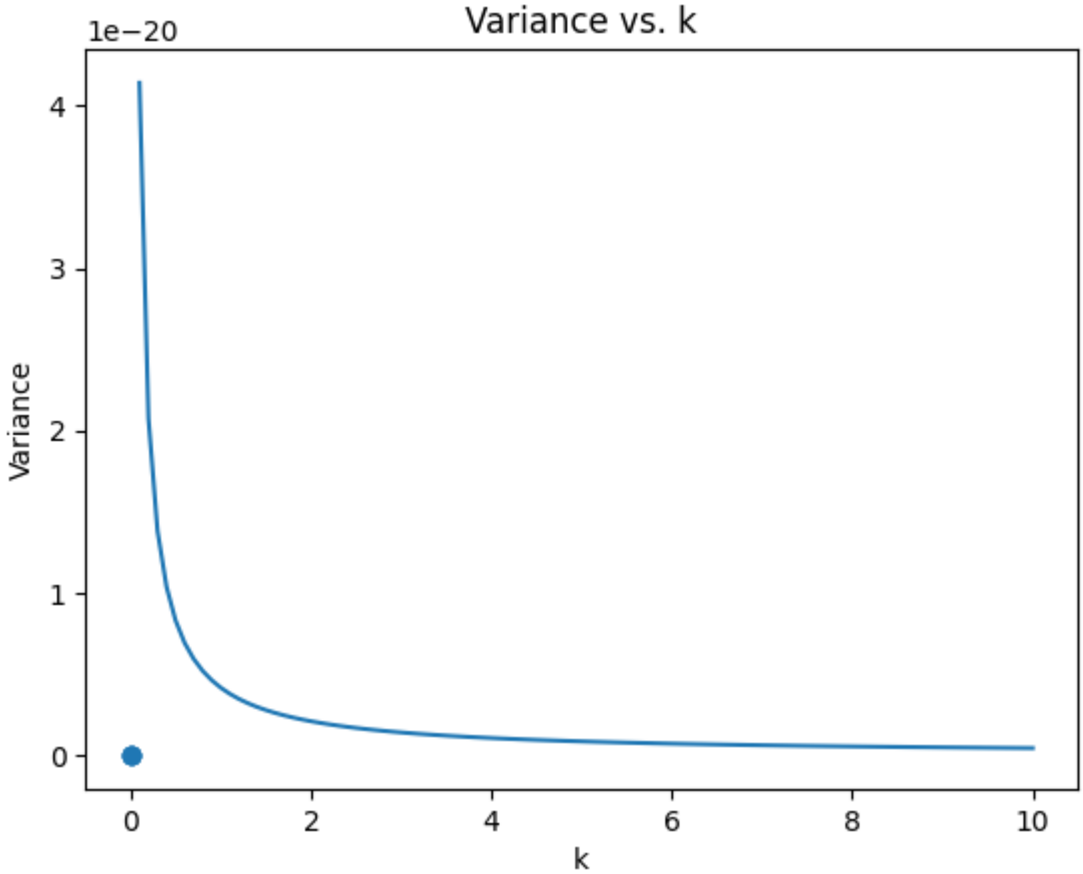
5.4.d

In [ ]:

```
positional_variance = []
p=[]
k_list =  [1.25, 2.5, 5, 10] #[10,5,2.5,1.25]
print(t)
import statistics

wx = np.random.normal(0,1,size=(t,1))
for j in k_list:
    position1=np.zeros((t,1))

    k= j*10**-6
    for i in range(t-1):
        position1[i+1] = position1[i] - k*position1[i]*dt/gamma
        + np.sqrt(2*kb*temperature*dt/gamma)*wx[i]
        position1[i+1]= position1[i+1]**2
    position1 = position1.flatten().tolist()
    positional_variance.append(statistics.pvariance(position1))

k_values = np.linspace(0.1, 10, 100)
variance_values = kb * temperature / k_values
k_list = np.array(k_list)*10**-6
plt.plot(k_values, variance_values)
plt.xlabel('k')
plt.ylabel('Variance')
plt.title('Variance vs. k')
plt.scatter(k_list, positional_variance)
```

```
plt.show()
```

5000



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

t = 5000
temperature = 300
gamma = 1.0
kb = 1.380649e-23
dt = 0.01

positional_variance = []
k_list = [1.25, 2.5, 5, 10]

for j in k_list:
    k = j * 1e-6
    wx = np.random.normal(0, 1, size=(t, 1))
    position1 = np.zeros((t, 1))

    for i in range(t - 1):
        position1[i + 1] = (
            position1[i]
            - k * position1[i] * dt / gamma
            + np.sqrt(2 * kb * temperature * dt / gamma) * wx[i]
        )

    positional_variance.append(np.var(position1.flatten()))
```

```
k_values = np.linspace(0.1, 10, 100)
variance_values_theoretical = kb * temperature / k_values

plt.plot(k_values, variance_values_theoretical, label='Theoreti
plt.scatter(k_list, positional_variance, label='Numerical Varia
plt.xlabel('k')
plt.ylabel('Variance')
plt.title('Variance vs. k')
plt.legend()
plt.show()
```