

# 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.

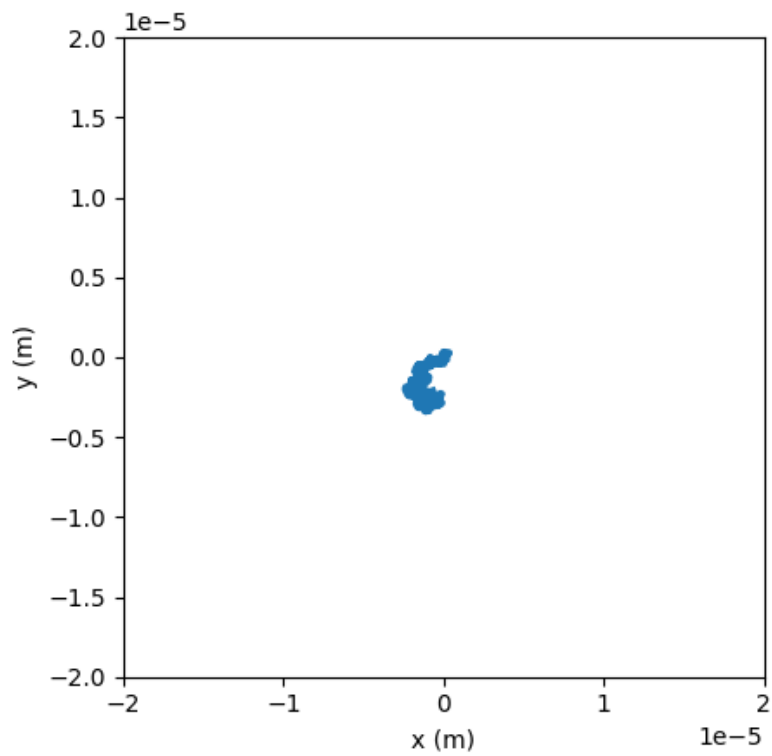


Figure 1:  $v = 0$ ,  $D_R = 5$  and  $D_T = 8e - 14$ .

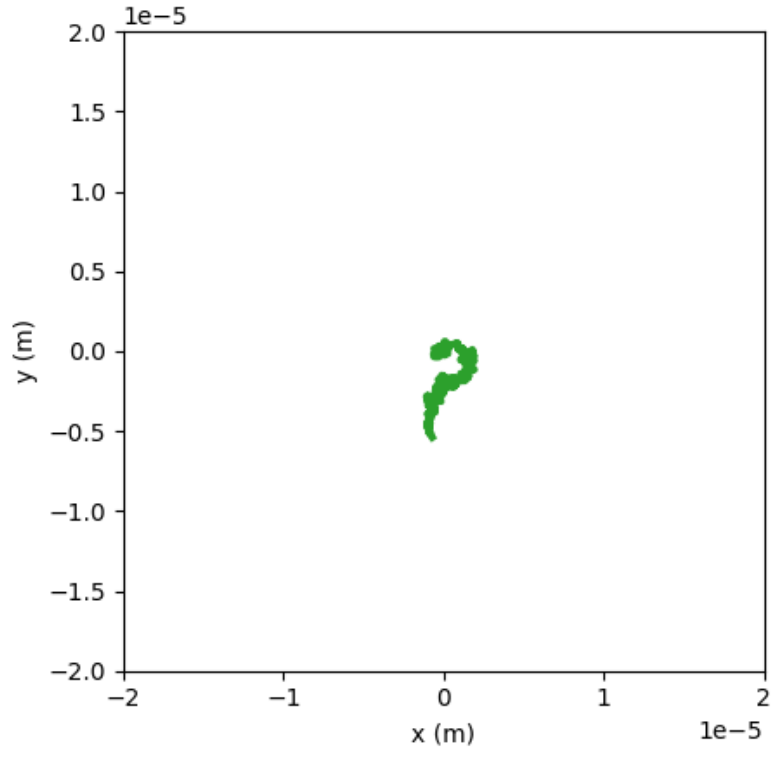


Figure 2:  $v = 1e - 6$ ,  $D_R = 5$  and  $D_T = 8e - 14$ .

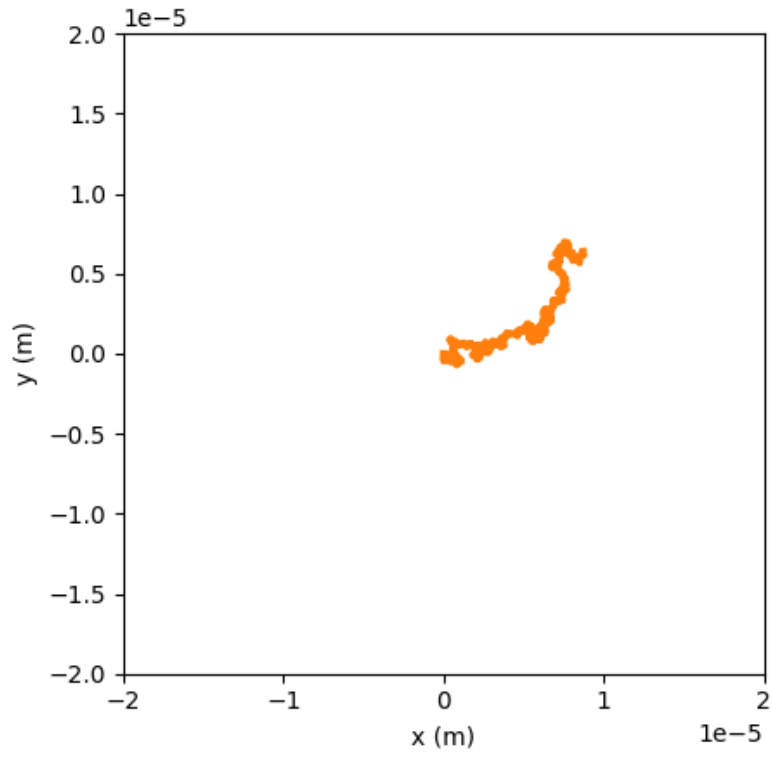


Figure 3:  $v = 2e - 6$ ,  $D_R = 5$  and  $D_T = 8e - 14$ .

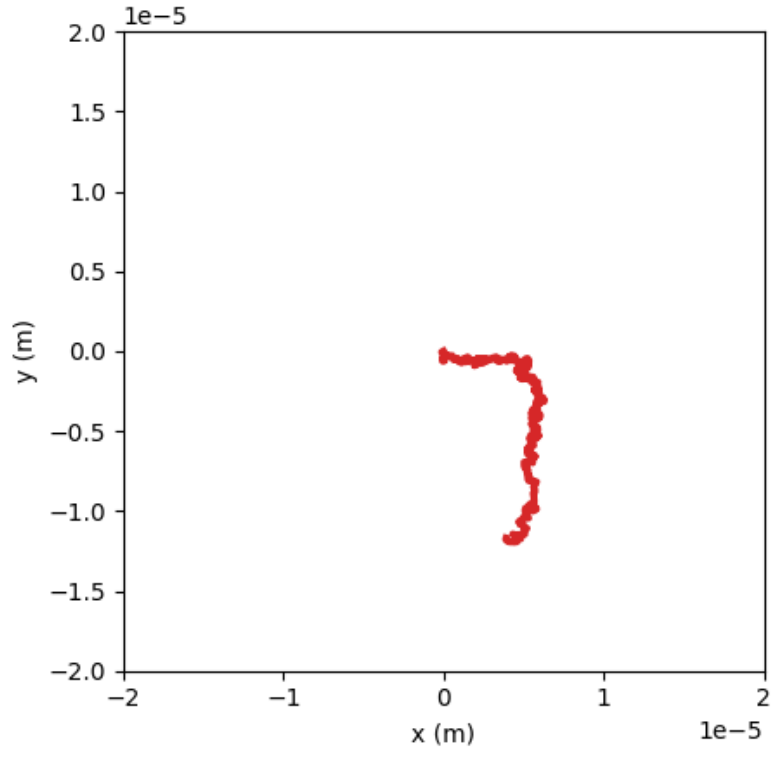


Figure 4:  $v = 3e - 6$ ,  $D_R = 5$  and  $D_T = 8e - 14$ .

### 9.1: c ( $D_T$ change)

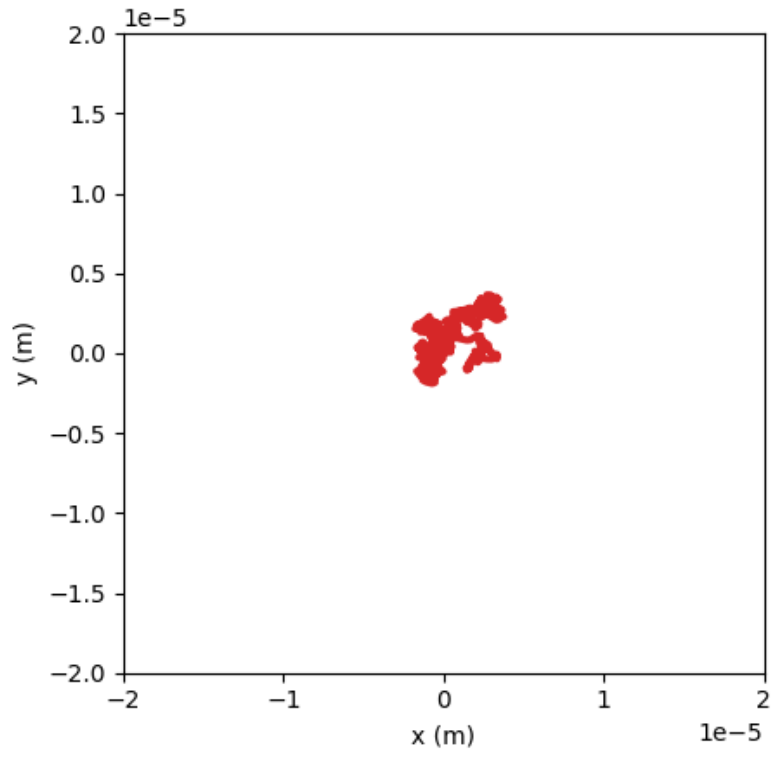


Figure 5:  $v = 3e - 6$ ,  $D_R = 0.5$  and  $D_T = 2e - 12$ .

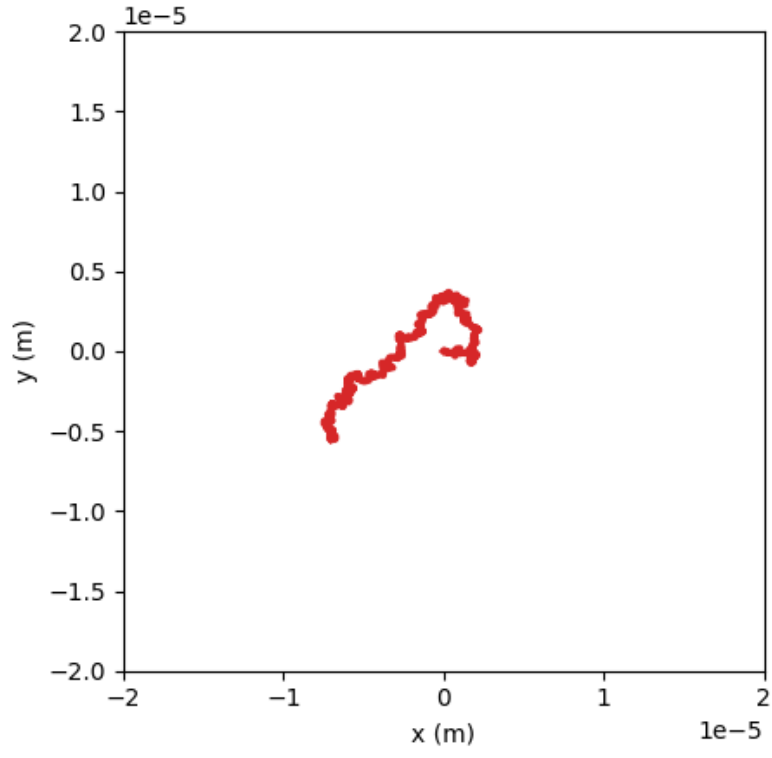


Figure 6:  $v = 3e - 6$ ,  $D_R = 0.5$  and  $D_T = 5e - 13$ .

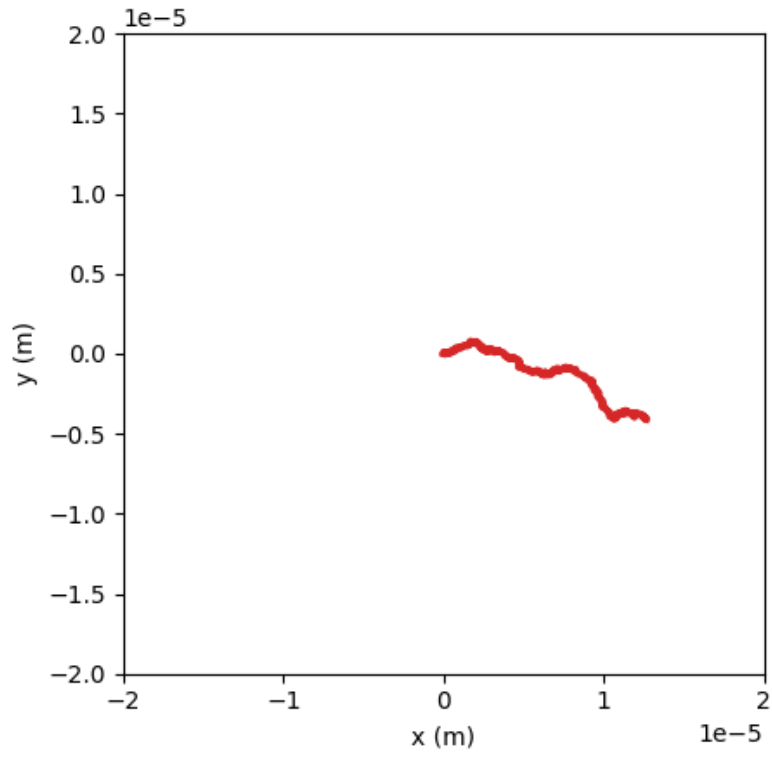


Figure 7:  $v = 3e - 6$ ,  $D_R = 0.5$  and  $D_T = 8e - 14$ .

### 9.1: d ( $D_R$ change)

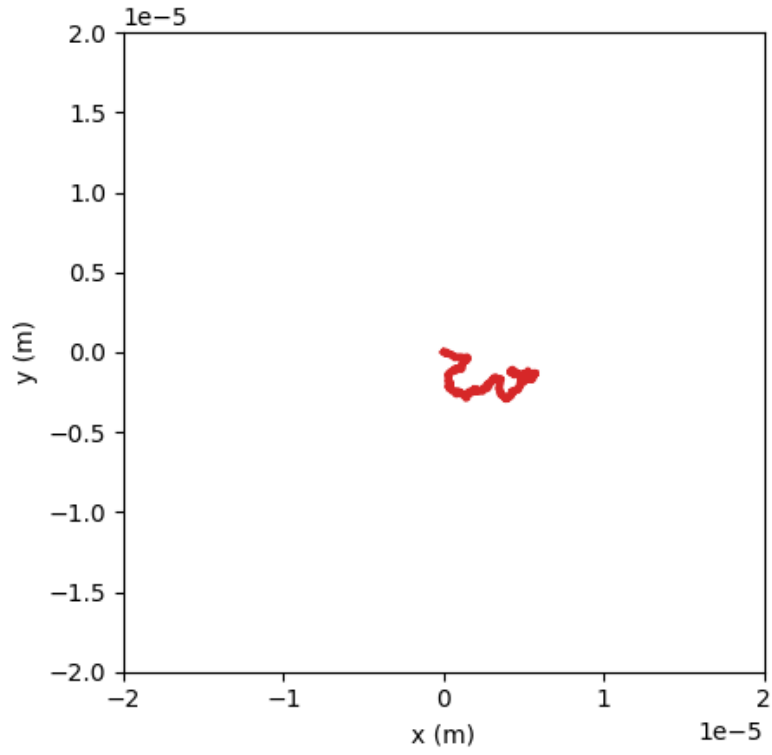


Figure 8:  $v = 3e - 6$ ,  $D_R = 5$  and  $D_T = 8e - 14$ .

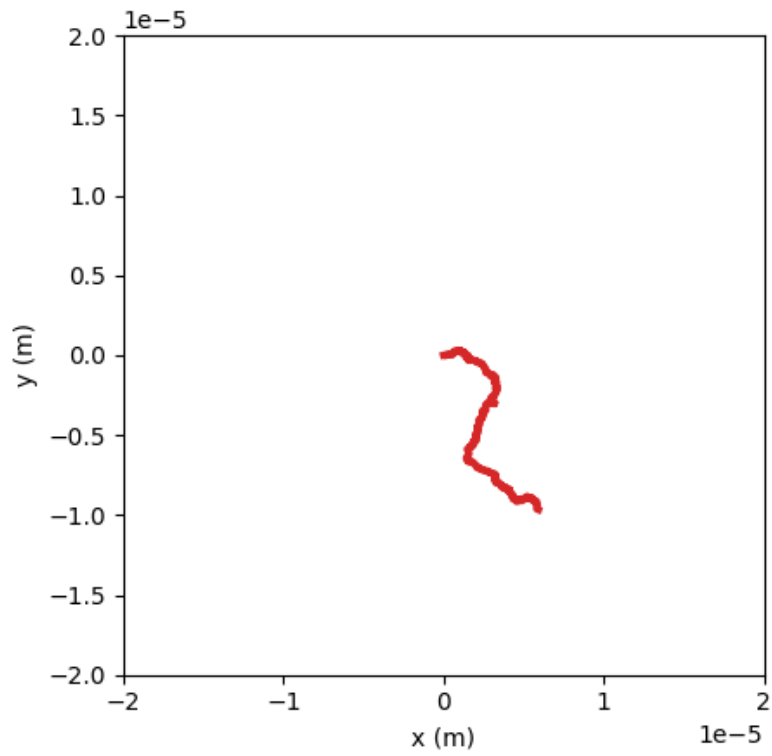


Figure 9:  $v = 3e - 6$ ,  $D_R = 0.5$  and  $D_T = 8e - 14$ .

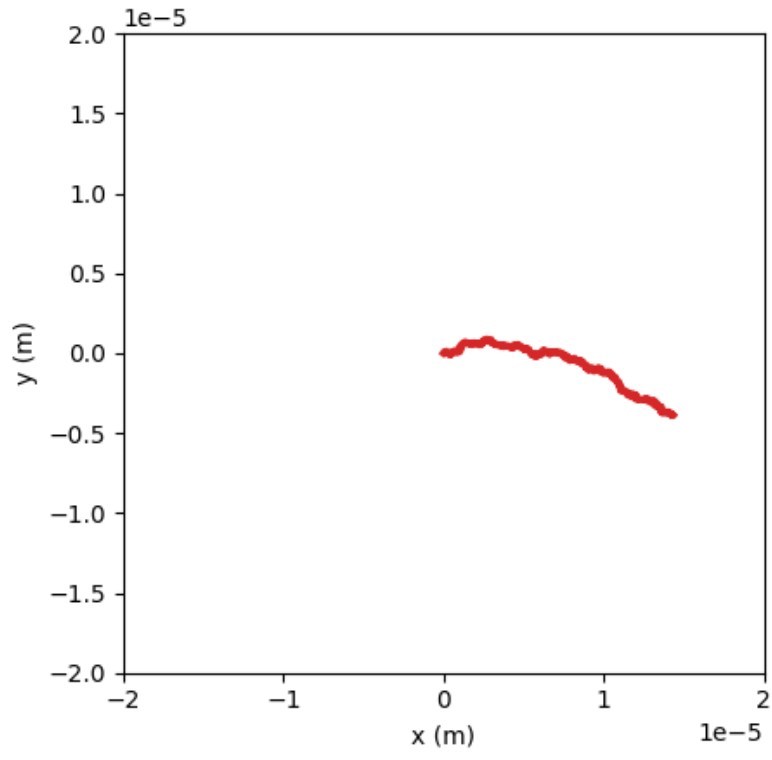


Figure 10:  $v = 3e - 6$ ,  $D_R = 0.05$  and  $D_T = 8e - 14$ .

## 9.2: a, b, c

10000 iterations for all runs in 9.2.

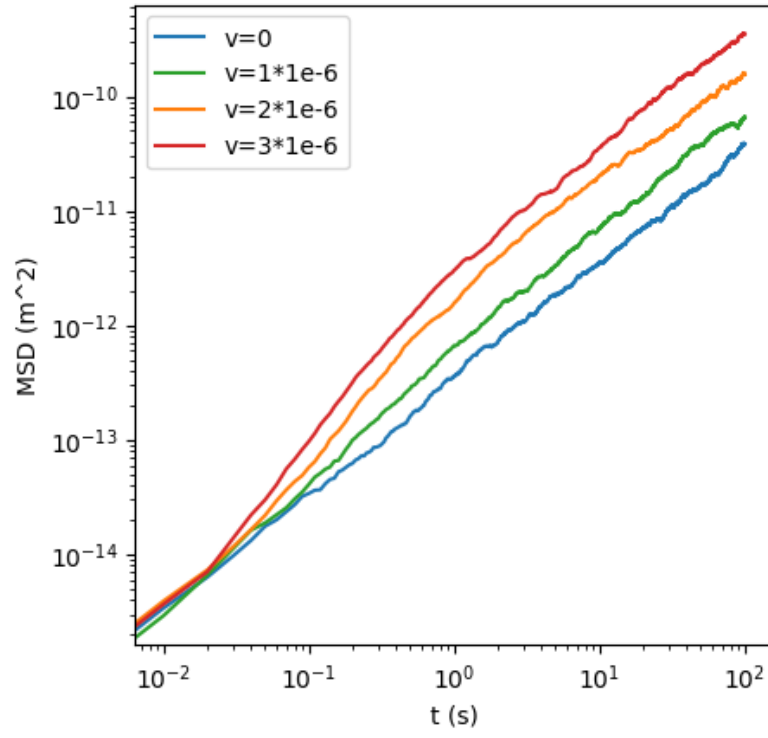


Figure 11: Ensemble averaged MSD.  $D_R = 5$  and  $D_T = 8e - 14$ .

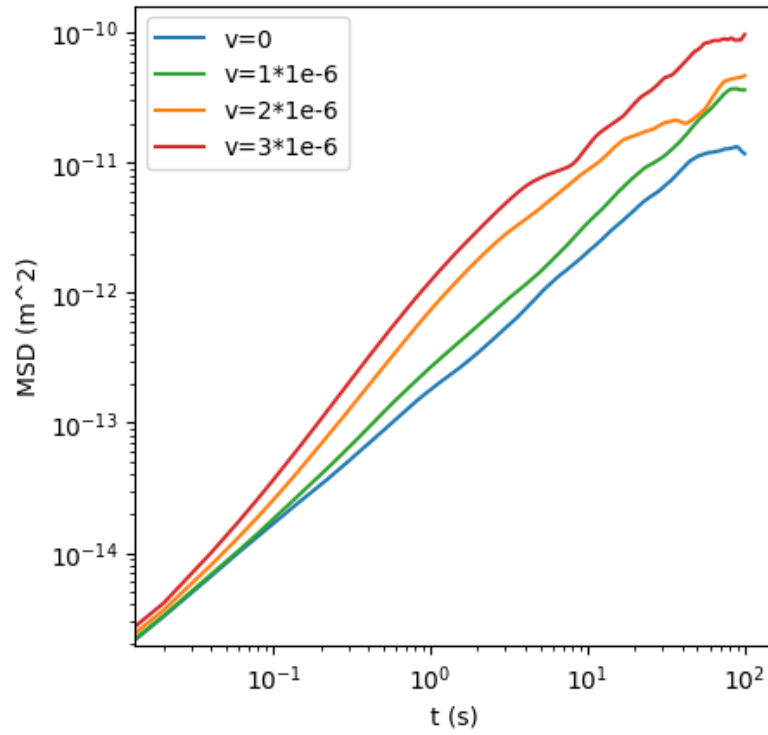


Figure 12: Time averaged MSD.  $D_R = 5$  and  $D_T = 8e - 14$ .

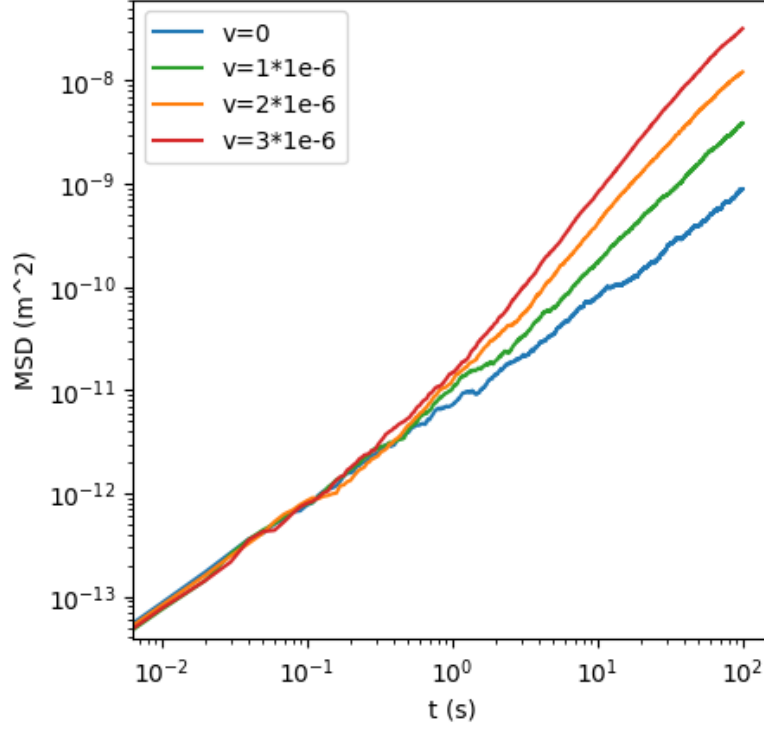


Figure 13: Ensemble averaged MSD.  $D_R = 0.05$  and  $D_T = 2e - 12$ .

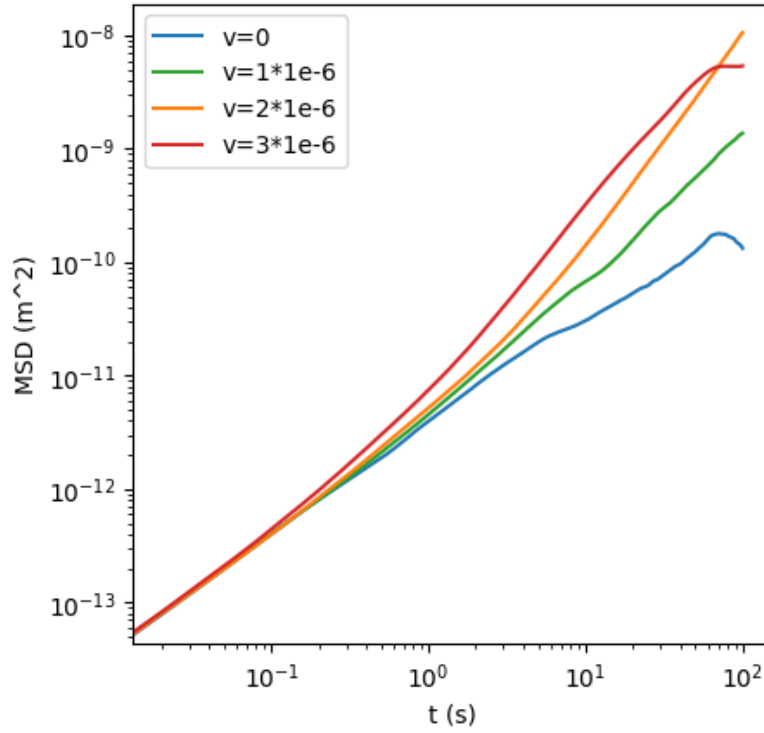


Figure 14: Time averaged MSD.  $D_R = 0.05$  and  $D_T = 2e - 12$ .



9.3: a, b, c, d

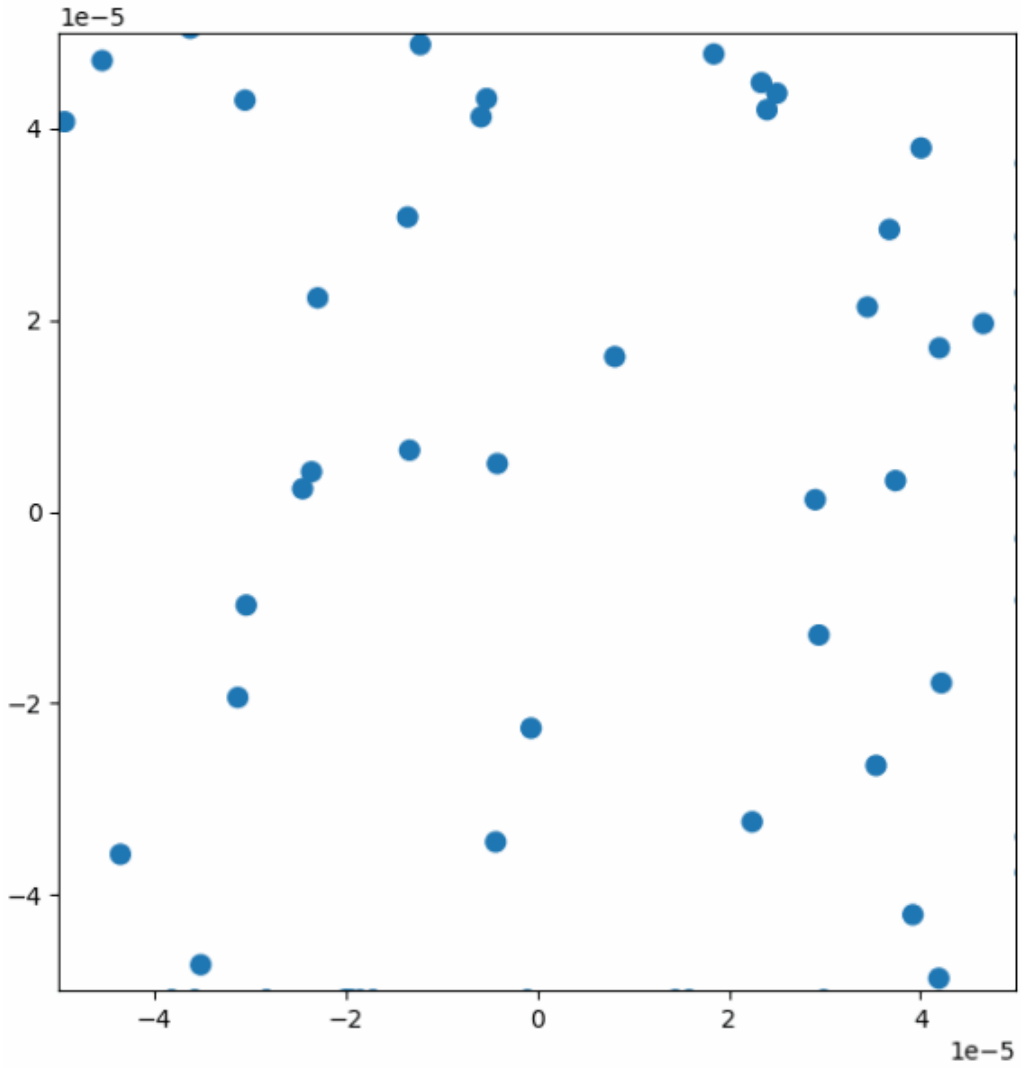


Figure 15: Both x and y-axis are length (m). No. time steps = 1000, no. particles = 100,  $\Delta t = 0.01$ ,  $v = 3e - 6$ ,  $R = 1e - 6$ ,  $D_R = 1$ ,  $D_T = 0.1e - 6$ .

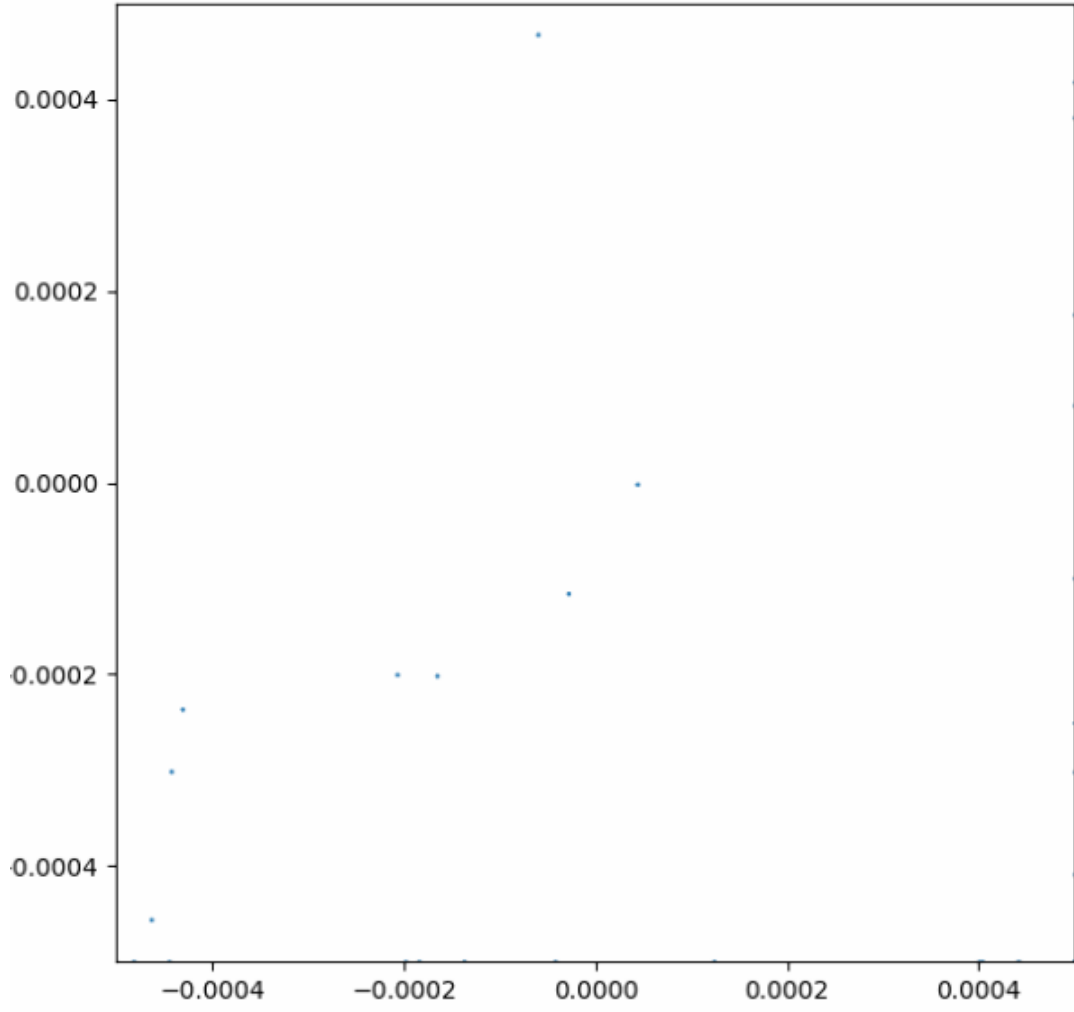


Figure 16: Both x and y-axis are length (m). No. time steps = 100, no. particles = 100,  $\Delta t = 10$ ,  $v = 3e - 6$ ,  $R = 1e - 6$ ,  $D_R = 1$ ,  $D_T = 0.1e - 6$ .

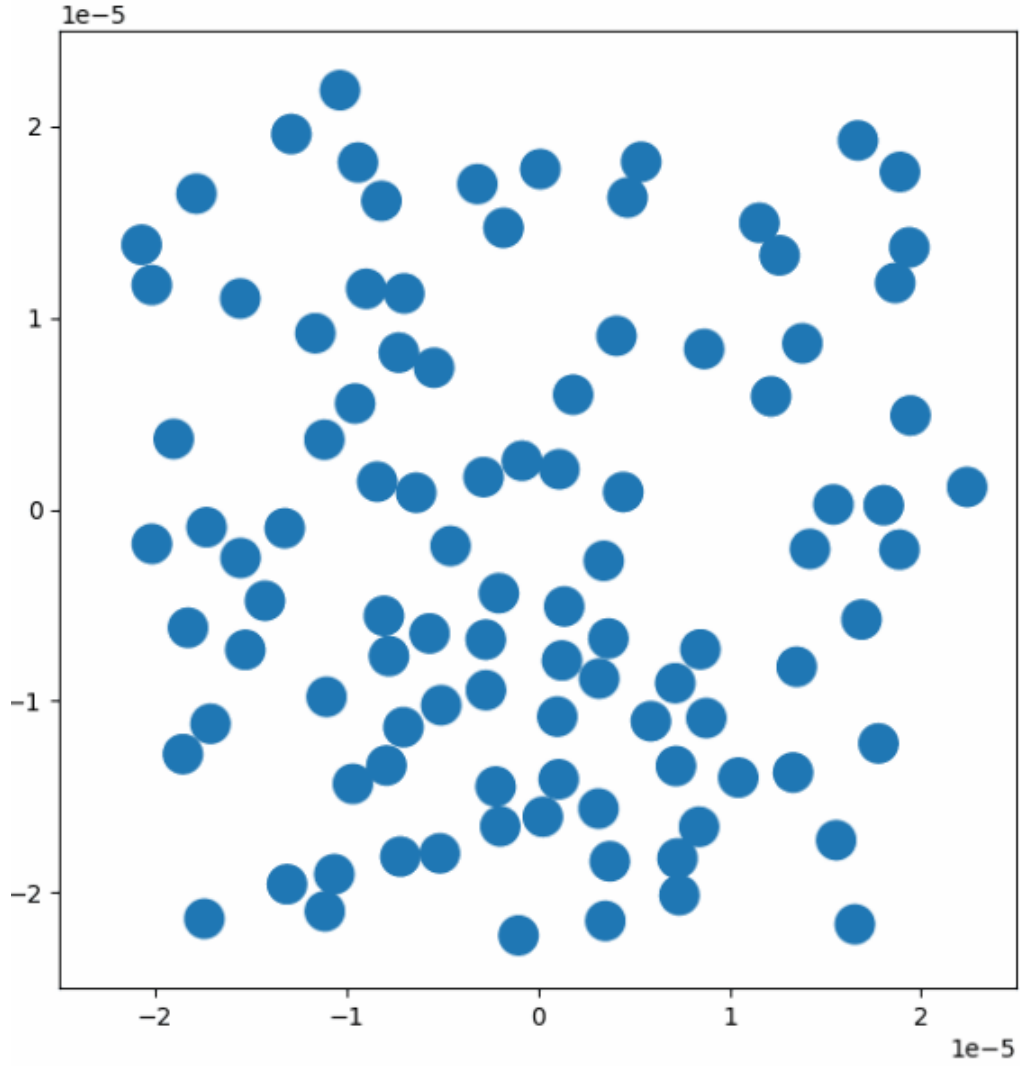


Figure 17: Both x and y-axis are length (m). No. time steps = 1000, no. particles = 100,  $\Delta t = 0.000001$ ,  $v = 3e - 6$ ,  $R = 1e - 6$ ,  $D_R = 1$ ,  $D_T = 0.1e - 6$ .

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

#### 9.4: a, b

Unfortunately no clustering.

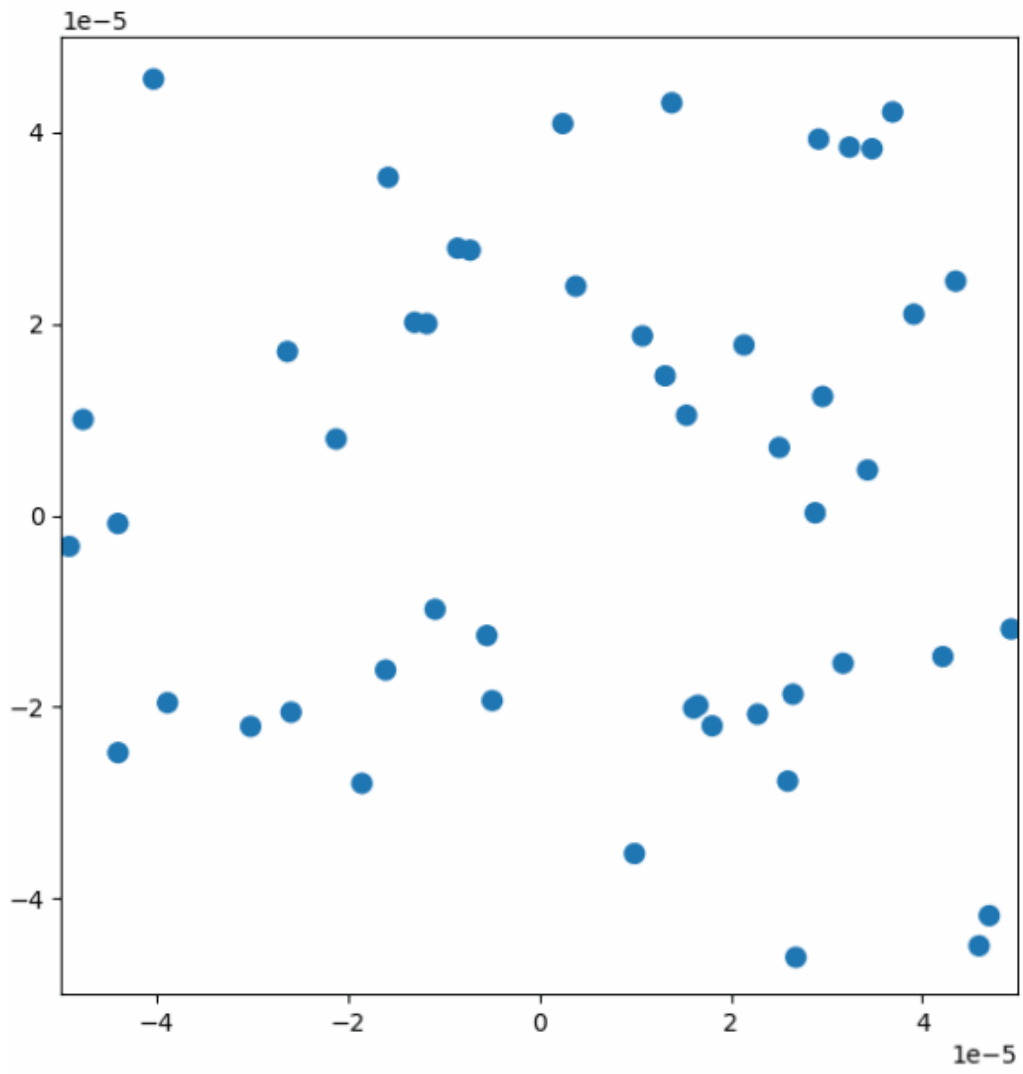


Figure 18: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50,  $\Delta t = 0.3$ ,  $v_0 = 20e - 6$ ,  $R = 1e - 6$ .

### 9.5: a, b

Unfortunately no clustering.

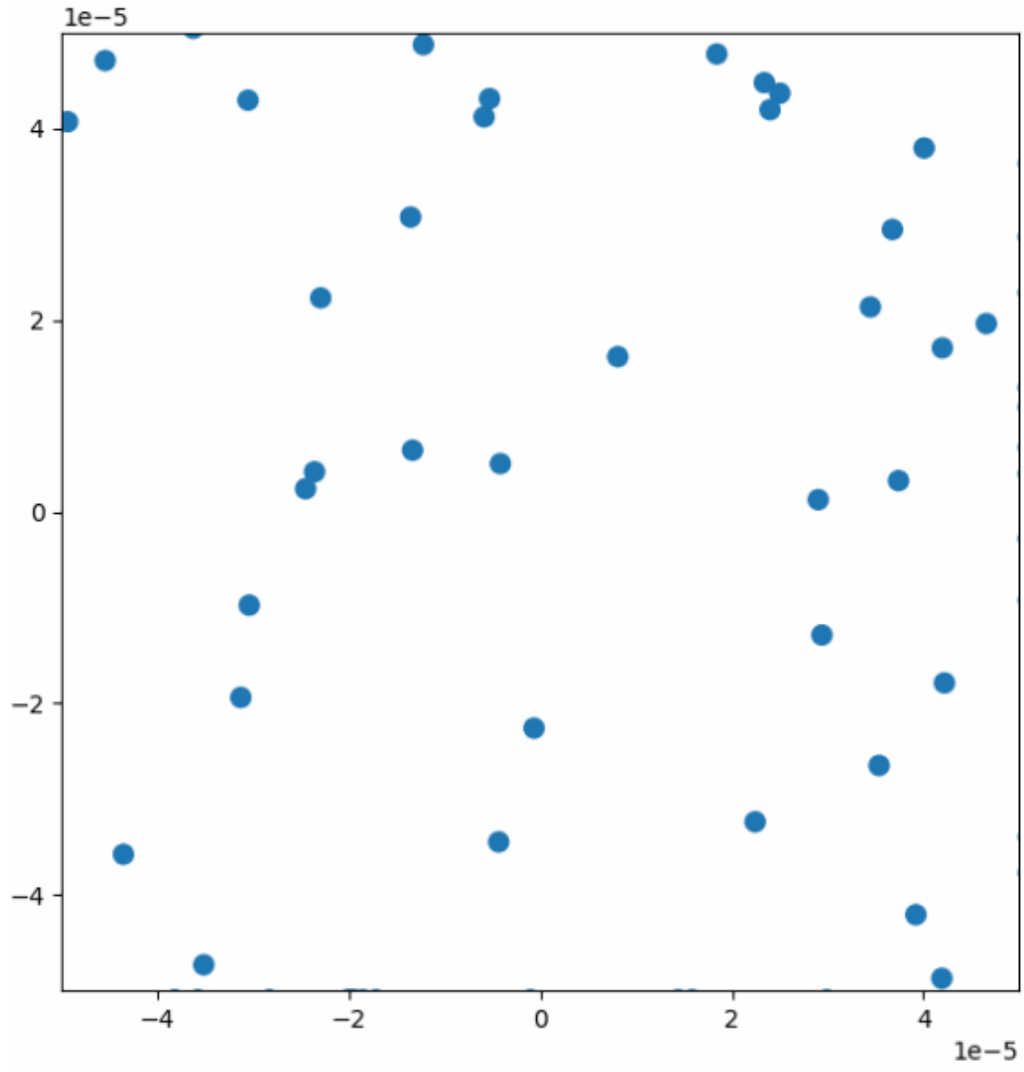


Figure 19: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50,  $\Delta t = 0.3$ ,  $v_0 = 0$ ,  $R = 1e - 6$ .

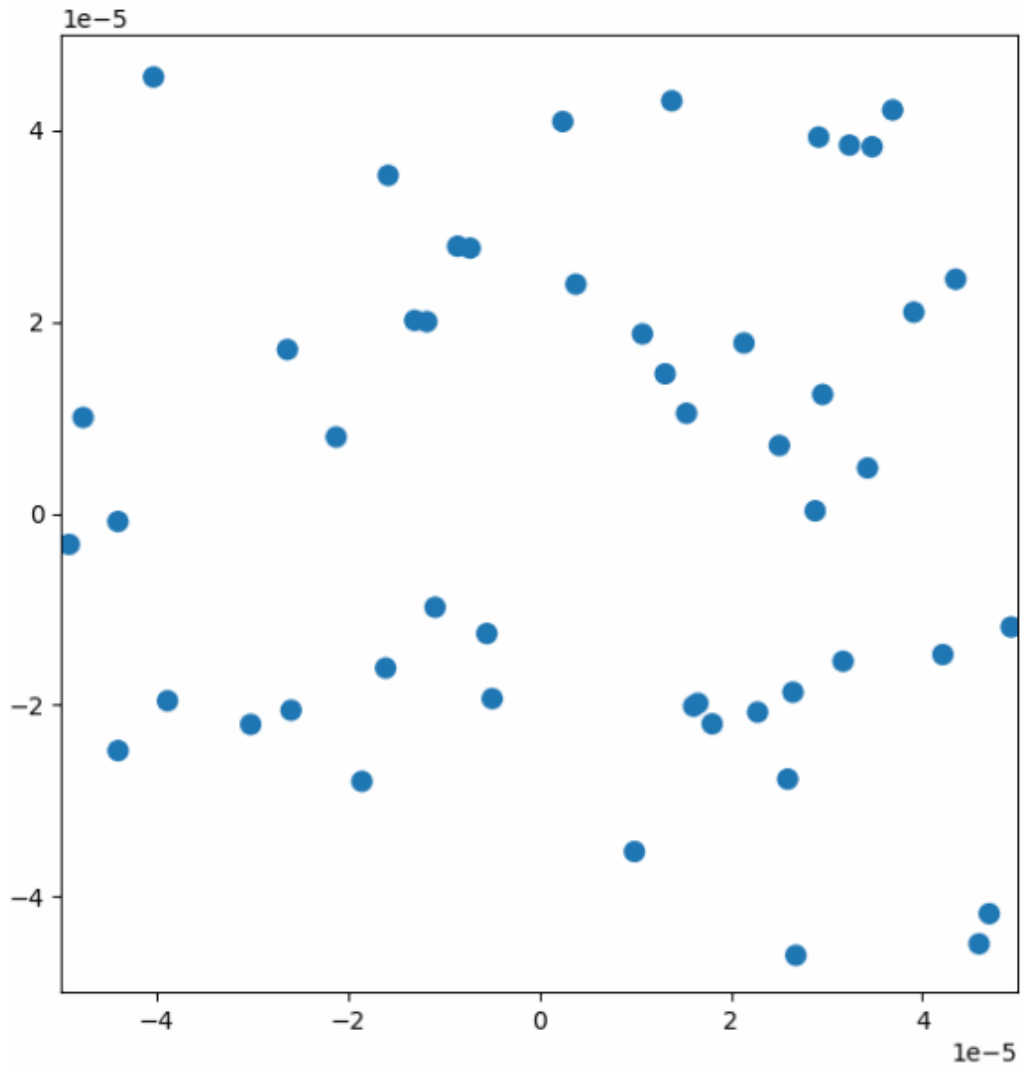


Figure 20: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50,  $\Delta t = 0.3$ ,  $v_0 = 20e - 6$ ,  $R = 1e - 6$ .

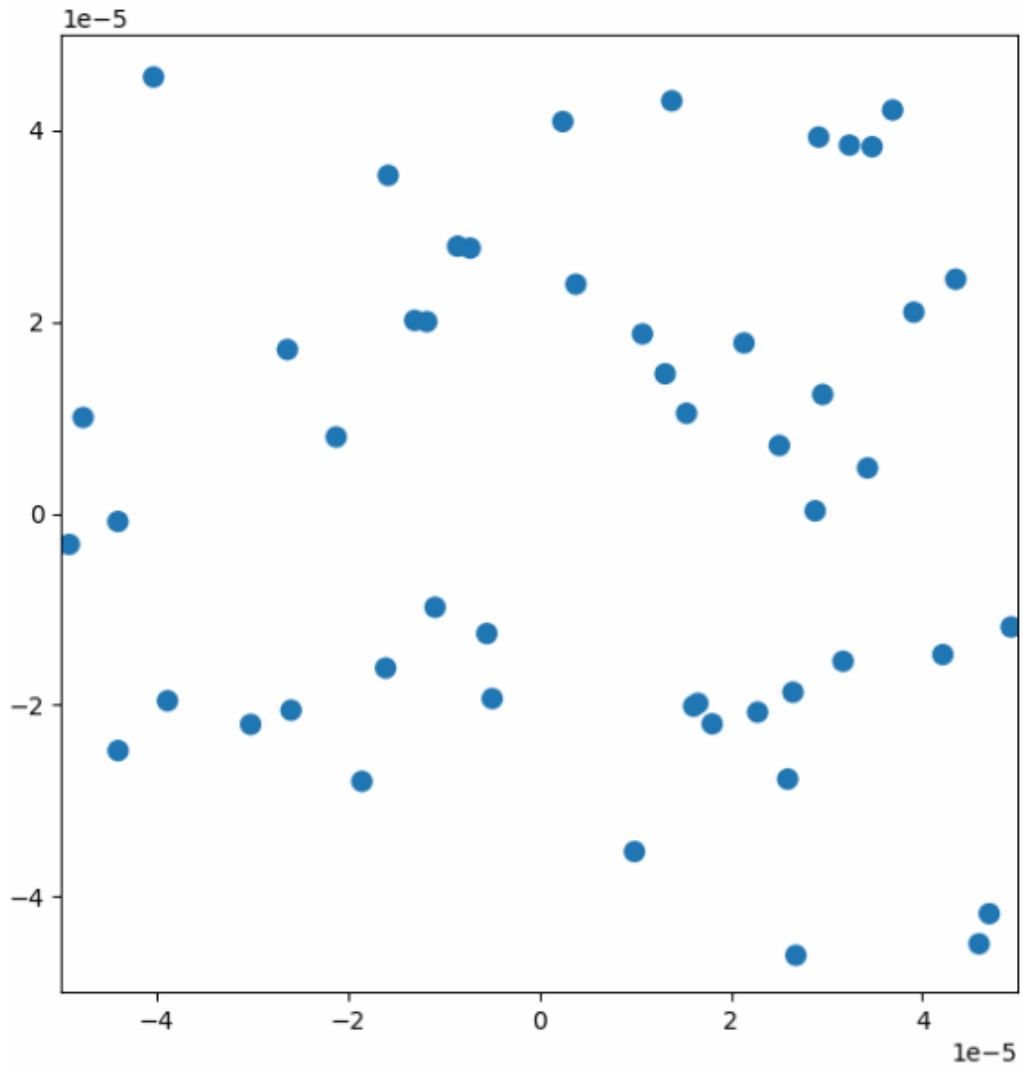


Figure 21: Both x and y-axis are length (m). No. time steps = 500, no. particles = 50,  $\Delta t = 0.3$ ,  $v_0 = 50 - 6$ ,  $R = 1e - 6$ .

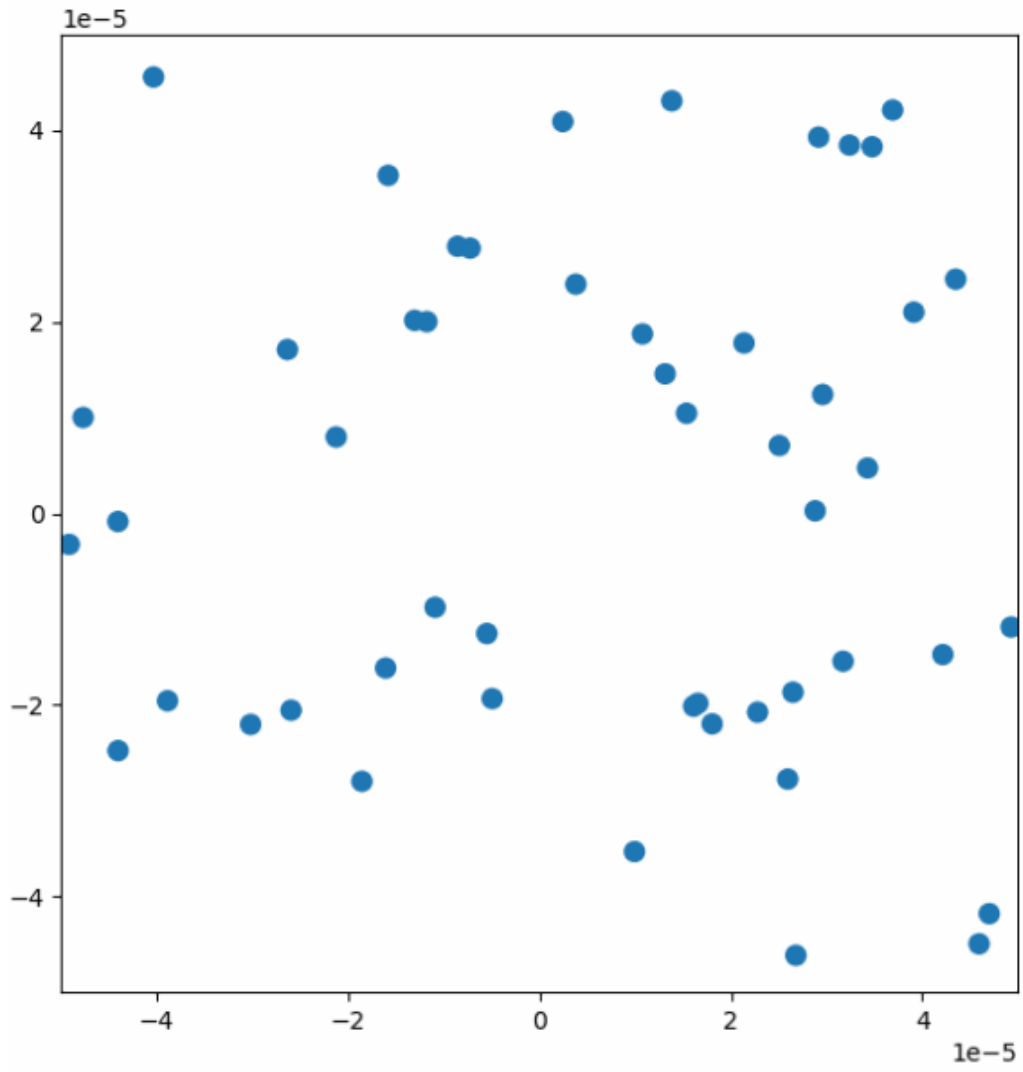


Figure 22: Both x and y-axis are length (m). No. time steps = 500, no. particles = 100,  $\Delta t = 0.3$ ,  $v_0 = 50 - 6$ ,  $R = 1e - 6$ .



```

1 # Exercise 9.1abcd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 T = 500
6 x = np.zeros((T+1))
7 y = x.copy()
8 phi = 0
9 dt = 0.01
10 # dt = tau*dt_step
11 # timesteps = int(tau/dt)*10s
12 # t = np.linspace(0, timesteps*dt_step, timesteps+1)
13
14 # Looping parameter
15 # Dt = 2*10**-12
16 # Dt = 5*10**-13 # OG
17 Dt = 8*10**-14
18
19 # Stretching parameter
20 # Dr = 0.05
21 # Dr = 0.5 # OG
22 Dr = 5
23
24 # Total length parameter
25 # v = 0
26 # v = 1*10**-6
27 # v = 2*10**-6
28 v = 3*10**-6 # OG
29
30 for t in range(T):
31
32     x[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
x[t]
33     y[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
y[t]
34     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
35
36 fig, ax = plt.subplots()
37
38 xmax = 2*10**-5
39 ymax = xmax
40
41 ax.plot(x, y, '-', linewidth=3, color='tab:red')
42 ax.set_xlim([-xmax,xmax])
43 ax.set_ylim([-ymax,ymax])
44 ax.set_box_aspect(1)
45 ax.set_ylabel('y (m)')
46 ax.set_xlabel('x (m)')
47
48 plt.savefig('exercise_9_1ab_v3_DrL.png', bbox_inches='tight')
49 plt.show()

```

```

1 # Exercise 9.2a
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sys
5
6 T = 10000
7 ensembles = 100
8 x = np.zeros((ensembles,T+1))
9 y = x.copy()
10 phi = 0
11 dt = 0.01
12 timesteps = int(T/dt)
13 t_iterations = np.linspace(0, T*dt, T+1)
14 # dt = tau*dt_step
15 # timesteps = int(tau/dt)*10s
16 # t = np.linspace(0, timesteps*dt_step, timesteps+1)
17
18 # Looping parameter
19 # Dt = 2*10**-12
20 # Dt = 5*10**-13 # OG
21 Dt = 8*10**-14
22
23 # Stretching parameter
24 # Dr = 0.05
25 # Dr = 0.5 # OG
26 Dr = 5
27
28 x_v0 = x.copy()
29 x_v1 = x.copy()
30 x_v2 = x.copy()
31 x_v3 = x.copy()
32
33 y_v0 = x.copy()
34 y_v1 = x.copy()
35 y_v2 = x.copy()
36 y_v3 = x.copy()
37
38 MSD_v0 = np.zeros((T+1))
39 MSD_v1 = MSD_v0.copy()
40 MSD_v2 = MSD_v0.copy()
41 MSD_v3 = MSD_v0.copy()
42
43 for ensemble in range(ensembles):
44
45     v = 0
46     for t in range(T):
47
48         x_v0[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*
49 (dt**0.5)) + x_v0[ensemble,t]
50         y_v0[ensemble,t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*
51 (dt**0.5)) + y_v0[ensemble,t]
52         phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
53
54     v = 1*10**-6
55     for t in range(T):
56
57         x_v1[ensemble,t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*
58 (dt**0.5)) + x_v1[ensemble,t]

```

```

56     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]
57     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
58
59     v = 2*10**-6
60     for t in range(T):
61
62         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]
63         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]
64         phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
65
66         v = 3*10**-6
67         for t in range(T):
68
69             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]
70             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]
71             phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
72
73     for t in range(T):
74         MSD_v0[t+1] = np.sum(x_v0[:,t+1]**2 + y_v0[:,t+1]**2) / ensembles
75         MSD_v1[t+1] = np.sum(x_v1[:,t+1]**2 + y_v1[:,t+1]**2) / ensembles
76         MSD_v2[t+1] = np.sum(x_v2[:,t+1]**2 + y_v2[:,t+1]**2) / ensembles
77         MSD_v3[t+1] = np.sum(x_v3[:,t+1]**2 + y_v3[:,t+1]**2) / ensembles
78
79     fig, ax = plt.subplots()
80
81     xmax = 2*10**-5
82     ymax = xmax
83
84     ax.plot(t_iterations, MSD_v0, '-', linewidth=1.5, color='tab:blue', label='v=0')
85     ax.plot(t_iterations, MSD_v1, '-', linewidth=1.5, color='tab:green', label='v=1*1e-
    6')
86     ax.plot(t_iterations, MSD_v2, '-', linewidth=1.5, color='tab:orange', label='v=2*1e-
    6')
87     ax.plot(t_iterations, MSD_v3, '-', linewidth=1.5, color='tab:red', label='v=3*1e-6')
88     # ax.set_xlim([-xmax,xmax])
89     # ax.set_ylim([-ymax,ymax])
90     ax.set_box_aspect(1)
91     ax.set_ylabel('MSD (m^2)')
92     ax.set_xlabel('t (s)')
93     ax.set_yscale('log')
94     ax.set_xscale('log')
95
96     plt.legend(loc="upper left")
97     plt.savefig('exercise_9_2a_DtL_DrL_ENSEMBLE.png', bbox_inches='tight')
98     plt.show()

```

```

1 # Exercise 9.2b
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sys
5
6 T = 10000
7 x = np.zeros((T+1))
8 y = x.copy()
9 phi = 0
10 dt = 0.01
11 timesteps = int(T/dt)
12 # dt = tau*dt_step
13 # timesteps = int(tau/dt)*10s
14 # t = np.linspace(0, timesteps*dt_step, timesteps+1)
15
16 # Looping parameter
17 Dt = 2*10**-12
18 # Dt = 5*10**-13 # OG
19 # Dt = 8*10**-14
20
21 # Stretching parameter
22 Dr = 0.05
23 # Dr = 0.5 # OG
24 # Dr = 5
25
26 x_v0 = x.copy()
27 x_v1 = x.copy()
28 x_v2 = x.copy()
29 x_v3 = x.copy()
30
31 y_v0 = x.copy()
32 y_v1 = x.copy()
33 y_v2 = x.copy()
34 y_v3 = x.copy()
35
36 # Simulation
37 v = 0
38 for t in range(T):
39
40     x_v0[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
41     x_v0[t]
42     y_v0[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
43     y_v0[t]
44     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
45
46 v = 1*10**-6
47 for t in range(T):
48
49     x_v1[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
50     x_v1[t]
51     y_v1[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
52     y_v1[t]
53     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
54
55 v = 2*10**-6
56 for t in range(T):
57
58     x_v2[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
59     x_v2[t]

```

```

55     y_v2[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
y_v2[t]
56     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
57
58 v = 3*10**-6
59 for t in range(T):
60
61     x_v3[t+1] = (v*np.cos(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
x_v3[t]
62     y_v3[t+1] = (v*np.sin(phi)*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*(dt**0.5)) +
y_v3[t]
63     phi = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi
64
65 # Time averaged MSD
66 t_range = int(T/2 + 1)
67 t_iterations = np.linspace(0, T*dt, t_range)
68
69 MSD_v0_time = np.zeros((t_range,t_range))
70 MSD_v1_time = MSD_v0_time.copy()
71 MSD_v2_time = MSD_v0_time.copy()
72 MSD_v3_time = MSD_v0_time.copy()
73
74 MSD_v0 = np.zeros((t_range))
75 MSD_v1 = MSD_v0.copy()
76 MSD_v2 = MSD_v0.copy()
77 MSD_v3 = MSD_v0.copy()
78
79 for i in range(len(x_v0)-t_range+1):
80
81     x_row_v0 = x_v0[i:i+t_range]
82     y_row_v0 = y_v0[i:i+t_range]
83     for j in range(len(y_row_v0)):
84         MSD_v0_time[i,j] = (x_row_v0[j] - x_row_v0[0])**2 + (y_row_v0[j] -
y_row_v0[0])**2
85
86     x_row_v1 = x_v1[i:i+t_range]
87     y_row_v1 = y_v1[i:i+t_range]
88     for j in range(len(y_row_v1)):
89         MSD_v1_time[i,j] = (x_row_v1[j] - x_row_v1[0])**2 + (y_row_v1[j] -
y_row_v1[0])**2
90
91     x_row_v2 = x_v2[i:i+t_range]
92     y_row_v2 = y_v2[i:i+t_range]
93     for j in range(len(y_row_v2)):
94         MSD_v2_time[i,j] = (x_row_v2[j] - x_row_v2[0])**2 + (y_row_v2[j] -
y_row_v2[0])**2
95
96     x_row_v3 = x_v3[i:i+t_range]
97     y_row_v3 = y_v3[i:i+t_range]
98     for j in range(len(y_row_v3)):
99         MSD_v3_time[i,j] = (x_row_v3[j] - x_row_v3[0])**2 + (y_row_v3[j] -
y_row_v3[0])**2
100
101 for i in range(t_range):
102     MSD_v0[i] = np.sum(MSD_v0_time[:,i])
103     MSD_v1[i] = np.sum(MSD_v1_time[:,i])
104     MSD_v2[i] = np.sum(MSD_v2_time[:,i])
105     MSD_v3[i] = np.sum(MSD_v3_time[:,i])
106
107 MSD_v0 = MSD_v0 / t_range

```

```
108 MSD_v1 = MSD_v1 / t_range
109 MSD_v2 = MSD_v2 / t_range
110 MSD_v3 = MSD_v3 / t_range
111
112 # t = 0
113 # for i in range(t_range):
114 #     for j in range(t_range):
115
116 #         MSD_v0[i] += (x_v0[t+j] - x_v0[t])**2 + (y_v0[t+j] - y_v0[t])**2
117 #         t += 1
118
119 # MSD_v0 = MSD_v0 / t_range
120
121 fig, ax = plt.subplots()
122
123 xmax = 2*10**-5
124 ymax = xmax
125
126 ax.plot(t_iterations, MSD_v0, '-', linewidth=1.5, color='tab:blue', label='v=0')
127 ax.plot(t_iterations, MSD_v1, '-', linewidth=1.5, color='tab:green', label='v=1*1e-6')
128 ax.plot(t_iterations, MSD_v2, '-', linewidth=1.5, color='tab:orange', label='v=2*1e-6')
129 ax.plot(t_iterations, MSD_v3, '-', linewidth=1.5, color='tab:red', label='v=3*1e-6')
130 # ax.set_xlim([-xmax,xmax])
131 # ax.set_ylim([-ymax,ymax])
132 ax.set_box_aspect(1)
133 ax.set_ylabel('MSD (m^2)')
134 ax.set_xlabel('t (s)')
135 ax.set_yscale('log')
136 ax.set_xscale('log')
137
138 plt.legend(loc="upper left")
139 plt.savefig('exercise_9_2b_DtS_DrS_TIME.png', bbox_inches='tight')
140 plt.show()
```

```

1 # Exercise 9.3a,b,c,d
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.animation import FuncAnimation
5 from matplotlib.animation import PillowWriter
6 import sys
7
8 T = 1000
9 dt = 0.000001
10 timesteps = int(T/dt)
11 t_iterations = np.linspace(0, T*dt, T+1)
12 Dt = 0.1*10**-6
13 # Dt = 0.1*10**-8
14 Dr = 1
15 # Dr = 0.01
16 v = 3*10**-6
17 R = 1*10**-6
18
19 N = 100 # No. particles
20 x = (np.random.rand(N,T+1)-0.5)*4*10**-5
21 y = (np.random.rand(N,T+1)-0.5)*4*10**-5
22 # d = np.zeros((N))
23 phi = (np.random.rand(N)-0.5)*2*np.pi
24
25 xmax = 2.5*10**-5
26 ymax = xmax
27 boundary_condition = xmax + R
28 marker_size = 20
29 fps_ani = 200
30 interval_ani = 0.01
31
32 fig, ax = plt.subplots(figsize=(7,7))
33
34 for t in range(T):
35
36     # Computing new positions
37     for n in range(N):
38
39         x[n,t+1] = (v*np.cos(phi[n])*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*
40 (dt**0.5)) + x[n,t]
41         y[n,t+1] = (v*np.sin(phi[n])*dt + ((2*Dt)**0.5)*np.random.normal(0,1)*
42 (dt**0.5)) + y[n,t]
43         phi[n] = ((2*Dr)**0.5)*np.random.normal(0,1)*(dt**0.5) + phi[n]
44
45     # Applying volume extraciton
46     for n in range(N):
47
48         distances = ((x[:,t+1]-x[n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
49         angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])
50         overlapp = distances < (2*R)
51         overlapp[n] = False
52
53         for i in np.where(overlapp)[0]:
54             x[n,t+1] = x[n,t+1] + (distances[i] - 2*R)*np.cos(angles[i])/2
55             y[n,t+1] = y[n,t+1] + (distances[i] - 2*R)*np.sin(angles[i])/2
56             x[i,t+1] = x[i,t+1] - (distances[i] - 2*R)*np.cos(angles[i])/2
57             y[i,t+1] = y[i,t+1] - (distances[i] - 2*R)*np.sin(angles[i])/2
58
59     # Boundary condition

```

```
58     if x[n,t+1] > boundary_condition:
59         x[n,t+1] = -boundary_condition
60
61     elif x[n,t+1] < -boundary_condition:
62         x[n,t+1] = boundary_condition
63
64     if y[n,t+1] > boundary_condition:
65         y[n,t+1] = -boundary_condition
66
67     elif y[n,t+1] < -boundary_condition:
68         y[n,t+1] = boundary_condition
69
70 def animate(i):
71     ax.clear()
72     for n in range(N):
73         x_point = x[n,i]
74         y_point = y[n,i]
75         # ax.plot(x_point, y_point, color='tab:blue', marker='.',
76 markersize=marker_size)
77         ax.add_patch(plt.Circle([x_point, y_point], radius=R, color='tab:blue'))
78     ax.set_xlim([-xmax, xmax])
79     ax.set_ylim([-ymax, ymax])
80 ax.set_ylabel('y (m)')
81 ax.set_xlabel('x (m)')
82 ax.set_box_aspect(1)
83 ani = FuncAnimation(fig, animate, frames=len(x[0,:]), interval=interval_ani)
84 ani.save('exercise_9_3c_VE_dt0.000001_T1000_Dt0.1e-6_Dr1.gif',
85         writer=PillowWriter(fps=fps_ani))
86 plt.show()
```



```

1 # Exercise 9.4, 9.5
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.animation import FuncAnimation
5 from matplotlib.animation import PillowWriter
6 import sys
7
8 T = 500
9 dt = 0.3
10 timesteps = int(T/dt)
11 t_iterations = np.linspace(0, T*dt, T+1)
12 R = 1*10**-6
13 gamma = 6*np.pi*0.001*R
14 kB = 1.380649*10**-23
15 Dt = kB*300/gamma
16 Dt2 = kB*300/(8*np.pi*0.001*R**3)
17 # Dt = 0.1*10**-6
18 # Dt = 0.1*10**-8
19 Dr = 1
20 # Dr = 0.01
21 v = 3*10**-6
22
23 N = 50 # No. particles
24 x = (np.random.rand(N,T+1)-0.5)*8*10**-5
25 y = (np.random.rand(N,T+1)-0.5)*8*10**-5
26 phi = (np.random.rand(N)-0.5)*2*np.pi
27
28 vpx = np.zeros(N)
29 vpy = np.zeros(N)
30 v0 = 50*10**-6
31
32 xmax = 50*10**-6
33 ymax = xmax
34 boundary_condition = xmax + R
35 marker_size = 20
36 fps_ani = 200
37 interval_ani = 0.01
38
39 fig, ax = plt.subplots(figsize=(7,7))
40
41 for t in range(T):
42
43     # Computing new positions
44     for n in range(N):
45
46         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]
47         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]
48         phi[n] = ((2*Dr*Dt2*dt)**0.5)*np.random.normal(0,1) + phi[n]
49         # phi[n] = ((2*Dr*dt)**0.5)*np.random.normal(0,1) + phi[n]
50
51     vpx = np.zeros(N)
52     vpy = np.zeros(N)
53
54     for n in range(N):
55
56         distances = ((x[:,t+1]-x[n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
57         angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])

```

```

58
59     interact = distances < 5*R
60     interact[n] = False
61
62     for i in np.where(interact)[0]:
63         vpx[n] = vpx[n] + (v0*R**2 / distances[i]**2) * np.cos(angles[i]) *
distances[i]
64         vpy[n] = vpx[n] + (v0*R**2 / distances[i]**2) * np.sin(angles[i]) *
distances[i]
65
66     # Applying volume extraciton
67     # for i in range(3):
68     for n in range(N):
69
70         distances = ((x[:,t+1]-x[-n,t+1])**2 + (y[:,t+1]-y[n,t+1])**2)**0.5
71         angles = np.arctan2(y[:,t+1]-y[n,t+1], x[:,t+1]-x[n,t+1])
72         overlap = distances < (2*R)
73         overlap[n] = False
74
75         for i in np.where(overlap)[0]:
76             x[n,t+1] = x[n,t+1] + (distances[i] - 2*R)*np.cos(angles[i])/2
77             y[n,t+1] = y[n,t+1] + (distances[i] - 2*R)*np.sin(angles[i])/2
78             x[i,t+1] = x[i,t+1] - (distances[i] - 2*R)*np.cos(angles[i])/2
79             y[i,t+1] = y[i,t+1] - (distances[i] - 2*R)*np.sin(angles[i])/2
80
81         # Boundary condition
82         if x[n,t+1] > boundary_condition:
83             x[n,t+1] = -boundary_condition
84
85         elif x[n,t+1] < -boundary_condition:
86             x[n,t+1] = boundary_condition
87
88         if y[n,t+1] > boundary_condition:
89             y[n,t+1] = -boundary_condition
90
91         elif y[n,t+1] < -boundary_condition:
92             y[n,t+1] = boundary_condition
93
94 def animate(i):
95     ax.clear()
96     for n in range(N):
97         x_point = x[n,i]
98         y_point = y[n,i]
99         ax.add_patch(plt.Circle([x_point, y_point], radius=R, color='tab:blue'))
100     ax.set_xlim([-xmax, xmax])
101     ax.set_ylim([-ymax, ymax])
102
103 ax.set_ylabel('y (m)')
104 ax.set_xlabel('x (m)')
105 ax.set_box_aspect(1)
106 ani = FuncAnimation(fig, animate, frames=len(x[0,:]), interval=interval_ani)
107 ani.save('exercise_9_5_v0_50_test.gif', writer=PillowWriter(fps=fps_ani))
108 plt.show()

```