

```

1 import random
2 from math import pi, sin, cos, e
3 from matplotlib.ticker import PercentFormatter
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 class Simulation(object):
8     def __init__(self, N, L, h, t, v=1, m=10, T=2,
9         Berendsen=1, d_intermolecular=1.5, d_wall=1):
10         self.N = N
11         self.L = L
12         self.h = h
13         self.t = t
14         self.v = v
15         self.m = m
16         self.T = T
17         self.Berendsen=Berendsen
18         self.d_intermolecular = d_intermolecular
19         self.d_wall = d_wall
20         self.pop = []
21
22         self.K0 = (2 * self.N /2) * self.T
23         self.spawn()
24         self.runEvent()
25
26     def spawn(self):
27         for i in range(self.N):
28             new_molecule = Molecule(self.L,self.v)
29             while not self.distanceCheck(new_molecule):
30                 new_molecule = Molecule(self.L, self.v)
31             self.pop.append(new_molecule)
32             new_molecule.l_position.append((new_molecule.x
33 , new_molecule.y))
34             new_molecule.l_velocity.append((new_molecule.
35 v_x, new_molecule.v_y))
36
37     def distanceCheck(self,a):
38         if len(self.pop)<1:
39             return True
40         else:
41             for i in self.pop:
42                 d_m = ((a.x-i.x)**2) + ((a.y-i.y)**2)
43                 if d_m <= self.d_intermolecular**2:
44                     return False
45                 if (a.x < self.d_wall) or (a.y < self.d_wall)
46 or (a.x > self.L-self.d_wall) or (a.y > self.L-self.d_wall
47 ):
48                     return False
49             return True

```

```

46     def plotTrajectory(self):
47
48         plt.style.use('ggplot')
49         fig = plt.figure()
50         ax1=fig.add_subplot(111)
51
52         num=0
53         for i in self.pop:
54             num+=1
55             x = []
56             y = []
57
58             x1=i.l_position[0][0]
59             y1=i.l_position[0][1]
60             ax1.scatter(x1, y1, color='r')
61             for j in i.l_position:
62                 x.append(j[0])
63                 y.append(j[1])
64             ax1.scatter(j[0], j[1], color='b')
65             ax1.plot(x,y,'--')
66
67             ax1.set_xlabel("x")
68             ax1.set_ylabel("y")
69             plt.title('Leonard Jones Simulation, t = ' + str(
self.t-0.1))
70             plt.show()
71
72     def plotInitialState(self,withVelocity=False):
73         x = []
74         y = []
75         plt.style.use('ggplot')
76         fig = plt.figure()
77         ax1 = fig.add_subplot(111)
78
79         for i in self.pop:
80             x.append(i.l_position[0][0])
81             y.append(i.l_position[0][1])
82             ax1.annotate("(" +str(i.l_position[0][0]))[:4]+
", "+str(i.l_position[0][1]))[:4]+")", (i.l_position[0][0],i.
l_position[0][1]))
83             ax1.scatter(x,y,color='r')
84
85         if withVelocity:
86             for i in self.pop:
87                 ax1.arrow(i.l_position[0][0],i.l_position[0
][1],i.l_velocity[0][0],i.l_velocity[0][1],color='k',
head_width=0.15)
88             ax1.set_xbound(0,self.L)
89             ax1.set_ybound(0,self.L)
90             ax1.set_xlabel("x")

```

```

91         ax1.set_ylabel("y")
92         plt.title('Initial State of Atoms')
93         plt.legend()
94         plt.show()
95
96     def runEvent(self):
97         for i in self.pop:
98             i.l_position.append((i.x+self.h*i.v_x,i.y+self
99             .h*i.v_y))
100             i.x=i.l_position[-1][0]
101             i.y=i.l_position[-1][1]
102
103         time = self.h
104         while time <= self.t:
105             time += self.h
106             F_total = []
107
108             K_system=0
109             for i in self.pop:
110                 K_system += 0.5 * self.m * ((i.v_x**2)+(i.
111                 v_y**2))
112
113             for i in self.pop:
114                 F_wall_x = ((self.m)/(i.x**(self.m+1)))+((
115                 self.m)/((i.x-self.L)**(self.m+1)))
116                 F_wall_y = ((self.m)/(i.y**(self.m+1)))+((
117                 self.m)/((i.y-self.L)**(self.m+1)))
118
119             F_lj_x = 0
120             F_lj_y = 0
121             for j in self.pop:
122                 if i!=j:
123                     r2 = ((i.x-j.x)**2) + ((i.y-j.y)**
124                     2)
125                     r2_factor = 24 * (2*(1/(r2**7
126                     )) - (1/(r2**4)))
127                     F_lj_x += (i.x-j.x)*r2_factor
128                     F_lj_y += (i.y-j.y)*r2_factor
129
130             F_thermostat_x = self.m * self.Berendsen
131             *((self.K0/K_system)-1)*i.v_x
132             F_thermostat_y = self.m * self.Berendsen
133             *((self.K0/K_system)-1)*i.v_y
134
135             F_total_x = F_wall_x + F_lj_x +
136             F_thermostat_x
137             F_total_y = F_wall_y + F_lj_y +
138             F_thermostat_y
139             F_total_vec = (F_total_x,F_total_y)
140             F_total.append(F_total_vec)

```

```

131
132         for i in range(len(self.pop)):
133             new_x = 2*self.pop[i].l_position[-1][0] -
self.pop[i].l_position[-2][0] + F_total[i][0]*(self.h**2)
134             new_y = 2*self.pop[i].l_position[-1][1] -
self.pop[i].l_position[-2][1] + F_total[i][1]*(self.h**2)
135             self.pop[i].l_position.append((new_x,new_y
))
136
137             new_vx = (new_x - self.pop[i].x)/self.h
138             new_vy = (new_y - self.pop[i].y)/self.h
139             self.pop[i].l_velocity.append((new_vx,
new_vy))
140
141             self.pop[i].x = new_x
142             self.pop[i].y = new_y
143             self.pop[i].v_x = new_vx
144             self.pop[i].v_y = new_vy
145
146     def MaxwellSpeedDistribution(self):
147
148         v_3=[]
149         v_4=[]
150         v_5=[]
151         v_6=[]
152
153         for i in self.pop:
154             v_3.append(i.l_velocity[100:(10**3):10])
155             v_4.append(i.l_velocity[100:(10**4):10])
156             v_5.append(i.l_velocity[100:(10**5):10])
157             v_6.append(i.l_velocity[100::100])
158
159         for i in [v_3,v_4,v_5,v_6]:
160             for j in i:
161                 for k in range(len(j)):
162                     j[k]=(j[k][0]**2)+(j[k][1]**2)
163         v_3=np.array(v_3)
164         v_4=np.array(v_4)
165         v_5=np.array(v_5)
166         v_6=np.array(v_6)
167
168         v_3_avg=[]
169         v_3_rms=[]
170
171         for i in range(int(np.size(v_3)/self.N)):
172             v_3_avg.append(np.mean(v_3[:,i]))
173             v_3_rms.append(np.sqrt(np.mean(v_3[:,i]**2)))
174
175         # v_3_avgS=np.std(v_3_avg)
176         # v_3_rmsS=np.std(v_3_rms)

```

```

177     v_3_avg=np.mean(v_3_avg)
178     v_3_rms_m=np.mean(v_3_rms)
179
180     v_4_avg=[]
181     v_4_rms=[]
182
183     for i in range(int(np.size(v_4)/self.N)):
184         v_4_avg.append(np.mean(v_4[:,i]))
185         v_4_rms.append(np.sqrt(np.mean(v_4[:,i]**2)))
186
187     # v_4_avgS=np.std(v_4_avg)
188     # v_4_rmsS=np.std(v_4_rms)
189     v_4_avg=np.mean(v_4_avg)
190     v_4_rms_m=np.mean(v_4_rms)
191
192     v_5_avg=[]
193     v_5_rms=[]
194
195     for i in range(int(np.size(v_5)/self.N)):
196         v_5_avg.append(np.mean(v_5[:,i]))
197         v_5_rms.append(np.sqrt(np.mean(v_5[:,i]**2)))
198
199     # v_5_avgS=np.std(v_5_avg)
200     # v_5_rmsS=np.std(v_5_rms)
201     v_5_avg=np.mean(v_5_avg)
202     v_5_rms_m=np.mean(v_5_rms)
203
204     v_6_avg=[]
205     v_6_rms=[]
206
207     for i in range(int(np.size(v_6)/self.N)):
208         v_6_avg.append(np.mean(v_6[:,i]))
209         v_6_rms.append(np.sqrt(np.mean(v_6[:,i]**2)))
210
211     # v_6_avgS=np.std(v_6_avg)
212     # v_6_rmsS=np.std(v_6_rms)
213     v_6_avg=np.mean(v_6_avg)
214     v_6_rms_m=np.mean(v_6_rms)
215
216     v=np.linspace(0,5,100)
217     g=((1/(2*pi*2))**1.5)*4*pi*(v**2))*(e**((-1)*(v**
218 2)/(2*2)))
219
220     plt.style.use('ggplot')
221     fig = plt.figure()
222     ax1 = fig.add_subplot(221)
223     ax1.hist(v_3_rms,weights=np.ones(len(v_3_rms))/len
(v_3_rms),label=r"M = $10^3$ : " + r"$\bar{v}$ = " + str(
v_3_avg)[:6] + r", $v_{rms}$ = " + str(v_3_rms_m)[:6])
223     ax1.plot(v,g,label="Maxwell")

```

```

224
225     ax2 = fig.add_subplot(222)
226     ax2.hist(v_4_rms, weights=np.ones(len(v_4_rms))/len
(v_4_rms), label=r" $M = 10^4$  : " + r"$\bar{v}$ = " + str(
v_4_avg)[:6] + r",  $\sigma_{rms}$  = " + str(v_4_rms_m)[:6])
227     ax2.plot(v, g, label="Maxwell")
228
229     ax3 = fig.add_subplot(223)
230     ax3.hist(v_5_rms, weights=np.ones(len(v_5_rms))/len
(v_5_rms), label=r" $M = 10^5$  : " + r"$\bar{v}$ = " + str(
v_5_avg)[:6] + r",  $\sigma_{rms}$  = " + str(v_5_rms_m)[:6])
231     ax3.plot(v, g, label="Maxwell")
232
233     ax4 = fig.add_subplot(224)
234     ax4.hist(v_6_rms, weights=np.ones(len(v_6_rms)) /
len(v_6_rms), label=r" $M = 10^6$  : " + r"$\bar{v}$ = " + str(
v_6_avg)[:6] + r",  $\sigma_{rms}$  = " + str(v_6_rms_m)[:6])
235     ax4.plot(v, g, label="Maxwell")
236
237     ax1.yaxis.set_major_formatter(PercentFormatter(1))
238     ax2.yaxis.set_major_formatter(PercentFormatter(1))
239     ax3.yaxis.set_major_formatter(PercentFormatter(1))
240     ax4.yaxis.set_major_formatter(PercentFormatter(1))
241     ax1.set_xlabel("Speed")
242     ax1.set_ylabel("Density")
243     ax2.set_xlabel("Speed")
244     ax2.set_ylabel("Density")
245     ax3.set_xlabel("Speed")
246     ax3.set_ylabel("Density")
247     ax4.set_xlabel("Speed")
248     ax4.set_ylabel("Density")
249     fig.suptitle('Maxwell Speed Distribution
Simulation')
250     ax1.legend()
251     ax2.legend()
252     ax3.legend()
253     ax4.legend()
254     plt.show()
255
256 class Molecule(object):
257     def __init__(self, L, v):
258         self.L = L
259         self.v = v
260         self.x = random.uniform(0,1)*self.L
261         self.y = random.uniform(0,1)*self.L
262         theta=random.uniform(0,2*pi)
263         self.v_x = sin(theta)*self.v
264         self.v_y = cos(theta)*self.v
265         self.l_position = []
266         self.l_velocity = []

```

```

267
268
269 # test=Simulation(N=10,L=20,h=0.001,t=1000.1)
270
271 # Problem 3.3
272 # test.plotInitialState() #a
273 # test.plotInitialState(True) #b
274
275 # Problem 3.4
276 # m = 10
277 # h = 0.001
278 # v0 = 1
279 # L = 20
280 # x = [L/2]
281 # v = [v0]
282 # x.append(10+0.001*v0)
283 #
284 # for i in range(10**5):
285 #     F_wall_x = (m / (x[-1] ** (m + 1))) + (m / ((x[-1]
286 #         ] - L) ** (m + 1)))
287 #     x_new=2*x[-1]-x[-2]+F_wall_x*(h**2)
288 #     v.append((x_new-x[-1])/h)
289 #     x.append(x_new)
290 # t=[i*h for i in range(0,10**5+2)]
291 # plt.style.use('ggplot')
292 # fig = plt.figure()
293 # ax1 = fig.add_subplot(111)
294 # ax1.plot(t,x,label="Trajectory x(t)")
295 # t.pop()
296 # ax1.plot(t,0.5*m*(np.array(v)**2),label="Energy E(t)")
297 # ax1.set_xbound(-1, h*(10**5))
298 # ax1.set_ybound(-1, L)
299 # ax1.legend()
300 # plt.show()
301
302 # Problem 3.7
303
304 # test=Simulation(N=10,L=20,h=0.001,t=2.1)
305 # test=Simulation(N=10,L=20,h=0.001,t=0.2)
306 # test.plotTrajectory()
307
308 # Problem 3.10
309 # test.MaxwellSpeedDistribution()
310
311
312

```