

ZJU Computational Physics: Homework #5

Due on Monday, December 23, 2024

Github: <https://github.com/NAKONAKO4/ZJU-computational-physics-NAKO>

NAKO

Problem 1

MD: Approach to Equilibrium

a. 可以看到能量和动量都是守恒的，且动量基本为 0 (10^{-14} 数量级)；同时计算得到的压强与理想气体压强差值相对误差较小。)

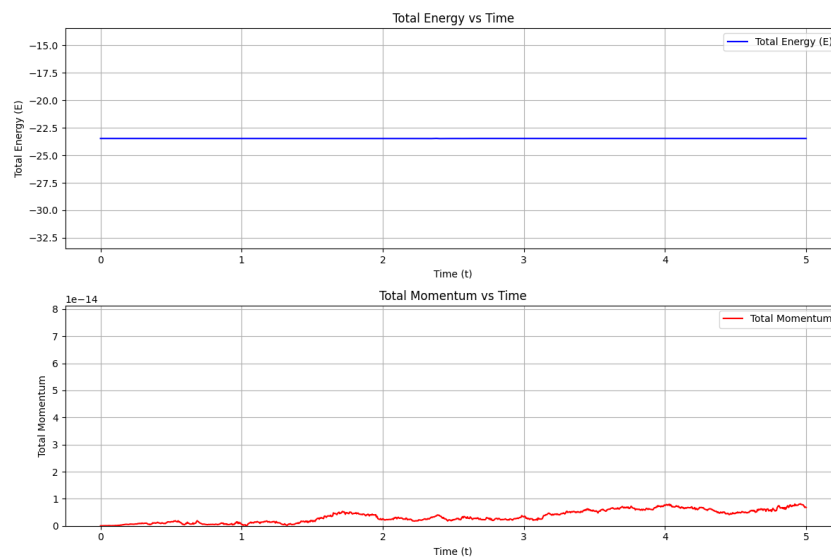


图 1: 能量和动量随时间变化

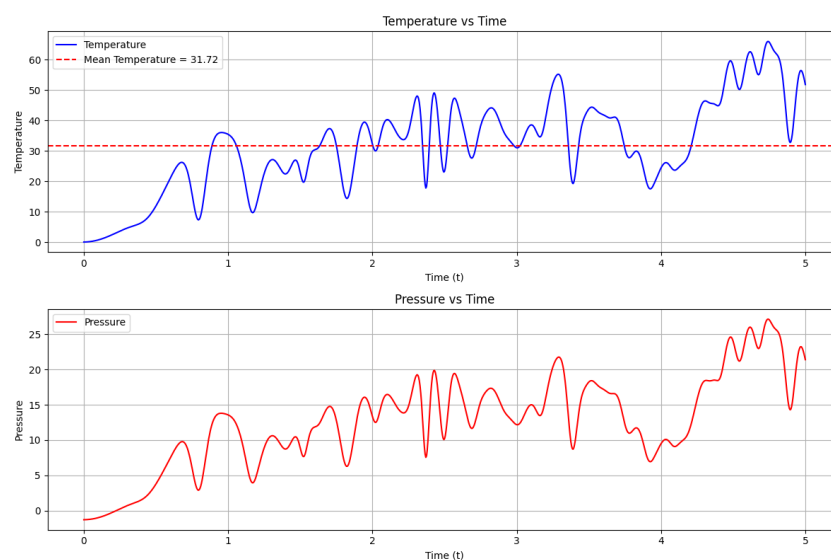


图 2: 温度和压强随时间变化

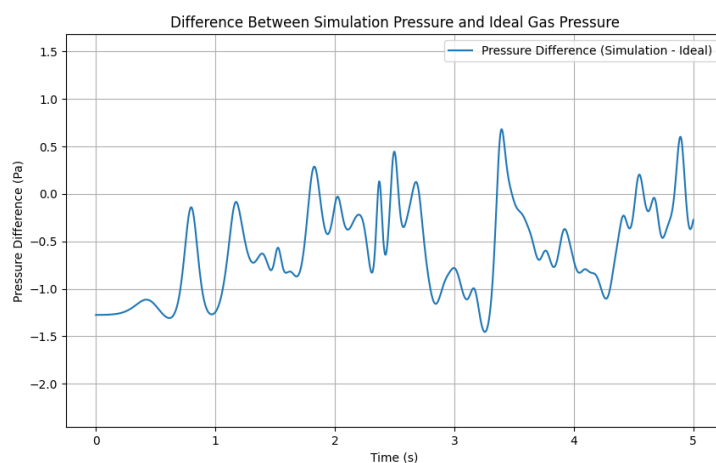


图 3: 压强计算值与理想气体公式计算得到压强差值

b. 可以看到能量和动量都是守恒的，且动量基本为 0 (10^{-14} 数量级))

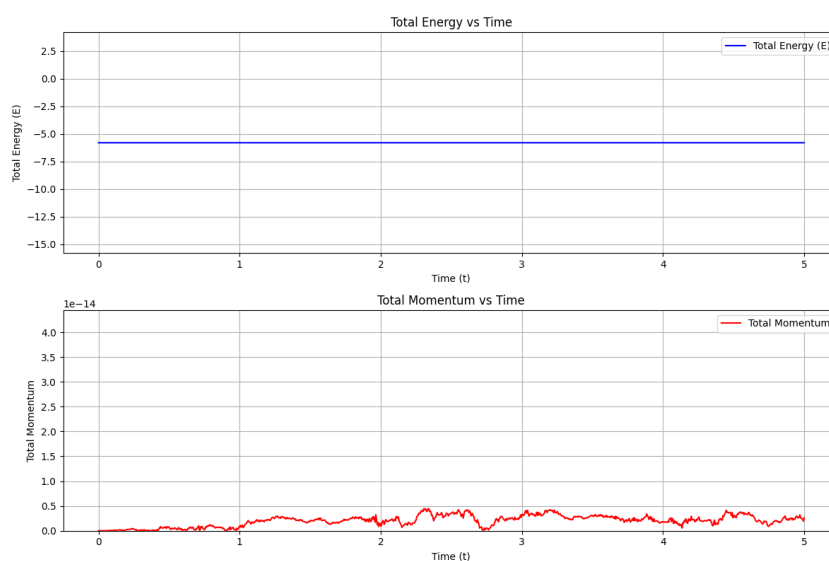


图 4: 能量和动量随时间变化

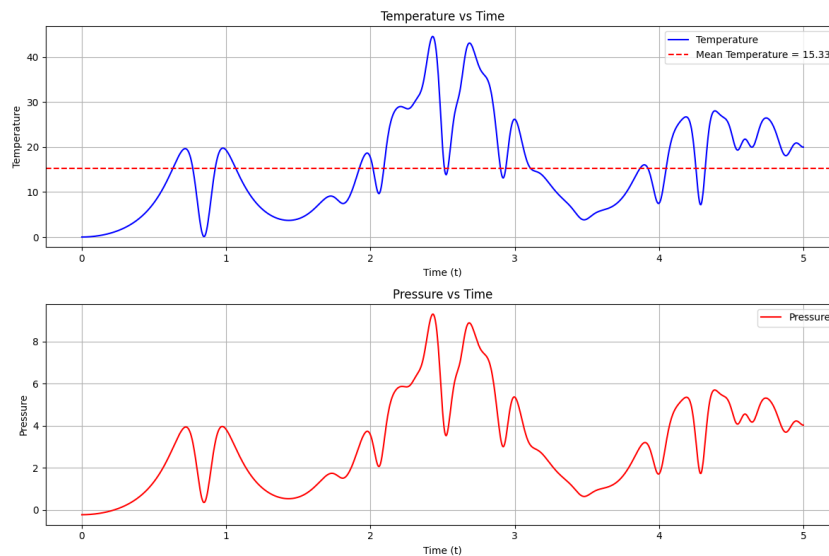
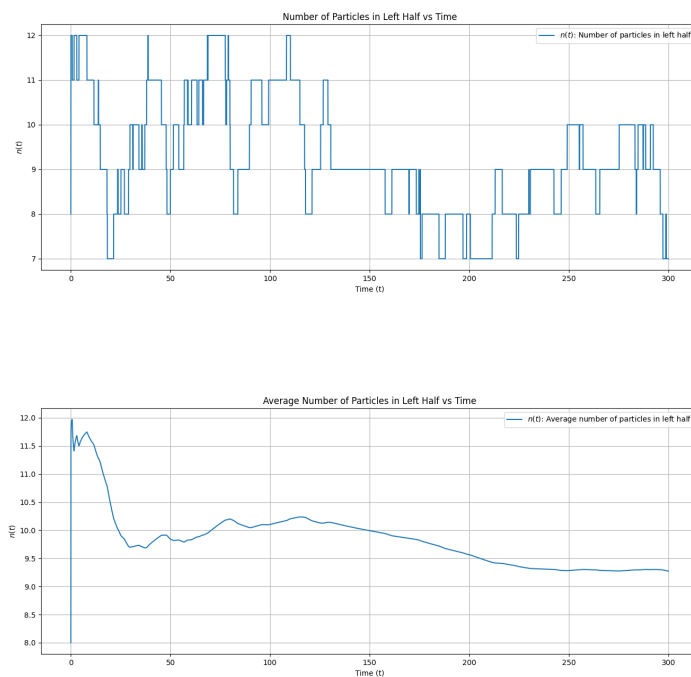


图 5: 温度和压强随时间变化

c. 可以看到接近平衡态时，左边粒子数均值会趋近于总粒子数的一半 $N/2 = 9$)



d. 可以看到能量和动量都是守恒的，且动量基本为 0 (10^{-14} 数量级)；同时计算得到的压强与理想气体压强差值相对误差较小。)

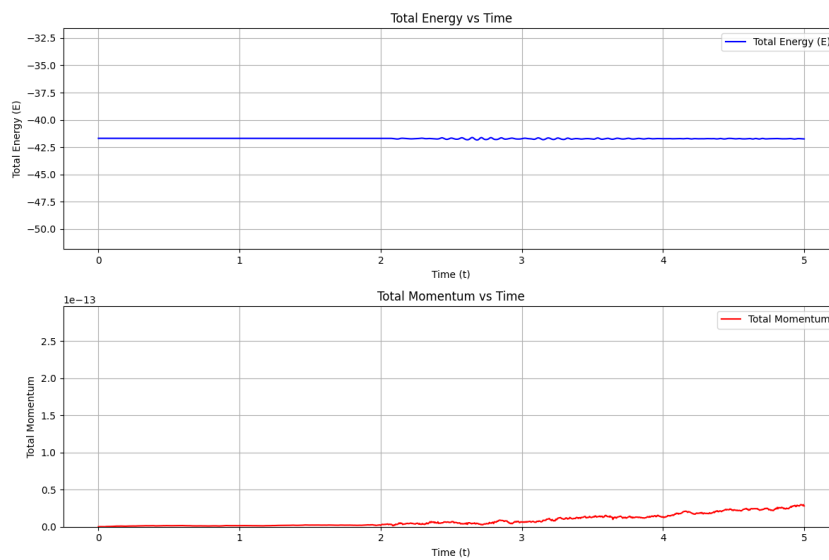


图 6: 能量和动量随时间变化

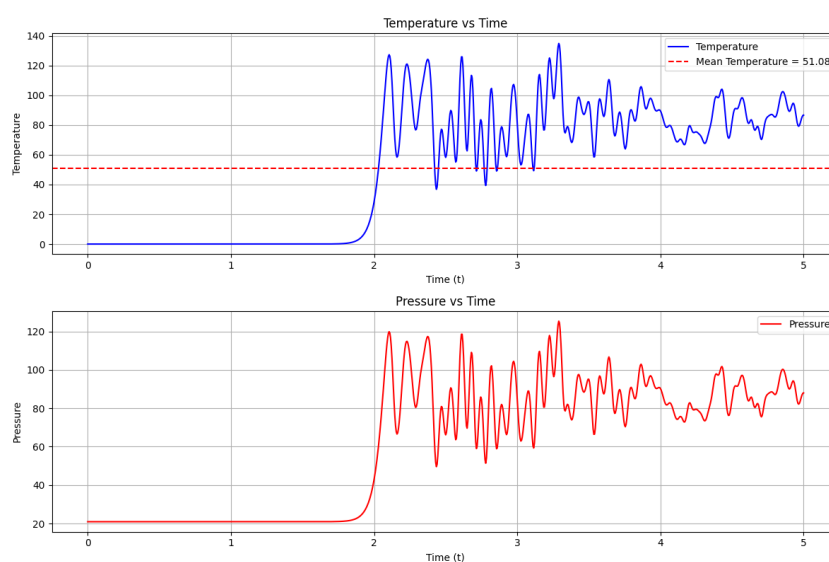


图 7: 温度和压强随时间变化

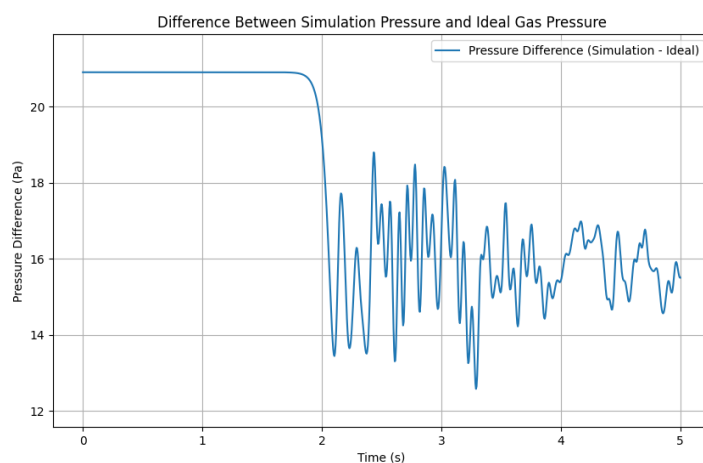


图 8: 压强计算值与理想气体公式计算得到压强差值

e. 可以看到能量和动量都是守恒的，且动量基本为 0 (10^{-14} 数量级)；同时计算得到的压强与理想气体压强差值相对误差较小。)

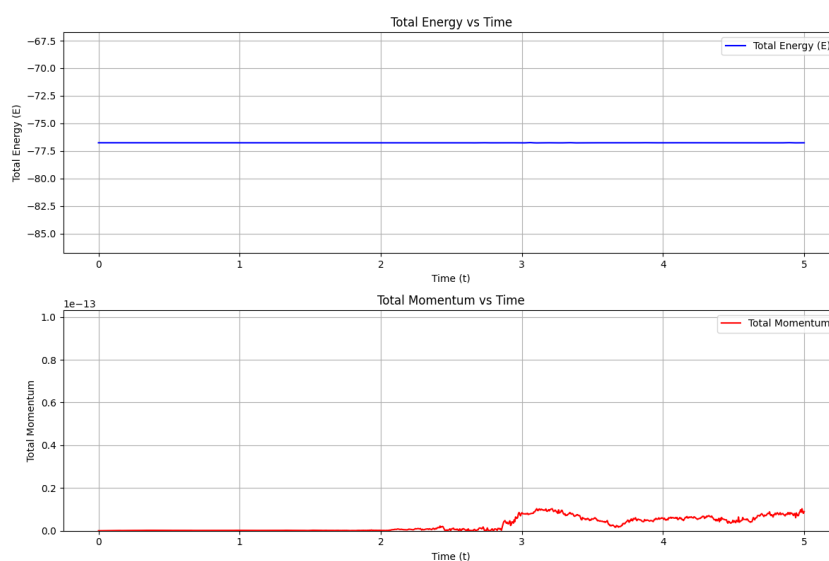


图 9: 能量和动量随时间变化

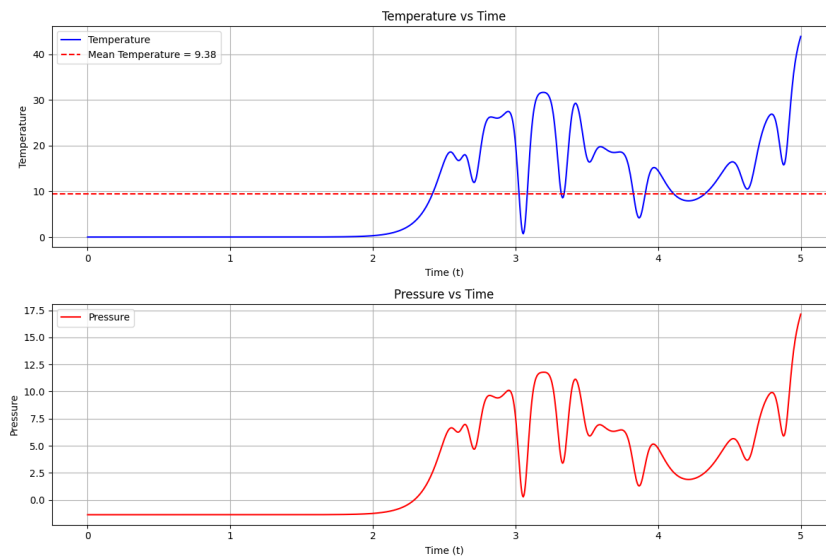


图 10: 温度和压强随时间变化

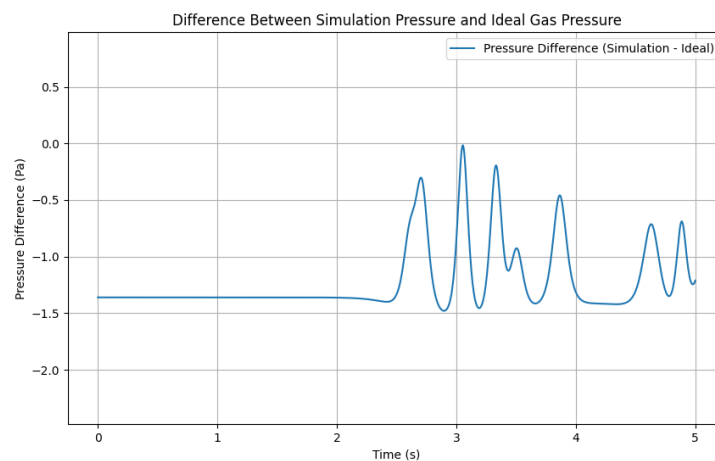


图 11: 压强计算值与理想气体公式计算得到压强差值

abde 题代码如下，对于不同的初始条件只需要更改代码中的初始条件设置即可。c 题代码见本代码的后一部分

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sigma = 0.1
5 epsilon = 1
6 mass = 6.69e-26
7 tau = sigma * np.sqrt(mass / epsilon)
8 k_B = 1.38e-23
```

```

9
10 dt = 0.005
11 dt2 = dt**2
12 N = 18
13 Lx, Ly = 6.0, 6.0
14 k_B = 1.38e-23
15 temperature = 100
16 m = 1
17 area = Lx * Ly
18
19 def create_lattice(N, Lx, Ly):
20     positions = np.zeros((N, 2))
21     num_rows = int(np.sqrt(N))
22     num_cols = int(np.ceil(N / num_rows))
23     x_spacing = Lx / num_cols
24     y_spacing = Ly / num_rows
25
26     index = 0
27     for row in range(num_rows):
28         for col in range(num_cols):
29             if index >= N:
30                 break
31             x_pos = (col + 0.5) * x_spacing
32             y_pos = (row + 0.5) * y_spacing
33             positions[index] = [x_pos, y_pos]
34             index += 1
35     return positions
36
37 def init_velocity(N, temperature, mass):
38     stddev = np.sqrt(k_B * temperature / mass)
39     velocities = np.random.normal(0, stddev, (N, 2))
40     velocities -= np.mean(velocities, axis=0)
41     return velocities
42
43 positions = create_lattice(N, Lx, Ly)
44 velocities = init_velocity(N, temperature, m)
45
46 x, y = positions[:, 0], positions[:, 1]
47 vx, vy = velocities[:, 0], velocities[:, 1]
48 print(vx)
49 ax = np.zeros(N)
50 ay = np.zeros(N)
51
52 plt.figure(figsize=(6, 6))
53 plt.scatter(x, y, c='blue', label='Particles', s=100)
54 plt.xlim(0, Lx)
55 plt.ylim(0, Ly)

```



```

56 plt.gca().set_aspect('equal', adjustable='box')
57 plt.xlabel('X_Position(Å)')
58 plt.ylabel('Y_Position(Å)')
59 plt.title('Particle_Positions_in_6x6_Box')
60 plt.grid(color='gray', linestyle='--', linewidth=0.5)
61 plt.legend()
62 plt.show()
63
64
65 def check_momentum(vx, vy):
66     vx -= np.sum(vx) / N
67     vy -= np.sum(vy) / N
68     return vx, vy
69
70 def pbc(pos, L):
71     return (pos + L) % L
72
73 def separation(ds, L):
74     return ds - L * np.round(ds / L)
75
76 def force(dx, dy):
77     r2 = dx**2 + dy**2
78     if r2 == 0:
79         return 0, 0, 0
80     rm2 = 1.0 / r2
81     rm6 = rm2**3
82     f_over_r = 24 * rm6 * (2 * rm6 - 1) * rm2
83     fx = f_over_r * dx
84     fy = f_over_r * dy
85     pot = 4.0 * (rm6**2 - rm6)
86     return fx, fy, pot
87
88 def accel(x, y, ax, ay):
89     ax.fill(0)
90     ay.fill(0)
91     pe = 0.0
92     virial = 0.0
93     for i in range(N - 1):
94         for j in range(i + 1, N):
95             dx = separation(x[i] - x[j], Lx)
96             dy = separation(y[i] - y[j], Ly)
97             fx, fy, pot = force(dx, dy)
98             ax[i] += fx
99             ay[i] += fy
100            ax[j] -= fx
101            ay[j] -= fy
102            pe += pot

```

```

103         virial += dx* fx + dy* fy
104     return pe, virial
105
106 def verlet(x, y, vx, vy, ax, ay):
107     x += vx * dt + 0.5 * ax * dt2
108     y += vy * dt + 0.5 * ay * dt2
109     x = pbc(x, Lx)
110     y = pbc(y, Ly)
111     vx += 0.5 * ax * dt
112     vy += 0.5 * ay * dt
113     pe, virial = accel(x, y, ax, ay)
114     vx += 0.5 * ax * dt
115     vy += 0.5 * ay * dt
116     ke = 0.5 * m * np.sum(vx**2 + vy**2)
117     return ke, pe, virial
118
119 def compute_momentum(vx, vy):
120     px = np.sum(m * vx)
121     py = np.sum(m * vy)
122     total_momentum = np.sqrt(px**2 + py**2)
123     return total_momentum
124
125 vx, vy = check_momentum(vx, vy)
126 ke = 0.5 * np.sum(vx**2 + vy**2)
127 pe, virial = accel(x, y, ax, ay)
128 total_momentum = compute_momentum(vx, vy)
129 print(f"{'time':>6}_{'E':>12}_{'Momentum':>12}_{'T':>12}_{'P':>12}")
130
131 t_values = []
132 T_values = []
133 E_values = []
134 P_values = []
135 momentum_values = []
136 pressure_diff_values = []
137
138 t = 0.0
139 while t < 5.0:
140     d=2
141     ke, pe, virial = verlet(x, y, vx, vy, ax, ay)
142     total_energy = ke + pe
143     total_momentum = compute_momentum(vx, vy)
144     pressure = (0.5 * virial) / area
145
146     T = (2 * ke*epsilon) / ((d * N-d) * k_B)
147     pressure = (N * k_B * T / (area*(sigma**2))) + (virial / (d * area))
148     P_ideal = (N * k_B *T) / (area*(sigma**2))
149

```

```

150     pressure_diff = pressure - P_ideal
151
152     t_values.append(t)
153     E_values.append(total_energy)
154     P_values.append(pressure)
155     momentum_values.append(total_momentum)
156     pressure_diff_values.append(pressure_diff)
157     #PRINT(F"{T:6.2F} {TOTAL_ENERGY:12.4F} {TOTAL_MOMENTUM:12.4F} {2*KE*MASS*(1.57E2)**2/ (N
        * K_B):12.4E} {PRESSURE:12.4E}")
158     T_values.append(T*1.65e-21)
159     t += dt
160
161 plt.figure(figsize=(12, 8))
162
163 plt.subplot(2, 1, 1)
164 plt.plot(t_values, E_values, label='Total_Energy_(E)', color='blue')
165 plt.ylim(np.min(E_values)-10, np.max(E_values)+10)
166 plt.xlabel('Time_(t)')
167 plt.ylabel('Total_Energy_(E)')
168 plt.title('Total_Energy_vs_Time')
169 plt.grid(True)
170 plt.legend()
171
172 plt.subplot(2, 1, 2)
173 plt.plot(t_values, momentum_values, label='Total_Momentum', color='red')
174 plt.ylim(0, 10*np.max(momentum_values))
175 plt.xlabel('Time_(t)')
176 plt.ylabel('Total_Momentum')
177 plt.title('Total_Momentum_vs_Time')
178 plt.grid(True)
179 plt.legend()
180
181 plt.tight_layout()
182 plt.show()
183
184 plt.figure(figsize=(12, 8))
185
186 plt.subplot(2, 1, 1)
187 plt.plot(t_values, T_values, label='Temperature', color='blue')
188 T_mean = np.mean(T_values)
189 plt.axhline(y=T_mean, color='red', linestyle='--', label=f'Mean_Temperature_={T_mean:.2f}')
190 plt.xlabel('Time_(t)')
191 plt.ylabel('Temperature')
192 plt.title('Temperature_vs_Time')
193 plt.grid(True)
194 plt.legend()
195

```

```

196 plt.subplot(2, 1, 2)
197 plt.plot(t_values, P_values, label='Pressure', color='red')
198 plt.xlabel('Time_(t)')
199 plt.ylabel('Pressure')
200 plt.title('Pressure_vs_Time')
201 plt.grid(True)
202 plt.legend()
203
204 plt.tight_layout()
205 plt.show()
206
207 plt.figure(figsize=(10, 6))
208 plt.plot(t_values, pressure_diff_values, label="PressureDifference_(Simulation_Ideal)")
209 plt.ylim(np.min(pressure_diff_values)-1,np.max(pressure_diff_values)+1)
210 plt.xlabel('Time_(s)')
211 plt.ylabel('PressureDifference_(Pa)')
212 plt.title('Difference_Between_Simulation_Pressure_and_Ideal_Gas_Pressure')
213 plt.legend()
214 plt.grid(True)
215 plt.show()

```

c 代码如下

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 sigma = 3.4e-10
5 epsilon = 1.65e-21
6 mass = 6.69e-26
7 tau = sigma * np.sqrt(mass / epsilon)
8 k_B = 1.38e-23
9
10 dt = 0.005
11 dt2 = dt**2
12 N = 18
13 Lx, Ly = 12.0, 6.0
14 k_B = 1.38e-23
15 temperature = 100
16 m = 1
17 area = Lx * Ly
18
19 def create_lattice(N, Lx, Ly):
20     positions = np.zeros((N, 2))
21     num_rows = int(np.sqrt(N))
22     num_cols = int(np.ceil(N / num_rows))
23     x_spacing = Lx / num_cols

```

```

24     y_spacing = Ly / num_rows
25
26     index = 0
27     for row in range(num_rows):
28         for col in range(num_cols):
29             if index >= N:
30                 break
31             x_pos = (col + 0.5) * x_spacing
32             y_pos = (row + 0.5) * y_spacing
33             positions[index] = [x_pos, y_pos]
34             index += 1
35     return positions
36
37 def init_velocity(N, temperature, mass):
38     stddev = np.sqrt(k_B * temperature / mass)
39     velocities = np.random.normal(0, stddev, (N, 2))
40     velocities -= np.mean(velocities, axis=0)
41     return velocities
42
43 positions = create_lattice(N, Lx, Ly)
44 velocities = init_velocity(N, temperature, m)
45
46 x, y = positions[:, 0], positions[:, 1]
47 vx, vy = velocities[:, 0], velocities[:, 1]
48 ax = np.zeros(N)
49 ay = np.zeros(N)
50
51 plt.figure(figsize=(6, 6))
52 plt.scatter(x, y, c='blue', label='Particles', s=100)
53 plt.xlim(0, Lx)
54 plt.ylim(0, Ly)
55 plt.gca().set_aspect('equal', adjustable='box')
56 plt.xlabel('X_Position(Å)')
57 plt.ylabel('Y_Position(Å)')
58 plt.title('Particle_Positions_in_6x6_Box')
59 plt.grid(color='gray', linestyle='--', linewidth=0.5)
60 plt.legend()
61 plt.show()
62
63
64 def check_momentum(vx, vy):
65     vx -= np.sum(vx) / N
66     vy -= np.sum(vy) / N
67     return vx, vy
68
69 def pbc(pos, L):
70     return (pos + L) % L

```

```
71
72 def separation(ds, L):
73     return ds - L * np.round(ds / L)
74
75 def force(dx, dy):
76     r2 = dx**2 + dy**2
77     if r2 == 0:
78         return 0, 0, 0
79     rm2 = 1.0 / r2
80     rm6 = rm2**3
81     f_over_r = 24 * rm6 * (2 * rm6 - 1) * rm2
82     fx = f_over_r * dx
83     fy = f_over_r * dy
84     pot = 4.0 * (rm6**2 - rm6)
85     return fx, fy, pot
86
87 def accel(x, y, ax, ay):
88     ax.fill(0)
89     ay.fill(0)
90     pe = 0.0
91     virial = 0.0
92     for i in range(N - 1):
93         for j in range(i + 1, N):
94             dx = separation(x[i] - x[j], Lx)
95             dy = separation(y[i] - y[j], Ly)
96             fx, fy, pot = force(dx, dy)
97             ax[i] += fx
98             ay[i] += fy
99             ax[j] -= fx
100            ay[j] -= fy
101            pe += pot
102            virial += dx * fx + dy * fy
103     return pe, virial
104
105 def verlet(x, y, vx, vy, ax, ay):
106     x += vx * dt + 0.5 * ax * dt2
107     y += vy * dt + 0.5 * ay * dt2
108     x = pbc(x, Lx)
109     y = pbc(y, Ly)
110     vx += 0.5 * ax * dt
111     vy += 0.5 * ay * dt
112     pe, virial = accel(x, y, ax, ay)
113     vx += 0.5 * ax * dt
114     vy += 0.5 * ay * dt
115     ke = 0.5 * m * np.sum(vx**2 + vy**2)
116     return ke, pe, virial
117
```

```

118 def compute_momentum(vx, vy):
119     px = np.sum(m * vx)
120     py = np.sum(m * vy)
121     total_momentum = np.sqrt(px**2 + py**2)
122     return total_momentum
123
124 n_t_values = []
125 left_half = Lx / 2
126
127 t = 0.0
128 while t < 300.0:
129     ke, pe, virial = verlet(x, y, vx, vy, ax, ay)
130
131     n_t = np.sum(x < left_half)
132     n_t_values.append(n_t)
133
134     t += dt
135 n_t_values = n_t_values[0:60000]
136
137 t_values = np.arange(0, 300.0, dt)
138 print(len(t_values))
139
140 n_t_mean = []
141
142 for i in range(len(t_values)):
143     n_t_mean.append(np.mean(n_t_values[0:i]))
144
145 plt.figure(figsize=(16, 6))
146 plt.plot(t_values, n_t_values, label='$n(t)$: Number of particles in left half')
147 plt.xlabel('Time (t)')
148 plt.ylabel('$n(t)$')
149 plt.title('Number of Particles in Left Half vs Time')
150 plt.legend()
151 plt.grid(True)
152 plt.show()
153
154 plt.figure(figsize=(16, 6))
155 plt.plot(t_values, n_t_mean, label='$n(t)$: Average number of particles in left half')
156 plt.xlabel('Time (t)')
157 plt.ylabel('$n(t)$')
158 plt.title('Average Number of Particles in Left Half vs Time')
159 plt.legend()
160 plt.grid(True)
161 plt.show()
162
163 print(f"Time-averaged number of particles in left half: {n_t_mean:.2f}")

```

Problem 2

Ising Model

a. 可以看到 T_c 随着材料扩大，逐渐逼近无限大系统的相变温度，大约在 2.35 附近

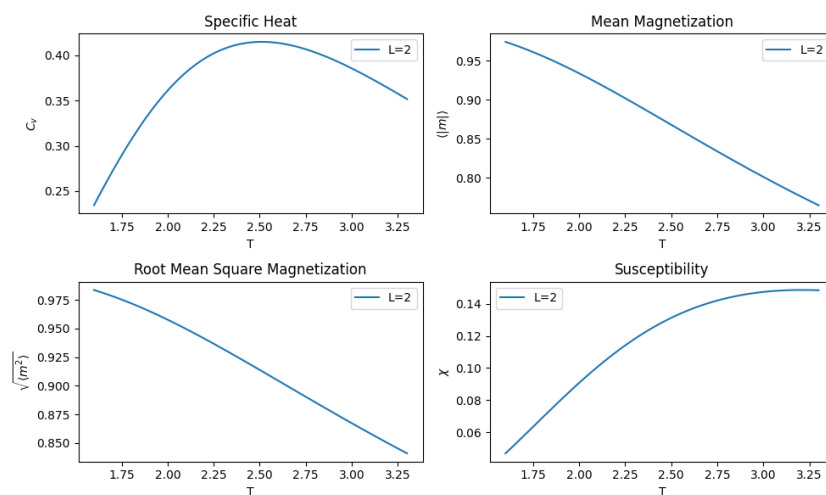


图 12: $L=2$

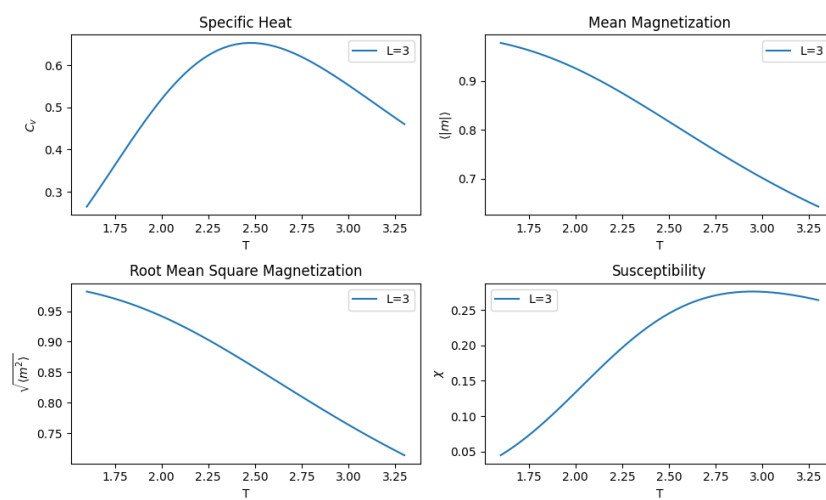
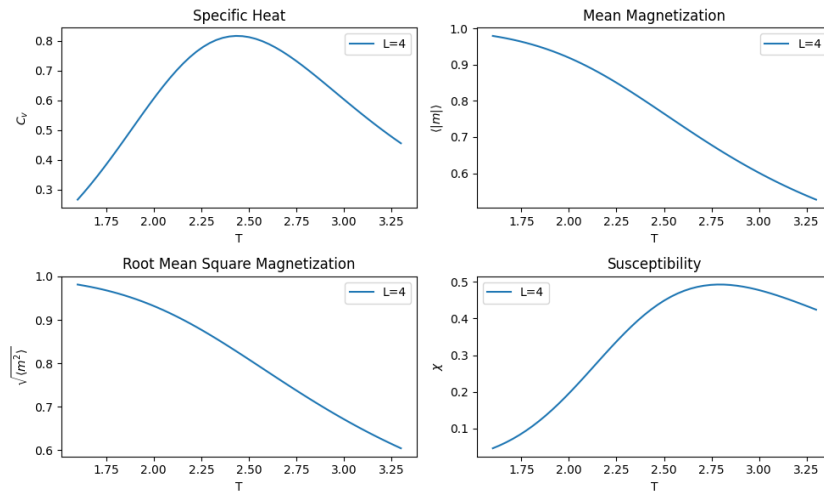
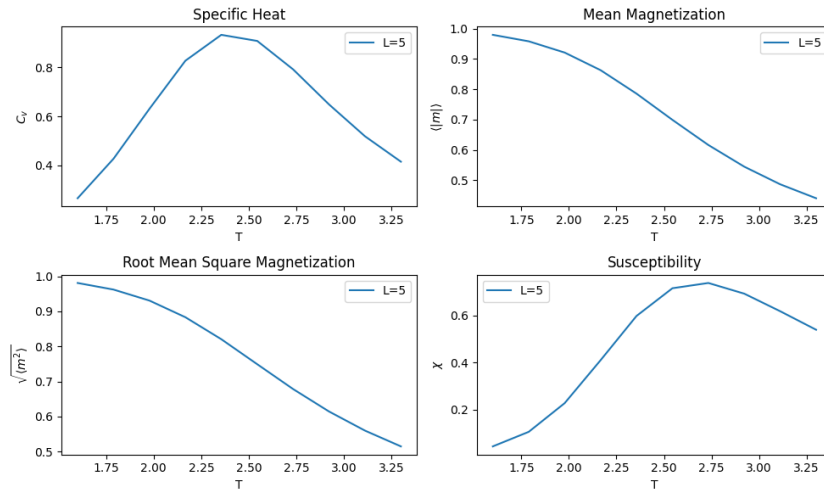


图 13: $L=3$

图 14: $L=4$ 图 15: $L=5$

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from itertools import product
4
5 J = 1.0
6 L_values = [2,3,4,5]
7 T_range = np.linspace(1.6, 3.3, 10)
8
9
10 def compute_ising_properties(L, T_range):
11
12     N = L * L
13     print(1)
14     states = np.array(list(product([-1, 1], repeat=N)))

```

```

15     print(states)
16     E_values = []
17     M_values = []
18
19     for state in states:
20         state = state.reshape(L, L)
21         E = 0
22         M = np.sum(state)
23         for i in range(L):
24             for j in range(L):
25                 E -= J * state[i, j] * (state[(i + 1) % L, j] + state[i, (j + 1) % L])
26         E_values.append(E)
27         M_values.append(M)
28     E_values = np.array(E_values)
29     M_values = np.array(M_values)
30
31     results = {"Cv": [], "m_mean": [], "sqrt_m2": [], "m_max": [], "chi": []}
32     for T in T_range:
33         beta = 1 / T
34         Z = np.sum(np.exp(-beta * E_values))
35         E_mean = np.sum(E_values * np.exp(-beta * E_values)) / Z
36         E2_mean = np.sum((E_values ** 2) * np.exp(-beta * E_values)) / Z
37         M_mean = np.sum(np.abs(M_values) * np.exp(-beta * E_values)) / Z
38         M2_mean = np.sum((M_values ** 2) * np.exp(-beta * E_values)) / Z
39         M_max = np.max(np.abs(M_values))
40
41         Cv = (E2_mean - E_mean ** 2) * beta ** 2 / N
42         chi = (M2_mean - M_mean ** 2) * beta / N
43
44         results["Cv"].append(Cv)
45         results["m_mean"].append(M_mean / N)
46         results["sqrt_m2"].append(np.sqrt(M2_mean) / N)
47         results["m_max"].append(M_max / N)
48         results["chi"].append(chi)
49     return results
50
51
52 for L in [1]:
53     results = compute_ising_properties(5, T_range)
54     plt.figure(figsize=(10, 6))
55
56     # 绘制 Cv
57     plt.subplot(2, 2, 1)
58     plt.plot(T_range, results["Cv"], label=f"L={L}")
59     plt.xlabel("T")
60     plt.ylabel("$C_v$")
61     plt.title("Specific_Heat")

```

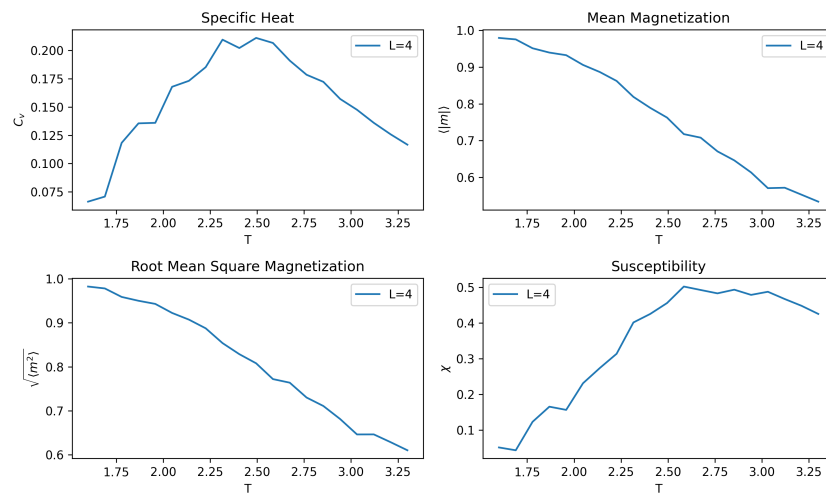
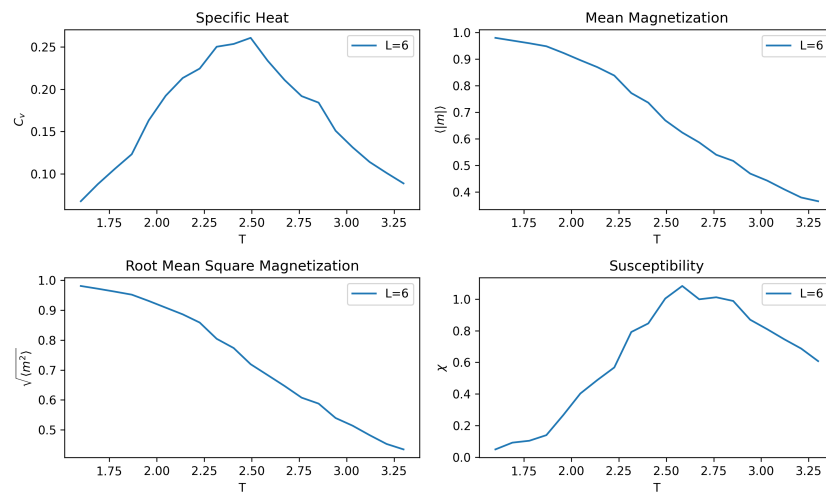
```

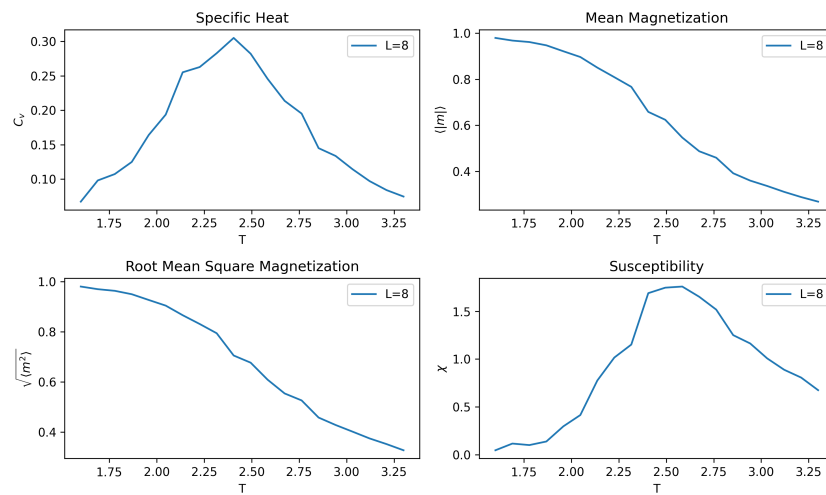
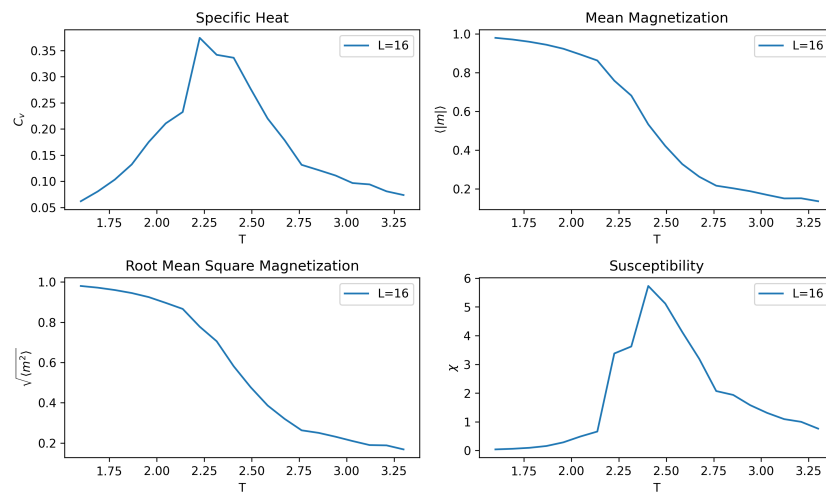
62     plt.legend()
63
64     # 绘制  $\langle |M| \rangle$ 
65     plt.subplot(2, 2, 2)
66     plt.plot(T_range, results["m_mean"], label=f"L={L}")
67     plt.xlabel("T")
68     plt.ylabel(" $\langle |m| \rangle$ ")
69     plt.title("Mean Magnetization")
70     plt.legend()
71
72     # 绘制  $\sqrt{\langle M^2 \rangle}$ 
73     plt.subplot(2, 2, 3)
74     plt.plot(T_range, results["sqrt_m2"], label=f"L={L}")
75     plt.xlabel("T")
76     plt.ylabel(" $\sqrt{\langle m^2 \rangle}$ ")
77     plt.title("Root Mean Square Magnetization")
78     plt.legend()
79
80     # 绘制  $\chi$ 
81     plt.subplot(2, 2, 4)
82     plt.plot(T_range, results["chi"], label=f"L={L}")
83     plt.xlabel("T")
84     plt.ylabel(" $\chi$ ")
85     plt.title("Susceptibility")
86     plt.legend()
87
88     plt.tight_layout()
89     plt.show()

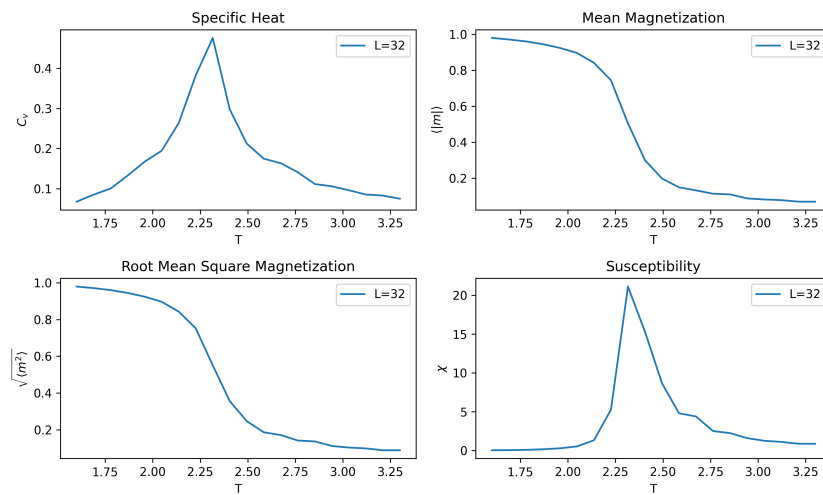
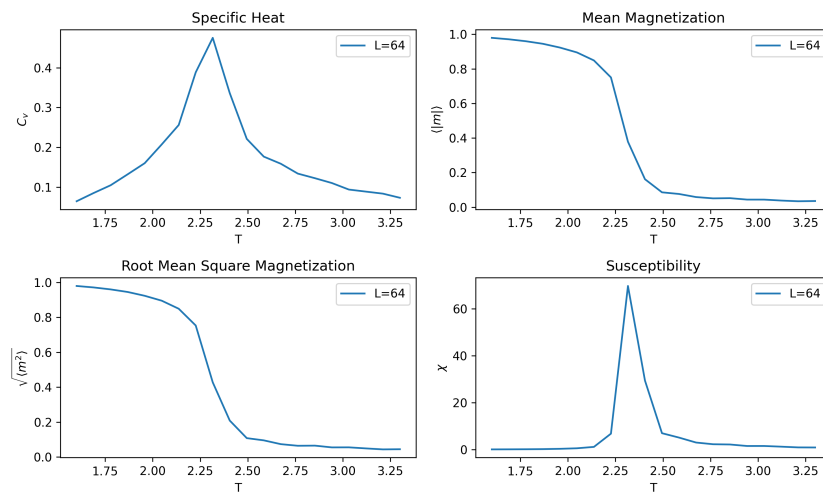
```

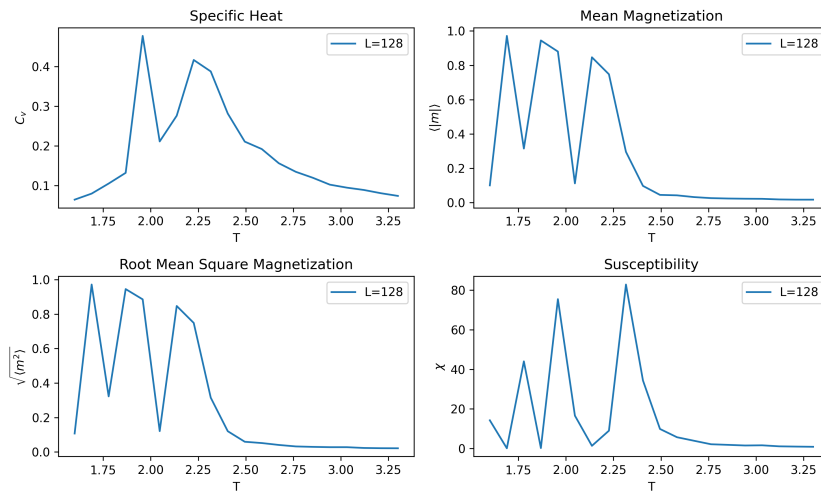
b. 可以看到 T_c 随着材料扩大逼近大约 2.25 附近的过程，其中 $L=128$ 的图片由于材料较大，在有限计算资源下难以达到接近平衡态的状态，进而出现多个峰值，可以估计出真实峰值应该接近 2.25。

蒙特卡洛模拟参数: 10000 步 MC 来达到平衡态, 之后 5000 步进行测量。由于 $L=64$ 、 $L=128$ 的材料尺寸很大，在这个 MC 模拟参数下运算一次需要很长时间，且计算资源有限，因此没有设置更大的模拟参数，如果计算资源足够可以设置更大参数，得到更准确结果。

图 16: $L=4$ 图 17: $L=6$

图 18: $L=8$ 图 19: $L=16$

图 20: $L=32$ 图 21: $L=64$

图 22: $L=128$

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 J = 1.0
5 L_values = [4, 6, 8, 10, 16, 32, 64, 128]
6 T_range = np.linspace(1.6, 3.3, 20)
7 MCS = 100
8 measurements = 50
9
10
11 def metropolis_step(spins, beta, L):
12
13     for _ in range(L * L):
14         i, j = np.random.randint(0, L), np.random.randint(0, L)
15         dE = 2 * J * spins[i, j] * (
16             spins[(i + 1) % L, j] + spins[i, (j + 1) % L] +
17             spins[(i - 1) % L, j] + spins[i, (j - 1) % L]
18         )
19         if dE <= 0 or np.random.rand() < np.exp(-beta * dE):
20             spins[i, j] *= -1
21
22
23 def monte_carlo_simulation(L, T_range, MCS, measurements):
24
25     N = L * L
26     results = {"Cv": [], "m_mean": [], "sqrt_m2": [], "m_max": [], "chi": []}
27
28     for T in T_range:
29         beta = 1 / T
30         spins = np.random.choice([-1, 1], size=(L, L))

```

```

31     E_values = []
32     M_values = []
33
34     for _ in range(MCS):
35         metropolis_step(spins, beta, L)
36
37     for _ in range(measurements):
38         metropolis_step(spins, beta, L)
39         E = -J * np.sum(spins * (
40             np.roll(spins, 1, axis=0) +
41             np.roll(spins, 1, axis=1)
42         )) / 2
43         M = np.sum(spins)
44         E_values.append(E)
45         M_values.append(M)
46
47     E_values = np.array(E_values)
48     M_values = np.array(M_values)
49     E_mean = np.mean(E_values)
50     E2_mean = np.mean(E_values ** 2)
51     M_mean = np.mean(np.abs(M_values))
52     M2_mean = np.mean(M_values ** 2)
53
54     Cv = beta ** 2 * (E2_mean - E_mean ** 2) / N
55     chi = beta * (M2_mean - M_mean ** 2) / N
56     m_max = np.max(np.abs(M_values)) / N
57
58     results["Cv"].append(Cv)
59     results["m_mean"].append(M_mean / N)
60     results["sqrt_m2"].append(np.sqrt(M2_mean) / N)
61     results["m_max"].append(m_max)
62     results["chi"].append(chi)
63
64     return results
65
66
67 for L in L_values:
68     print(f"Running simulation for L={L}...")
69     results = monte_carlo_simulation(L, T_range, MCS, measurements)
70
71     plt.figure(figsize=(10, 6))
72
73     # 绘制 Cv
74     plt.subplot(2, 2, 1)
75     plt.plot(T_range, results["Cv"], label=f"L={L}")
76     plt.xlabel("T")
77     plt.ylabel("$C_v$")

```



```
78     plt.title("Specific_Heat")
79     plt.legend()
80
81     # 绘制  $\langle |M| \rangle$ 
82     plt.subplot(2, 2, 2)
83     plt.plot(T_range, results["m_mean"], label=f"L={L}")
84     plt.xlabel("T")
85     plt.ylabel(" $\langle |m| \rangle$ ")
86     plt.title("Mean_Magnetization")
87     plt.legend()
88
89     # 绘制  $\sqrt{\langle M^2 \rangle}$ 
90     plt.subplot(2, 2, 3)
91     plt.plot(T_range, results["sqrt_m2"], label=f"L={L}")
92     plt.xlabel("T")
93     plt.ylabel(" $\sqrt{\langle m^2 \rangle}$ ")
94     plt.title("Root_Mean_Square_Magnetization")
95     plt.legend()
96
97     # 绘制  $\chi$ 
98     plt.subplot(2, 2, 4)
99     plt.plot(T_range, results["chi"], label=f"L={L}")
100    plt.xlabel("T")
101    plt.ylabel(" $\chi$ ")
102    plt.title("Susceptibility")
103    plt.legend()
104
105    plt.tight_layout()
106    plt.show()
```
