Homework 3, Living Crystals

Erik Norlin, 19970807-9299

November 27, 2022

9.1: a, b (v change)

500 iterations for every run in 9.1.

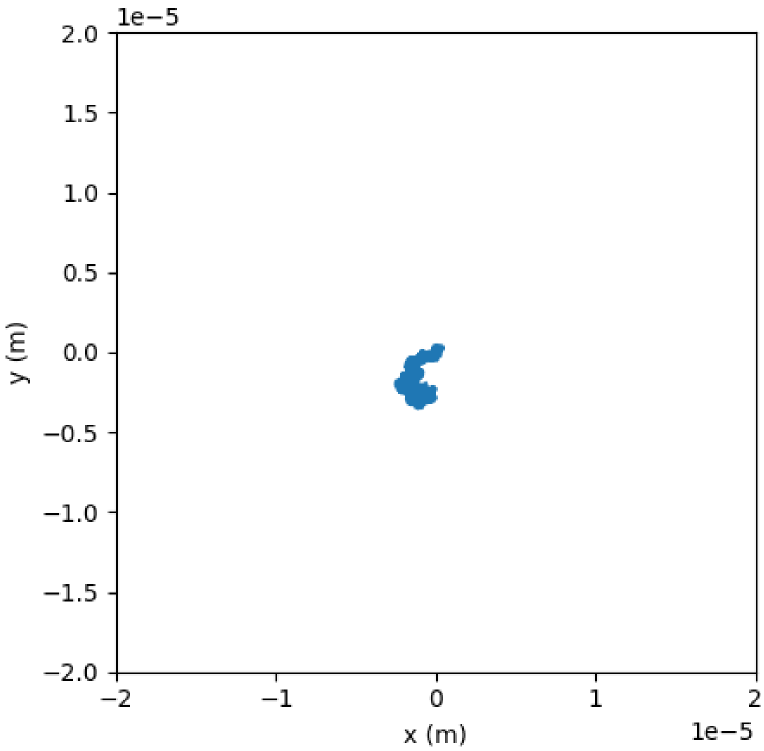

Figure 1: $v = 0$, $D_R = 5$ and $D_T = 8e - 14$.

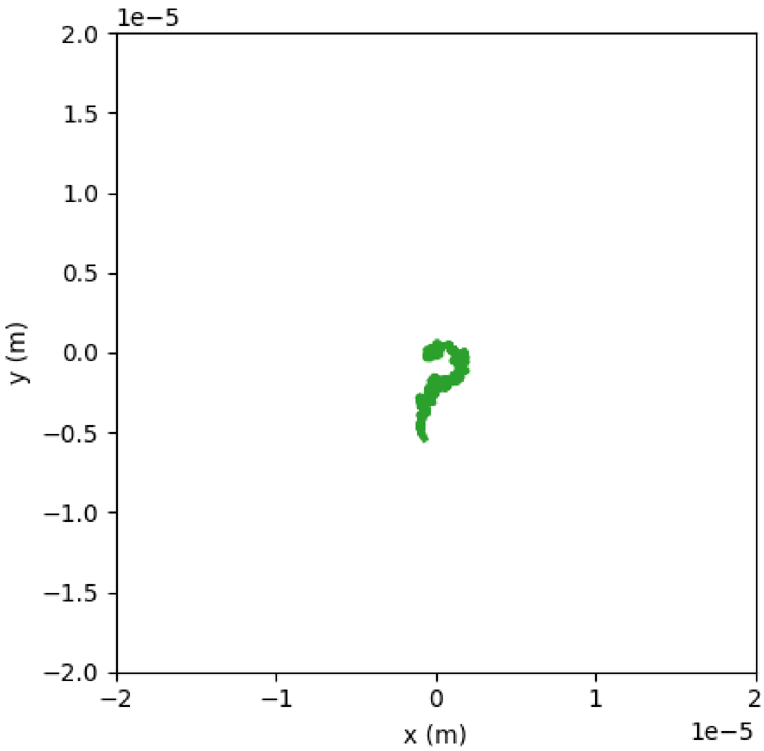

Figure 2: $v = 1e - 6$, $D_R = 5$ and $D_T = 8e - 14$.

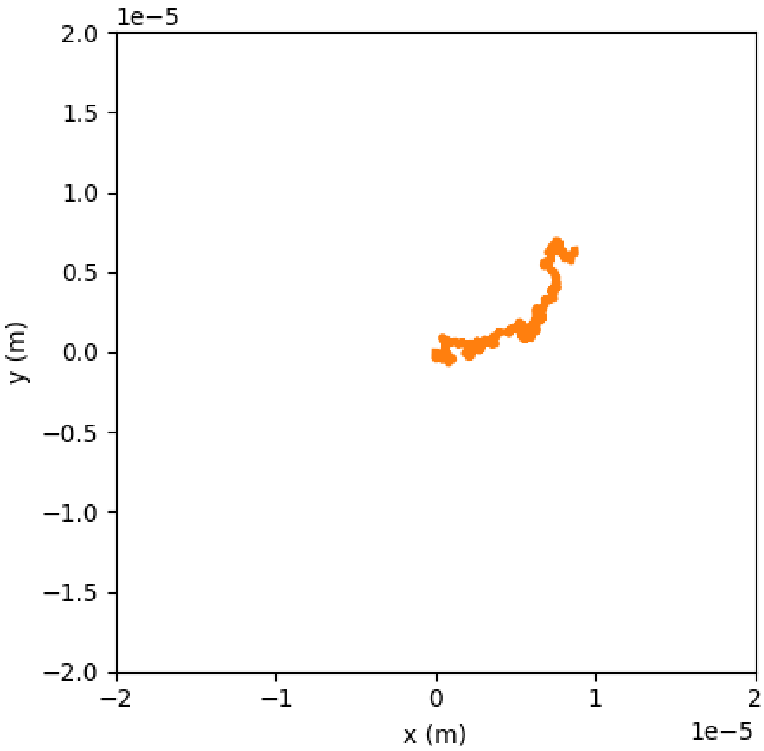

Figure 3: $v = 2e - 6$, $D_R = 5$ and $D_T = 8e - 14$.

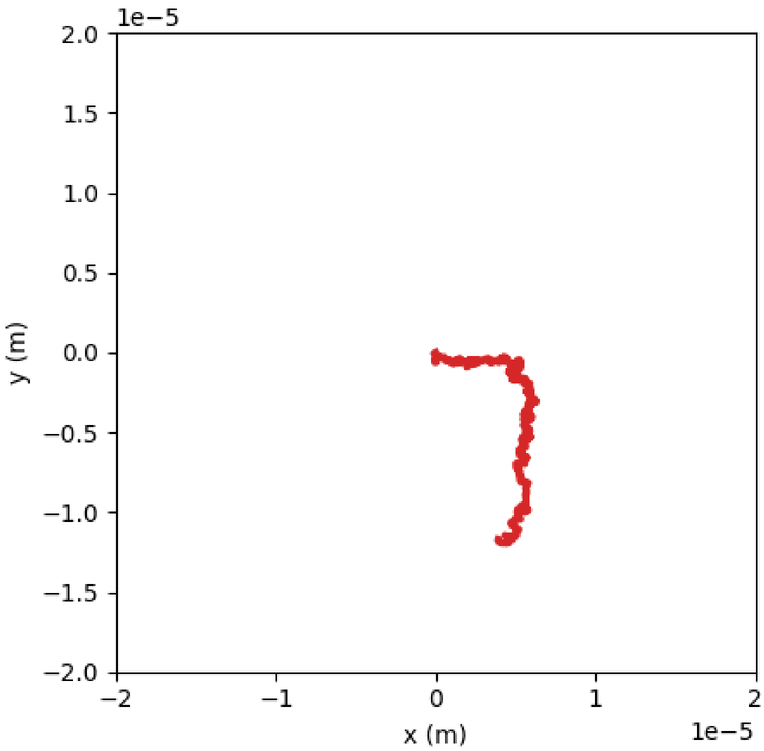

Figure 4: $v = 3e - 6$, $D_R = 5$ and $D_T = 8e - 14$.

9.1: c (D_T change)

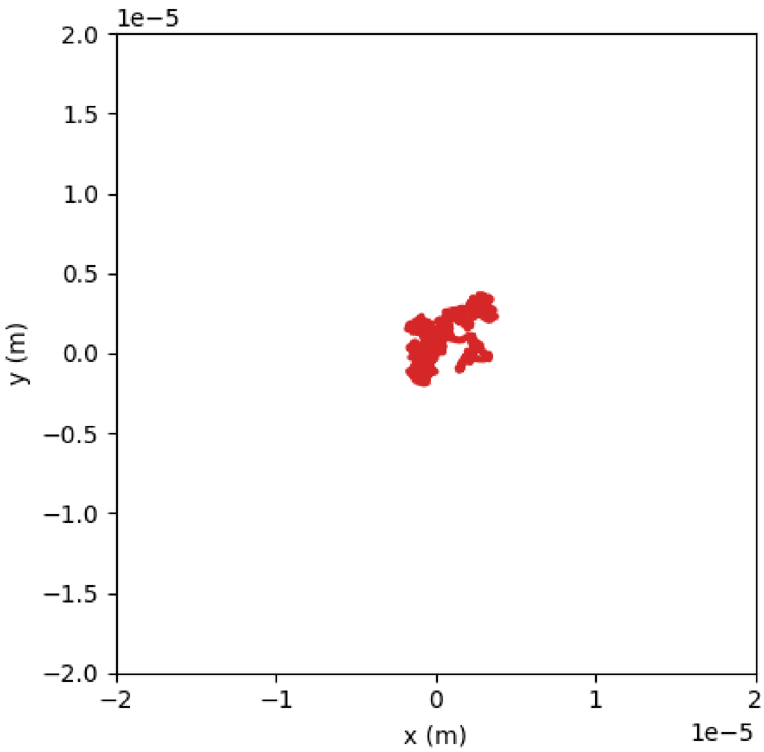

Figure 5: $v = 3e - 6$, $D_R = 0.5$ and $D_T = 2e - 12$.

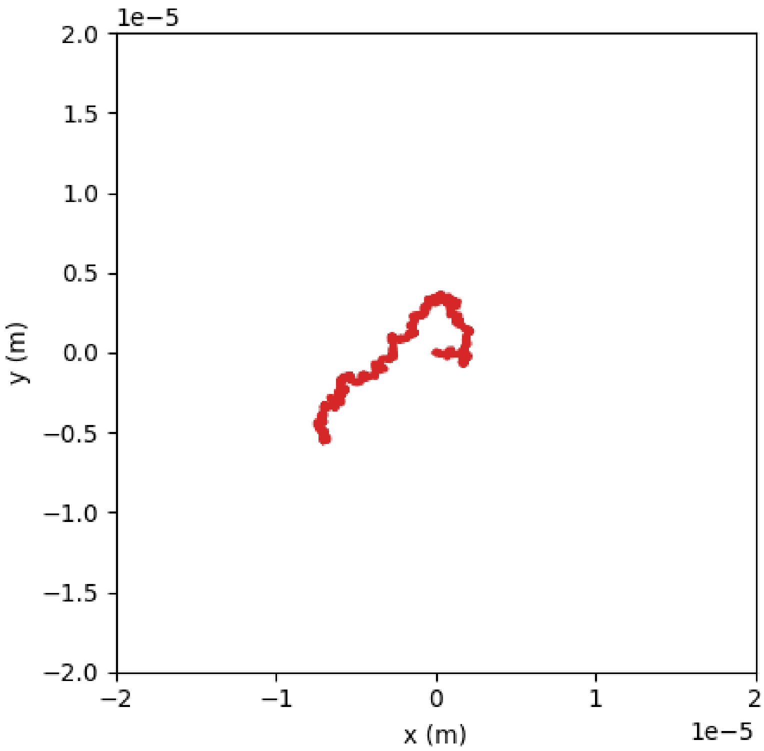

Figure 6: $v = 3e-6$, $D_R = 0.5$ and $D_T = 5e-13$.

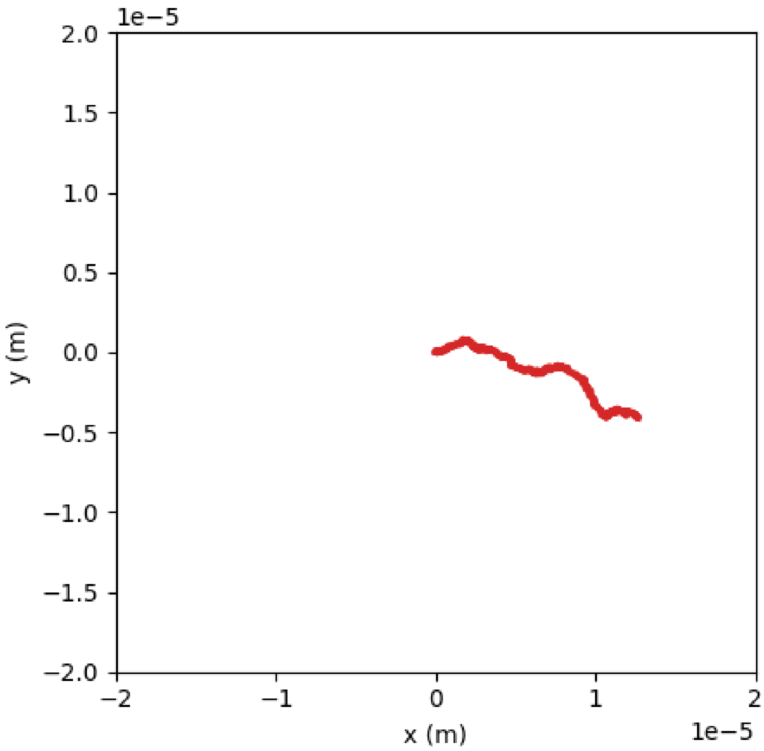

Figure 7: $v = 3e-6$, $D_R = 0.5$ and $D_T = 8e-14$.

9.1: d (D_R change)

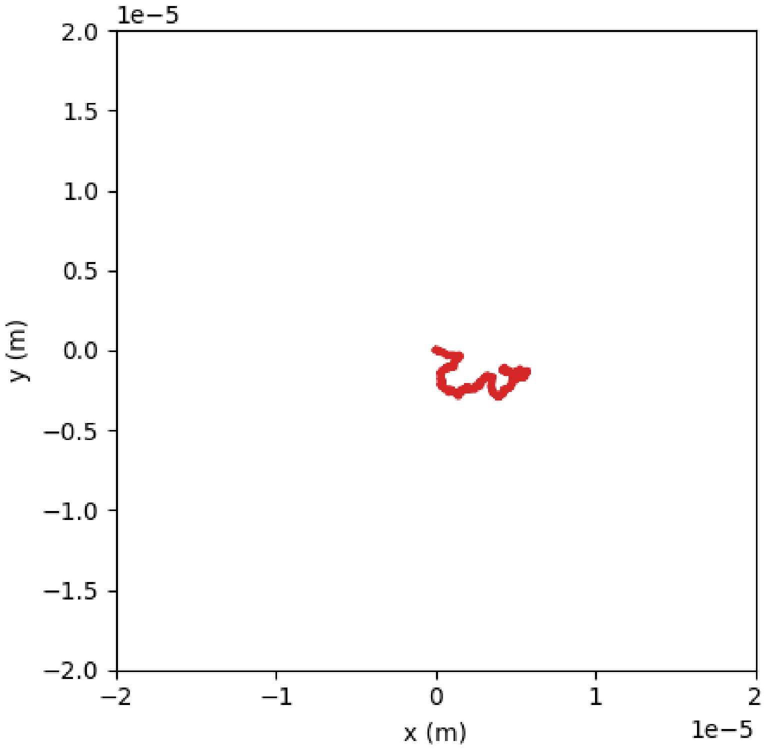

Figure 8: $v = 3e - 6$, $D_R = 5$ and $D_T = 8e - 14$.

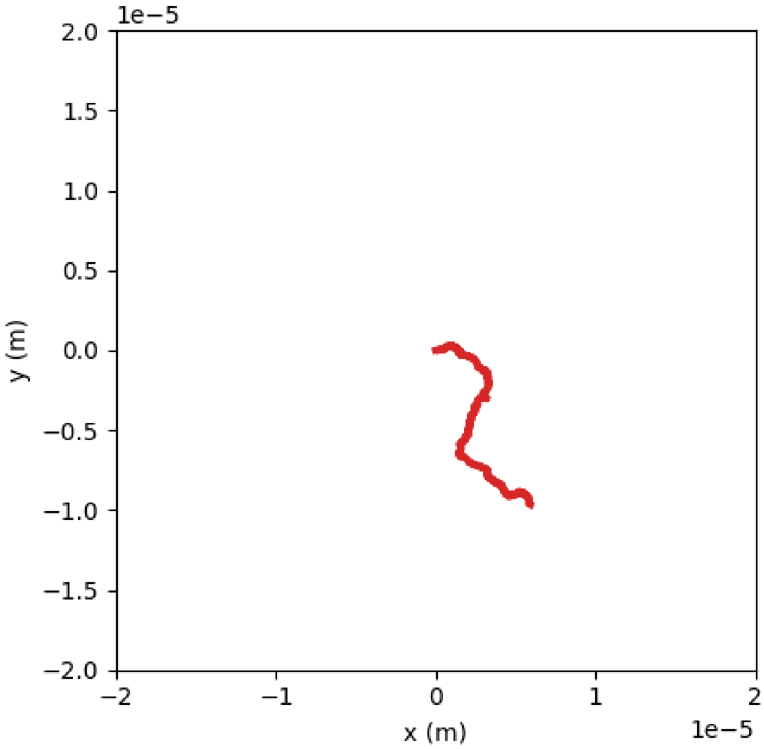

Figure 9: $v = 3e - 6$, $D_R = 0.5$ and $D_T = 8e - 14$.

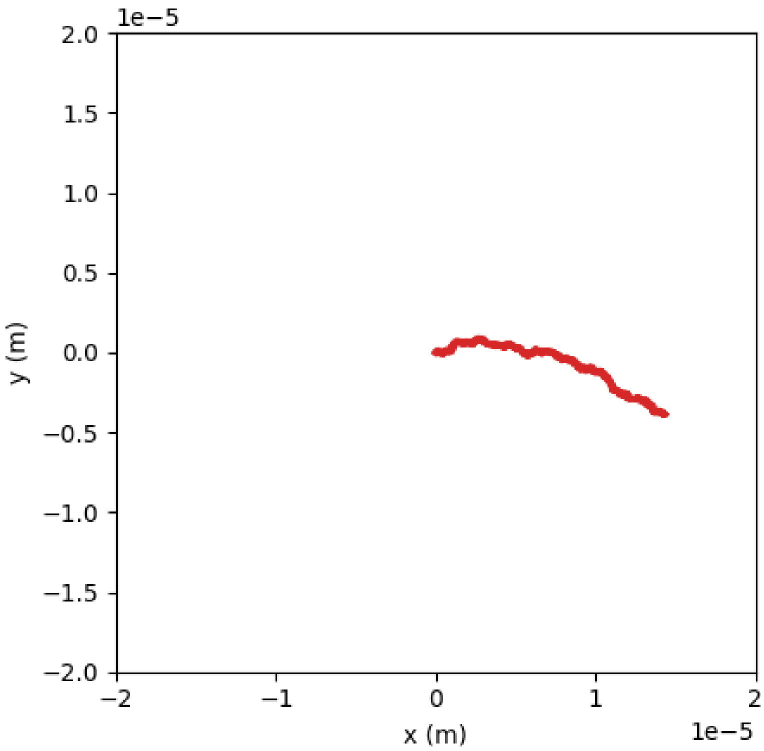

Figure 10: $v = 3e - 6$, $D_R = 0.05$ and $D_T = 8e - 14$.

9.2: a, b, c

10000 iterations for all runs in 9.2.

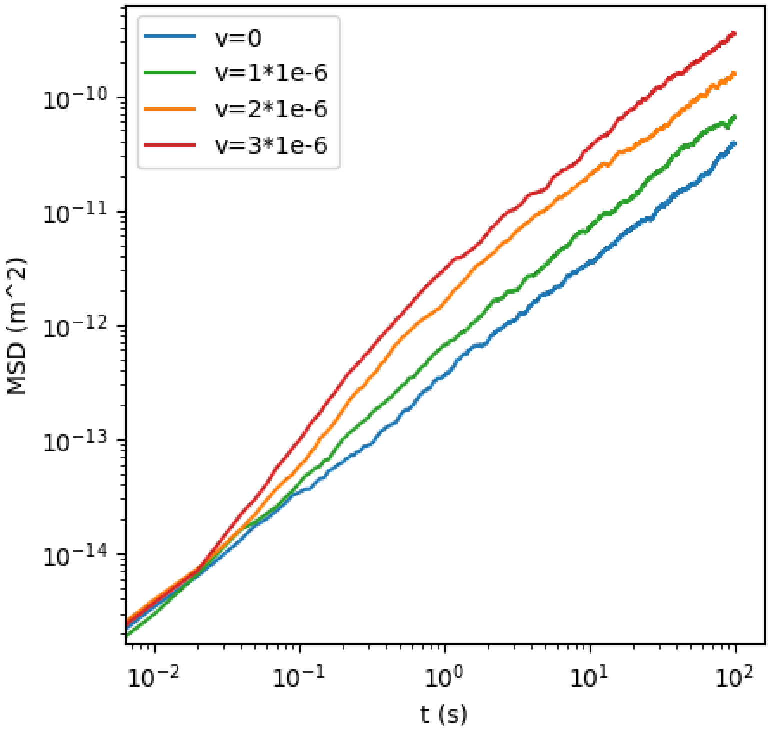

Figure 11: Ensemble averaged MSD. $D_R = 5$ and $D_T = 8e - 14$.

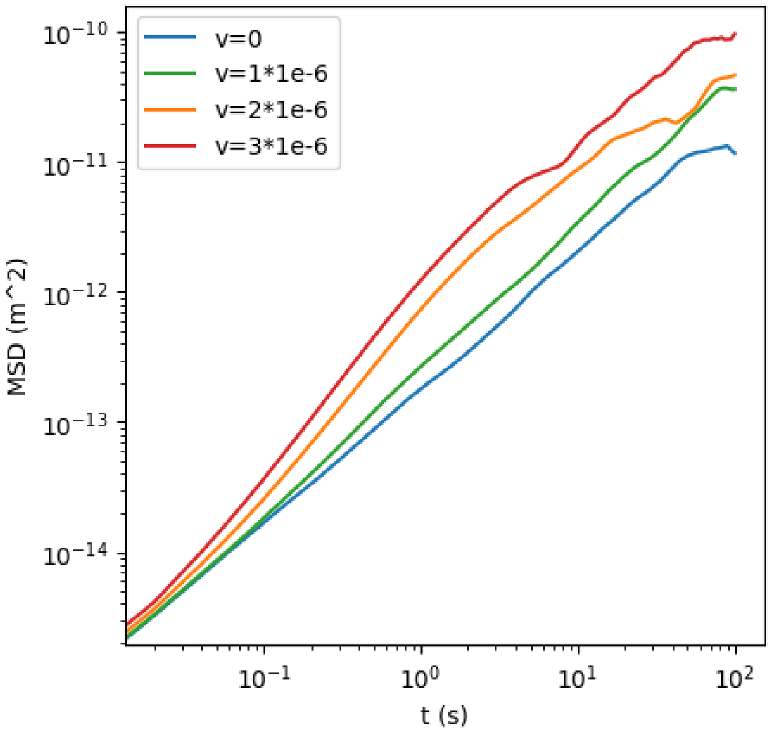

Figure 12: Time averaged MSD. $D_R = 5$ and $D_T = 8e - 14$.

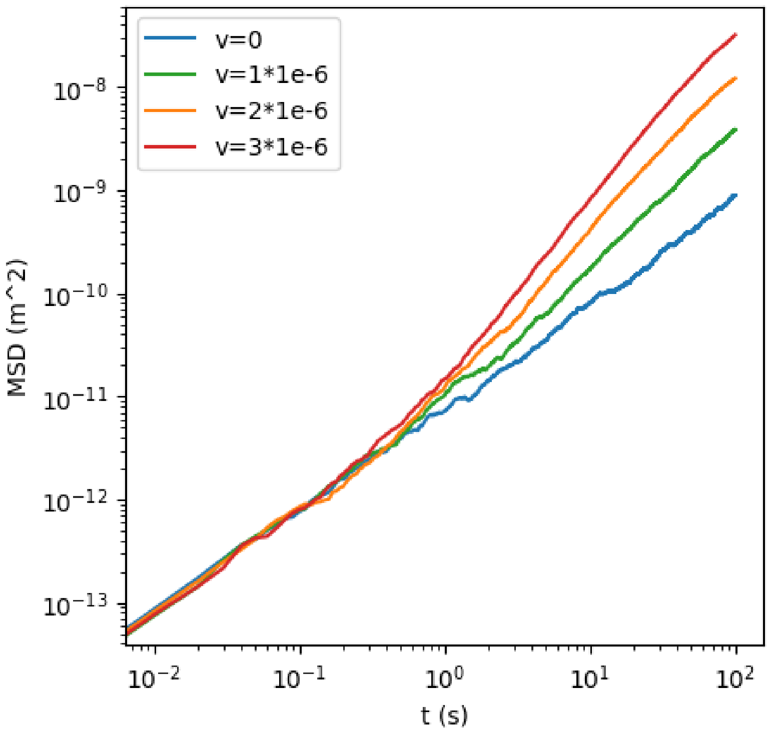

Figure 13: Ensemble averaged MSD. $D_R = 0.05$ and $D_T = 2e - 12$.

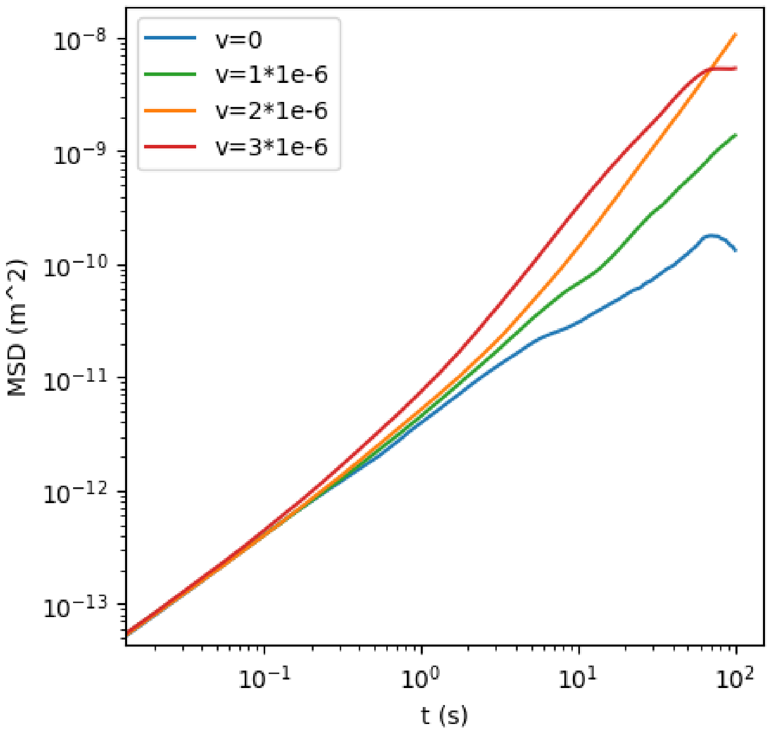

Figure 14: Time averaged MSD. $D_R = 0.05$ and $D_T = 2e - 12$.

9.3: a, b, c, d

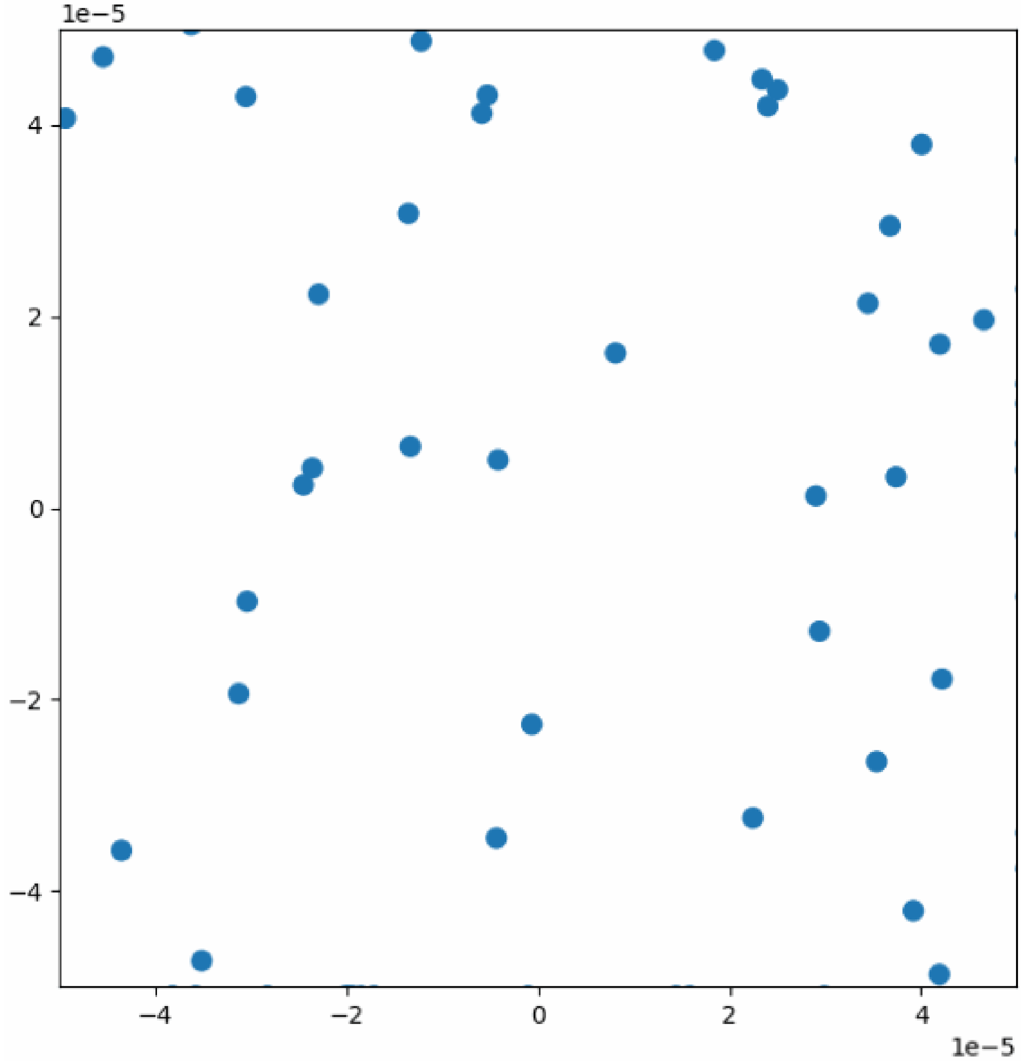

Figure 15: Both x and y-axis are length (m). No. time steps = 1000, no. particles = 100, $\Delta t = 0.01$, $v = 3e-6$, $R = 1e-6$, $D_R = 1$, $D_T = 0.1e-6$.

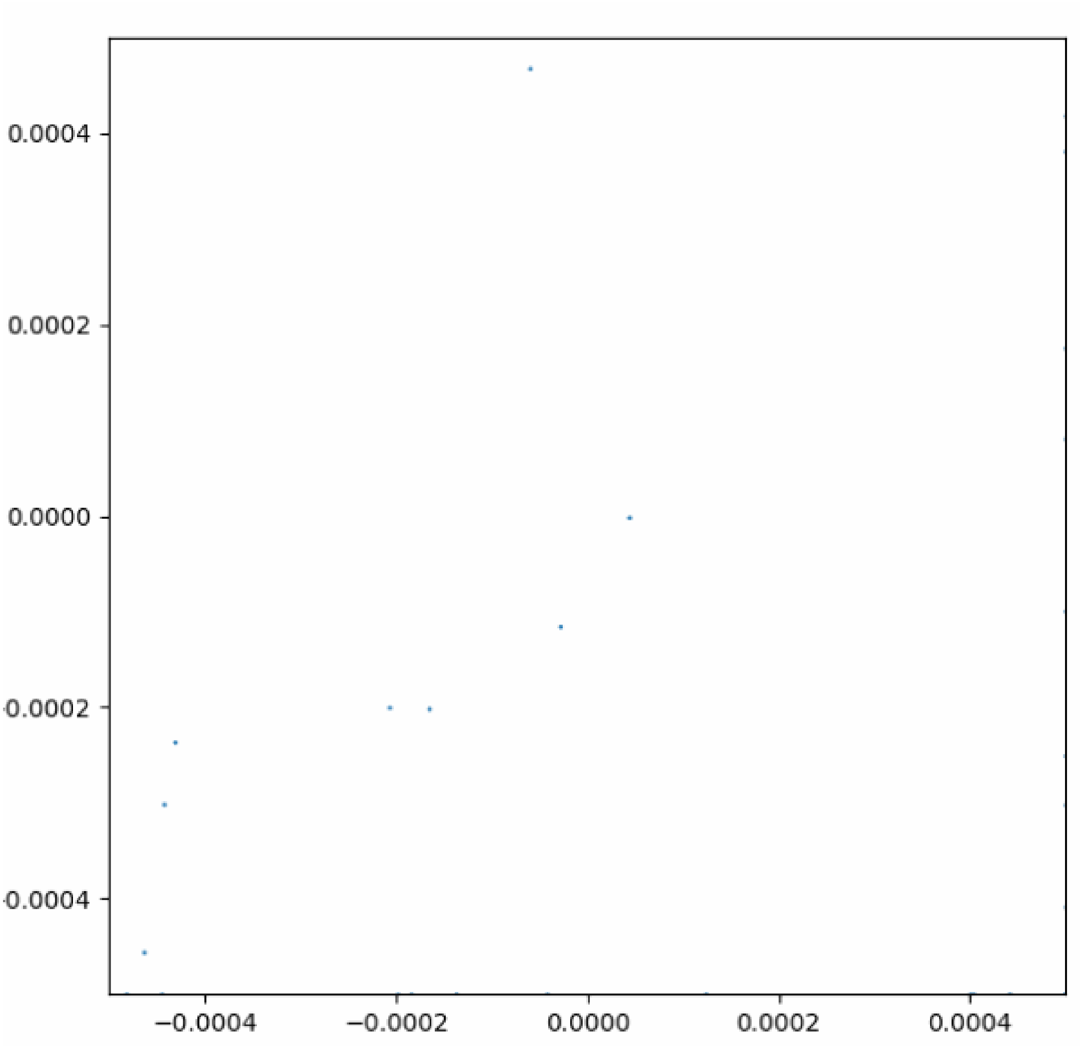

Figure 16: Both x and y-axis are length (m). No. time steps = 100, no. particles = 100, $\Delta t = 10$, $v = 3e-6$, $R = 1e-6$, $D_R = 1$, $D_T = 0.1e-6$.

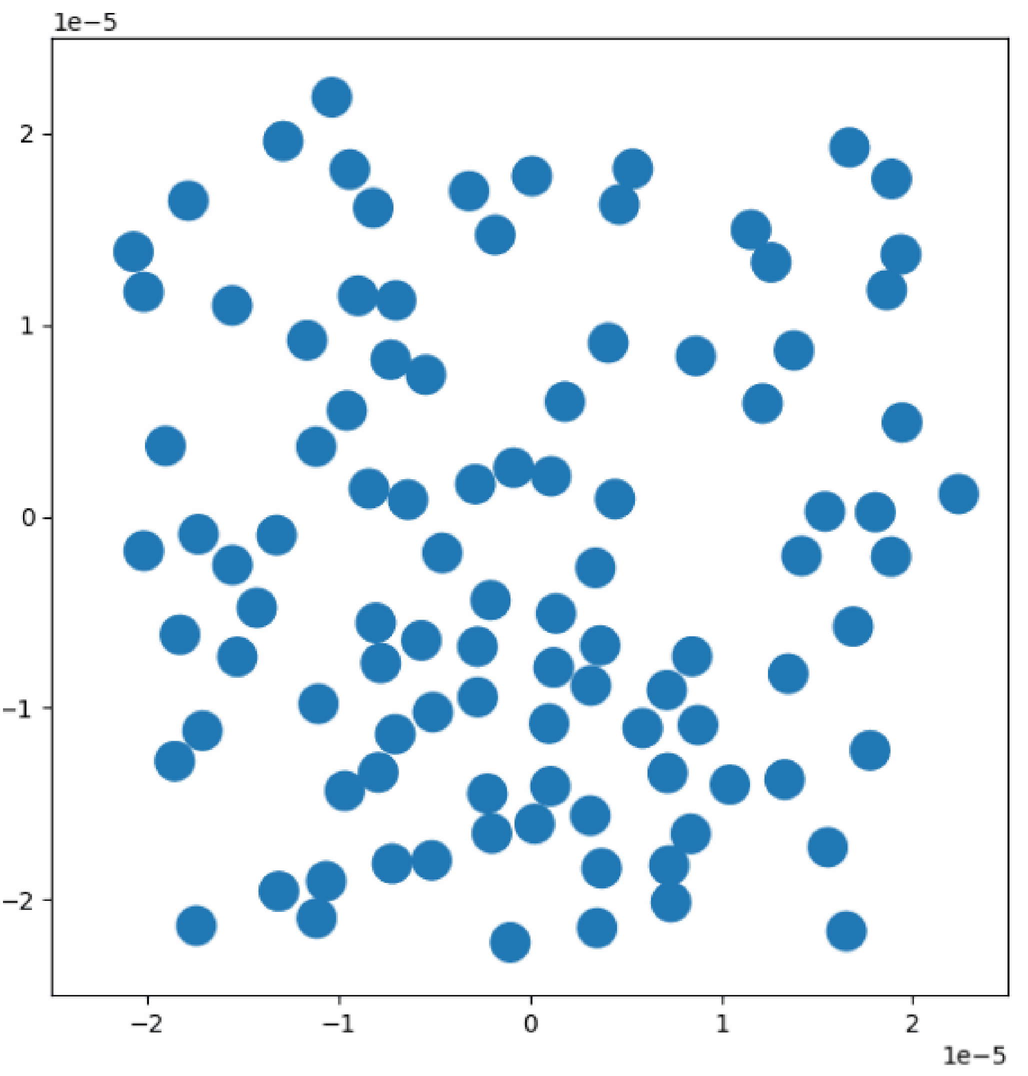

Figure 17: Both x and y-axis are length (m). No. time steps = 1000, no. particles = 100, $\Delta t = 0.000001$, $v = 3e-6$, $R = 1e-6$, $D_R = 1$, $D_T = 0.1e-6$.

For large Δt the particles "jumps" unrealistically and vice versa. Thus, the concentration of the particles is determined by Δt so the choice of Δt is very important for a physical correct behaviour of the simulation.

9.4: a, b

Unfortunately no clustering.

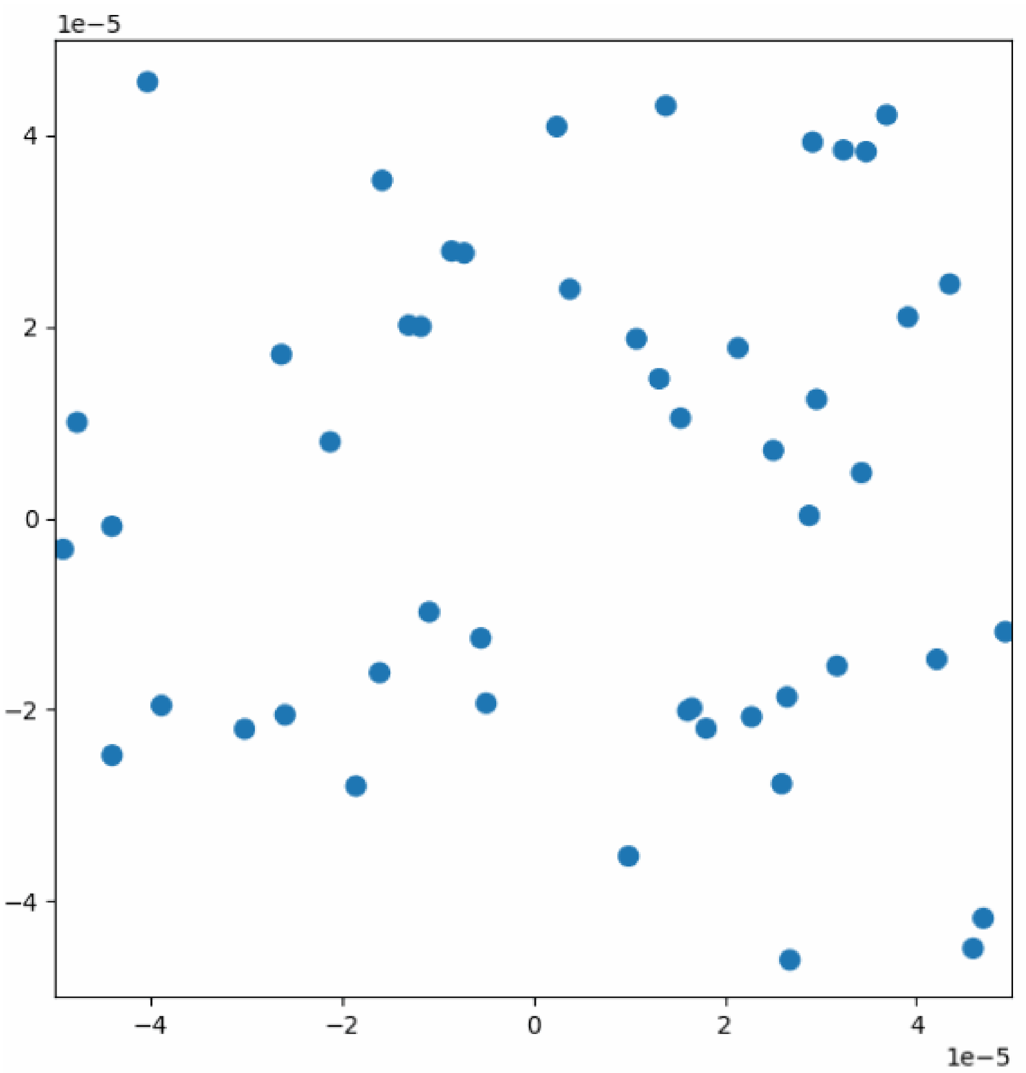

Figure 18: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50, $\Delta t = 0.3$, $v_0 = 20e-6$, $R = 1e-6$.

9.5: a, b

Unfortunately no clustering.

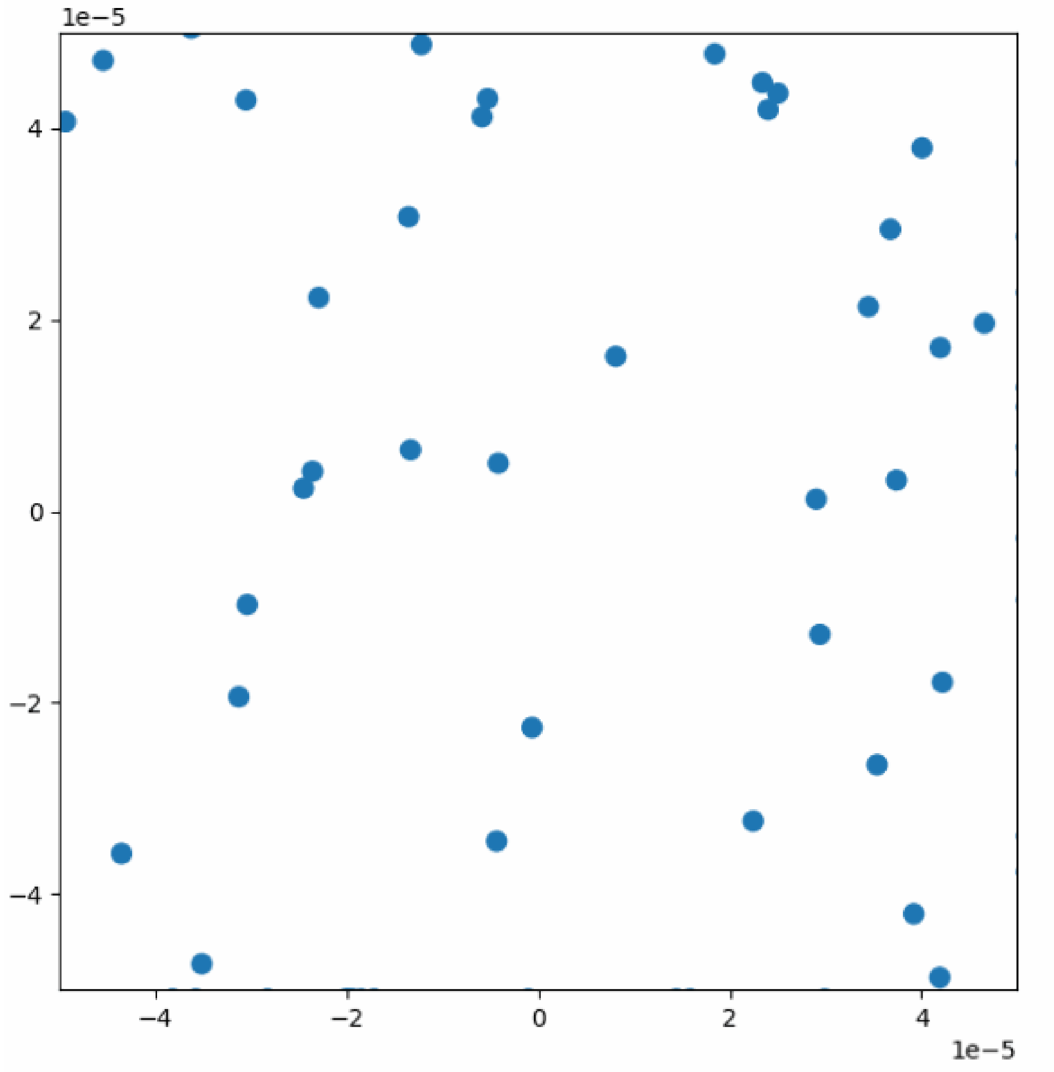

Figure 19: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50, $\Delta t = 0.3$, $v_0 = 0$, $R = 1e - 6$.

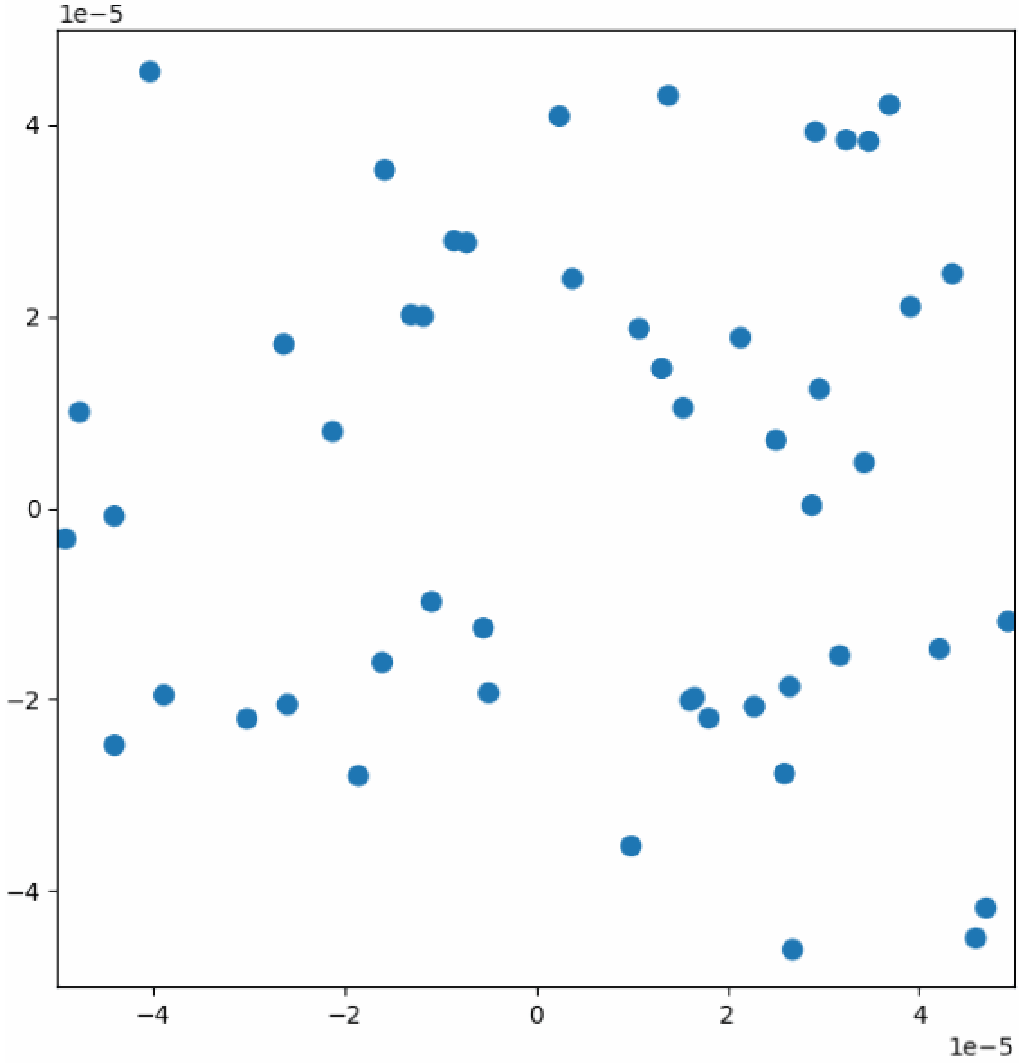

Figure 20: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50, $\Delta t = 0.3$, $v_0 = 20e-6$, $R = 1e-6$.

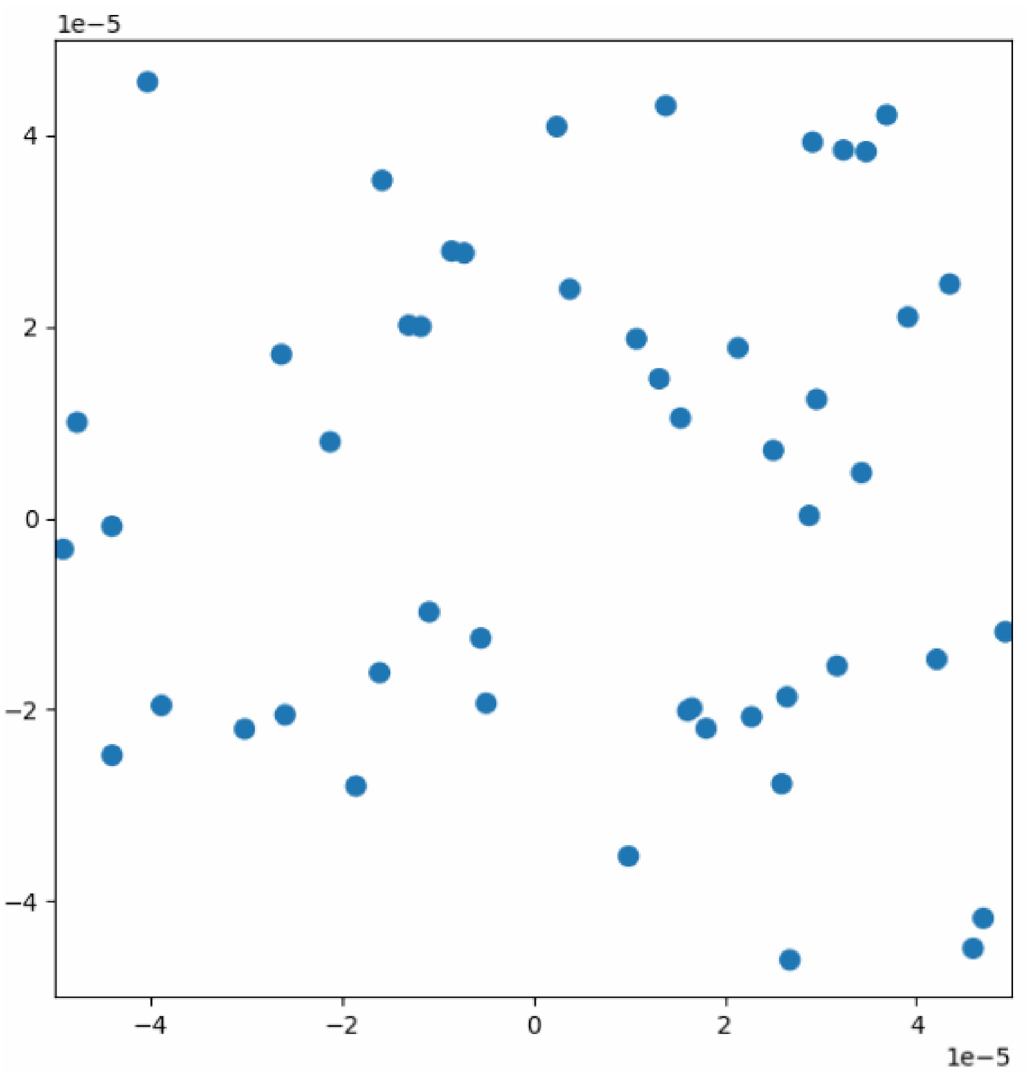

Figure 21: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50, $\Delta t = 0.3$, $v_0 = 50 - 6$, $R = 1e - 6$.

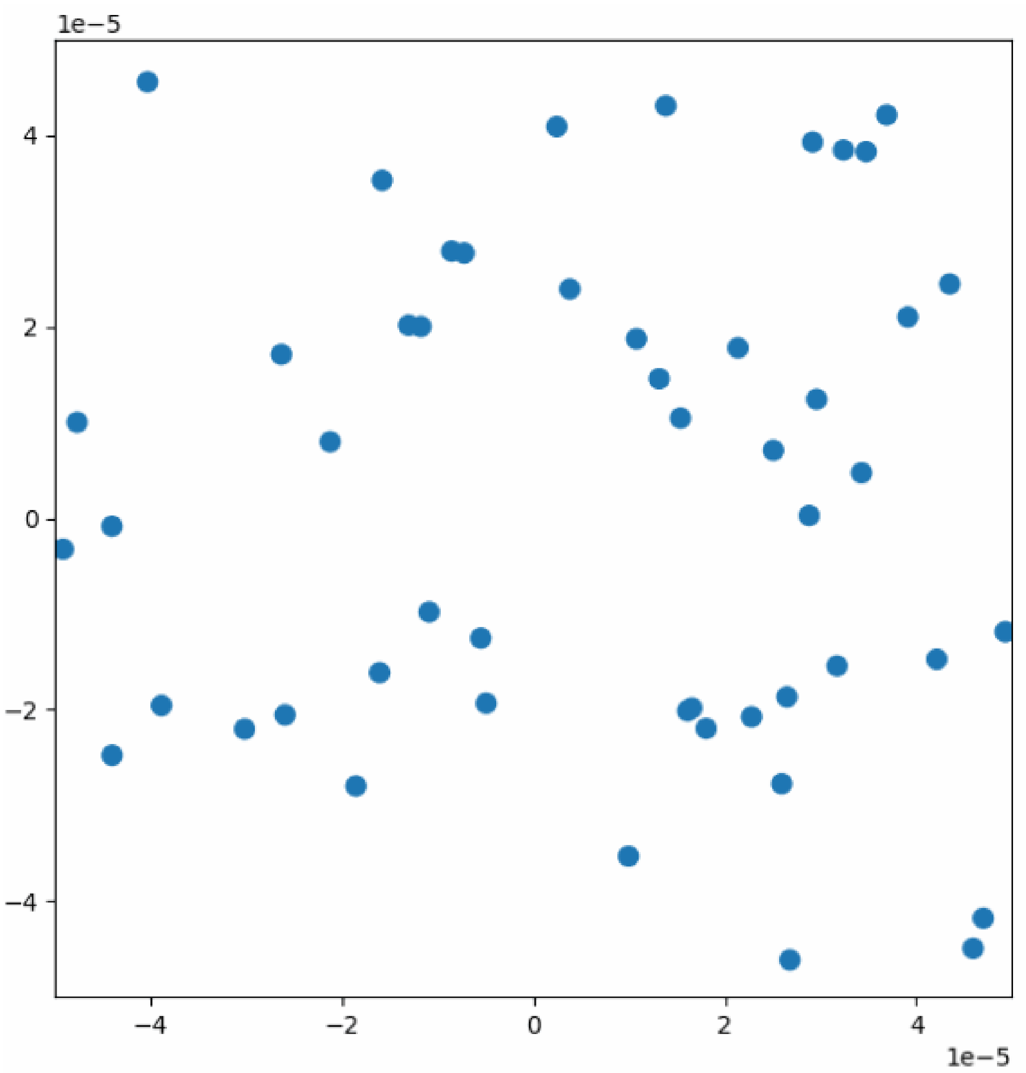

Figure 22: Both x and y-axis are length (m). No. time steps = 500, no. particles = 100, $\Delta t = 0.3$, $v_0 = 50 - 6$, $R = 1e - 6$.

```python
# Exercise 9.1abcd
import numpy as np
import matplotlib.pyplot as plt

T = 500
x = np.zeros((T+1))
y = x.copy()
phi = 0
dt = 0.01
# dt = tau*dt_step
# timesteps = int(tau/dt)*10s
# t = np.linspace(0, timesteps*dt_step, timesteps+1)

# Looping parameter
# Dt = 2*10**-12
# Dt = 5*10**-13 # OG
Dt = 8*10**-14

# Stretching parameter
# Dr = 0.05
# Dr = 0.5 # OG
Dr = 5

# Total length parameter
# v = 0
# v = 1*10**-6
# v = 2*10**-6
v = 3*10**-6 # OG

for t in range(T):

    x[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x[t]
    y[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y[t]
    phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

fig, ax = plt.subplots()

xmax = 2*10**-5
ymax = xmax

ax.plot(x, y, '-', linewidth=3, color='tab:red')
ax.set_xlim([-xmax,xmax])
ax.set_ylim([-ymax,ymax])
ax.set_box_aspect(1)
ax.set_ylabel('y (m)')
ax.set_xlabel('x (m)')

plt.savefig('exercise_9_1ab_v3_DrL.png', bbox_inches='tight')
plt.show()
```

```python
# Exercise 9.2a
import numpy as np
import matplotlib.pyplot as plt
import sys

T = 10000
ensembles = 100
x = np.zeros((ensembles,T+1))
y = x.copy()
phi = 0
dt = 0.01
timesteps = int(T/dt)
t_iterations = np.linspace(0, T*dt, T+1)
# dt = tau*dt_step
# timesteps = int(tau/dt)*10s
# t = np.linspace(0, timesteps*dt_step, timesteps+1)

# Looping parameter
# Dt = 2*10**-12
# Dt = 5*10**-13 # OG
Dt = 8*10**-14

# Stretching parameter
# Dr = 0.05
# Dr = 0.5 # OG
Dr = 5

x_v0 = x.copy()
x_v1 = x.copy()
x_v2 = x.copy()
x_v3 = x.copy()

y_v0 = x.copy()
y_v1 = x.copy()
y_v2 = x.copy()
y_v3 = x.copy()

MSD_v0 = np.zeros((T+1))
MSD_v1 = MSD_v0.copy()
MSD_v2 = MSD_v0.copy()
MSD_v3 = MSD_v0.copy()

for ensemble in range(ensembles):

    v = 0
    for t in range(T):

        x_v0[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x_v0[ensemble,t]
        y_v0[ensemble,t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v0[ensemble,t]
        phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

    v = 1*10**-6
    for t in range(T):

        x_v1[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x_v1[ensemble,t]
```

```python
        y_v1[ensemble,t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v1[ensemble,t]
        phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

    v = 2*10**-6
    for t in range(T):

        x_v2[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x_v2[ensemble,t]
        y_v2[ensemble,t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v2[ensemble,t]
        phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

    v = 3*10**-6
    for t in range(T):

        x_v3[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x_v3[ensemble,t]
        y_v3[ensemble,t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v3[ensemble,t]
        phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

for t in range(T):
    MSD_v0[t+1] = np.sum(x_v0[:,t+1]**2 + y_v0[:,t+1]**2) / ensembles
    MSD_v1[t+1] = np.sum(x_v1[:,t+1]**2 + y_v1[:,t+1]**2) / ensembles
    MSD_v2[t+1] = np.sum(x_v2[:,t+1]**2 + y_v2[:,t+1]**2) / ensembles
    MSD_v3[t+1] = np.sum(x_v3[:,t+1]**2 + y_v3[:,t+1]**2) / ensembles

fig, ax = plt.subplots()

xmax = 2*10**-5
ymax = xmax

ax.plot(t_iterations, MSD_v0, '-', linewidth=1.5, color='tab:blue', label='v=0')
ax.plot(t_iterations, MSD_v1, '-', linewidth=1.5, color='tab:green', label='v=1*1e-6')
ax.plot(t_iterations, MSD_v2, '-', linewidth=1.5, color='tab:orange', label='v=2*1e-6')
ax.plot(t_iterations, MSD_v3, '-', linewidth=1.5, color='tab:red', label='v=3*1e-6')
# ax.set_xlim([-xmax,xmax])
# ax.set_ylim([-ymax,ymax])
ax.set_box_aspect(1)
ax.set_ylabel('MSD (m^2)')
ax.set_xlabel('t (s)')
ax.set_yscale('log')
ax.set_xscale('log')

plt.legend(loc="upper left")
plt.savefig('exercise_9_2a_DtL_DrL_ENSEMBLE.png', bbox_inches='tight')
plt.show()
```

```python
# Exercise 9.2b
import numpy as np
import matplotlib.pyplot as plt
import sys

T = 10000
x = np.zeros((T+1))
y = x.copy()
phi = 0
dt = 0.01
timesteps = int(T/dt)
# dt = tau*dt_step
# timesteps = int(tau/dt)*10s
# t = np.linspace(0, timesteps*dt_step, timesteps+1)

# Looping parameter
Dt = 2*10**-12
# Dt = 5*10**-13 # OG
# Dt = 8*10**-14

# Stretching parameter
Dr = 0.05
# Dr = 0.5 # OG
# Dr = 5

x_v0 = x.copy()
x_v1 = x.copy()
x_v2 = x.copy()
x_v3 = x.copy()

y_v0 = x.copy()
y_v1 = x.copy()
y_v2 = x.copy()
y_v3 = x.copy()

# Simulation
v = 0
for t in range(T):

    x_v0[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
x_v0[t]
    y_v0[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
y_v0[t]
    phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

v = 1*10**-6
for t in range(T):

    x_v1[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
x_v1[t]
    y_v1[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
y_v1[t]
    phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

v = 2*10**-6
for t in range(T):

    x_v2[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
x_v2[t]
```

```
    y_v2[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v2[t]
    phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

v = 3*10**-6
for t in range(T):

    x_v3[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x_v3[t]
    y_v3[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y_v3[t]
    phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi

# Time averged MSD
t_range = int(T/2 + 1)
t_iterations = np.linspace(0, T*dt, t_range)

MSD_v0_time = np.zeros((t_range,t_range))
MSD_v1_time = MSD_v0_time.copy()
MSD_v2_time = MSD_v0_time.copy()
MSD_v3_time = MSD_v0_time.copy()

MSD_v0 = np.zeros((t_range))
MSD_v1 = MSD_v0.copy()
MSD_v2 = MSD_v0.copy()
MSD_v3 = MSD_v0.copy()

for i in range(len(x_v0)-t_range+1):

    x_row_v0 = x_v0[i:i+t_range]
    y_row_v0 = y_v0[i:i+t_range]
    for j in range(len(y_row_v0)):
        MSD_v0_time[i,j] = (x_row_v0[j] - x_row_v0[0])**2 + (y_row_v0[j] - y_row_v0[0])**2

    x_row_v1 = x_v1[i:i+t_range]
    y_row_v1 = y_v1[i:i+t_range]
    for j in range(len(y_row_v1)):
        MSD_v1_time[i,j] = (x_row_v1[j] - x_row_v1[0])**2 + (y_row_v1[j] - y_row_v1[0])**2

    x_row_v2 = x_v2[i:i+t_range]
    y_row_v2 = y_v2[i:i+t_range]
    for j in range(len(y_row_v2)):
        MSD_v2_time[i,j] = (x_row_v2[j] - x_row_v2[0])**2 + (y_row_v2[j] - y_row_v2[0])**2

    x_row_v3 = x_v3[i:i+t_range]
    y_row_v3 = y_v3[i:i+t_range]
    for j in range(len(y_row_v3)):
        MSD_v3_time[i,j] = (x_row_v3[j] - x_row_v3[0])**2 + (y_row_v3[j] - y_row_v3[0])**2

for i in range(t_range):
    MSD_v0[i] = np.sum(MSD_v0_time[:,i])
    MSD_v1[i] = np.sum(MSD_v1_time[:,i])
    MSD_v2[i] = np.sum(MSD_v2_time[:,i])
    MSD_v3[i] = np.sum(MSD_v3_time[:,i])

MSD_v0 = MSD_v0 / t_range
```

```python
MSD_v1 = MSD_v1 / t_range
MSD_v2 = MSD_v2 / t_range
MSD_v3 = MSD_v3 / t_range

# t = 0
# for i in range(t_range):
#     for j in range(t_range):

#         MSD_v0[i] += (x_v0[t+j] - x_v0[t])**2 + (y_v0[t+j] - y_v0[t])**2
#         t += 1

# MSD_v0 = MSD_v0 / t_range

fig, ax = plt.subplots()

xmax = 2*10**-5
ymax = xmax

ax.plot(t_iterations, MSD_v0, '-', linewidth=1.5, color='tab:blue', label='v=0')
ax.plot(t_iterations, MSD_v1, '-', linewidth=1.5, color='tab:green', label='v=1*1e-6')
ax.plot(t_iterations, MSD_v2, '-', linewidth=1.5, color='tab:orange', label='v=2*1e-6')
ax.plot(t_iterations, MSD_v3, '-', linewidth=1.5, color='tab:red', label='v=3*1e-6')
# ax.set_xlim([-xmax,xmax])
# ax.set_ylim([-ymax,ymax])
ax.set_box_aspect(1)
ax.set_ylabel('MSD (m^2)')
ax.set_xlabel('t (s)')
ax.set_yscale('log')
ax.set_xscale('log')

plt.legend(loc="upper left")
plt.savefig('exercise_9_2b_DtS_DrS_TIME.png', bbox_inches='tight')
plt.show()
```

```python
# Exercise 9.3a,b,c,d
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.animation import PillowWriter
import sys

T = 1000
dt = 0.000001
timesteps = int(T/dt)
t_iterations = np.linspace(0, T*dt, T+1)
Dt = 0.1*10**-6
# Dt = 0.1*10**-8
Dr = 1
# Dr = 0.01
v = 3*10**-6
R = 1*10**-6

N = 100 # No. particles
x = (np.random.rand(N,T+1)-0.5)*4*10**-5
y = (np.random.rand(N,T+1)-0.5)*4*10**-5
# d = np.zeros((N))
phi = (np.random.rand(N)-0.5)*2*np.pi

xmax = 2.5*10**-5
ymax = xmax
boundary_condition = xmax + R
marker_size = 20
fps_ani = 200
interval_ani = 0.01

fig, ax = plt.subplots(figsize=(7,7))

for t in range(T):

    # Computing new positions
    for n in range(N):

        x[n,t+1] = (v*np.cos(phi[n])*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + x[n,t]
        y[n,t+1] = (v*np.sin(phi[n])*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) + y[n,t]
        phi[n] = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi[n]

    # Applying volume extraciton
    for n in range(N):

        distances = ((x[:,t+1]-x[n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
        angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])
        overlapp = distances < (2*R)
        overlapp[n] = False

        for i in np.where(overlapp)[0]:
            x[n,t+1] = x[n,t+1] + (distances[i] - 2*R)*np.cos(angles[i])/2
            y[n,t+1] = y[n,t+1] + (distances[i] - 2*R)*np.sin(angles[i])/2
            x[i,t+1] = x[i,t+1] - (distances[i] - 2*R)*np.cos(angles[i])/2
            y[i,t+1] = y[i,t+1] - (distances[i] - 2*R)*np.sin(angles[i])/2

        # Boundary condition
```

```python
        if x[n,t+1] > boundary_condition:
            x[n,t+1] = -boundary_condition

        elif x[n,t+1] < -boundary_condition:
            x[n,t+1] = boundary_condition

        if y[n,t+1] > boundary_condition:
            y[n,t+1] = -boundary_condition

        elif y[n,t+1] < -boundary_condition:
            y[n,t+1] = boundary_condition

def animate(i):
    ax.clear()
    for n in range(N):
        x_point = x[n,i]
        y_point = y[n,i]
        # ax.plot(x_point, y_point, color='tab:blue', marker='.', markersize=marker_size)
        ax.add_patch(plt.Circle([x_point, y_point], radius=R, color='tab:blue'))
    ax.set_xlim([-xmax, xmax])
    ax.set_ylim([-ymax, ymax])

ax.set_ylabel('y (m)')
ax.set_xlabel('x (m)')
ax.set_box_aspect(1)
ani = FuncAnimation(fig, animate, frames=len(x[0,:]), interval=interval_ani)
ani.save('exercise_9_3c_VE_dt0.000001_T1000_Dt0.1e-6_Dr1.gif', writer=PillowWriter(fps=fps_ani))
plt.show()
```

```python
# Exercise 9.4, 9.5
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.animation import PillowWriter
import sys

T = 500
dt = 0.3
timesteps = int(T/dt)
t_iterations = np.linspace(0, T*dt, T+1)
R = 1*10**-6
gamma = 6*np.pi*0.001*R
kB = 1.380649*10**-23
Dt = kB*300/gamma
Dt2 = kB*300/(8*np.pi*0.001*R**3)
# Dt = 0.1*10**-6
# Dt = 0.1*10**-8
Dr = 1
# Dr = 0.01
v = 3*10**-6

N = 50 # No. particles
x = (np.random.rand(N,T+1)-0.5)*8*10**-5
y = (np.random.rand(N,T+1)-0.5)*8*10**-5
phi = (np.random.rand(N)-0.5)*2*np.pi

vpx = np.zeros(N)
vpy = np.zeros(N)
v0 = 50*10**-6

xmax = 50*10**-6
ymax = xmax
boundary_condition = xmax + R
marker_size = 20
fps_ani = 200
interval_ani = 0.01

fig, ax = plt.subplots(figsize=(7,7))

for t in range(T):

    # Computing new positions
    for n in range(N):

        x[n,t+1] = v*np.cos(phi[n])*dt + ((2*Dt*dt)**0.5)*np.random.normal(0,1) +
vpx[n]*dt + x[n,t]
        y[n,t+1] = v*np.sin(phi[n])*dt + ((2*Dt*dt)**0.5)*np.random.normal(0,1) +
vpy[n]*dt + y[n,t]
        phi[n] = ((2*Dr*Dt2*dt)**0.5)*np.random.normal(0,1) + phi[n]
        # phi[n] = ((2*Dr*dt)**0.5)*np.random.normal(0,1) + phi[n]

    vpx = np.zeros(N)
    vpy = np.zeros(N)

    for n in range(N):

        distances = ((x[:,t+1]-x[n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
        angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])
```

```
        interact = distances < 5*R
        interact[n] = False

        for i in np.where(interact)[0]:
            vpx[n] = vpx[n] + (v0*R**2 / distances[i]**2) * np.cos(angles[i]) * distances[i]
            vpy[n] = vpx[n] + (v0*R**2 / distances[i]**2) * np.sin(angles[i]) * distances[i]

    # Applying volume extraciton
    # for i in range(3):
    for n in range(N):

        distances = ((x[:,t+1]-x[-n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
        angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])
        overlap = distances < (2*R)
        overlap[n] = False

        for i in np.where(overlap)[0]:
            x[n,t+1] = x[n,t+1] + (distances[i] - 2*R)*np.cos(angles[i])/2
            y[n,t+1] = y[n,t+1] + (distances[i] - 2*R)*np.sin(angles[i])/2
            x[i,t+1] = x[i,t+1] - (distances[i] - 2*R)*np.cos(angles[i])/2
            y[i,t+1] = y[i,t+1] - (distances[i] - 2*R)*np.sin(angles[i])/2

        # Boundary condition
        if x[n,t+1] > boundary_condition:
            x[n,t+1] = -boundary_condition

        elif x[n,t+1] < -boundary_condition:
            x[n,t+1] = boundary_condition

        if y[n,t+1] > boundary_condition:
            y[n,t+1] = -boundary_condition

        elif y[n,t+1] < -boundary_condition:
            y[n,t+1] = boundary_condition

def animate(i):
    ax.clear()
    for n in range(N):
        x_point = x[n,i]
        y_point = y[n,i]
        ax.add_patch(plt.Circle([x_point, y_point], radius=R, color='tab:blue'))
    ax.set_xlim([-xmax, xmax])
    ax.set_ylim([-ymax, ymax])

ax.set_ylabel('y (m)')
ax.set_xlabel('x (m)')
ax.set_box_aspect(1)
ani = FuncAnimation(fig, animate, frames=len(x[0,:]), interval=interval_ani)
ani.save('exercise_9_5_v0_50_test.gif', writer=PillowWriter(fps=fps_ani))
plt.show()
```