# 计算物理第四次作业
# 分子动力学问题

张楚珩（121120173）

July 8, 2014

## 1 问题描述

本次作业是一个分子动力学模拟问题。它通过直接求解物理系统中每一个粒子所遵循的运动方程，以获得相空间中的位形变化，由此模拟整个系统的行为。

本次作业着重研究Melting现象。并考察Melting的相变温度随密度的变化关系。

在本次作业中，构建一个6×6×6的晶胞进行模拟。

## 2 计算模型

分类上来讲，这是一个模拟平衡态的分子动力学模拟，由于我们要模拟的模型上有动能恒定（即温度恒定）的约束，因此它是正则系统的分子动力学模拟。在正则系统中，系统能量可能有一定的涨落，但由于系统置于热浴之中，系统的温度保持一定。即，系统中粒子数N、体积V、温度T和总动量P保持恒定。

### 2.1 设定模拟物理模型

我们假定体系中每个粒子都遵循牛顿运动学方程。粒子受到其他分子的作用力由粒子所处的势能函数$V(\vec{r})$决定。

$$m\ddot{\vec{r}}_i(t) = \vec{F}_i(\vec{r}) = -\frac{dV_i(\vec{r})}{dr}, \quad i = 1, 2, \cdots, N \tag{1}$$

模拟中我们采用Lennard-Jones位势，

$$V(r) = 4\varepsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}] \tag{2}$$

其中，$-\varepsilon$是位势的最小值，它可以确定能量的单位；$\sigma$是$V(r)=0$时的$r$值，它可以确定长度的单位。由这两个量，我们可以导出其他物理量的单位，从而将计算无量纲化。量纲的单位如下：

$$\begin{cases} \text{质量：} & m \\ \text{长度：} & \sigma \\ \text{能量：} & \varepsilon \\ \text{时间：} & \sqrt{m\sigma^2/\varepsilon} \\ \text{速度：} & \sqrt{\varepsilon/m} \\ \text{温度：} & \varepsilon/k_B \end{cases} \tag{3}$$

其中,$m$是粒子的质量，$k_B$是玻尔兹曼常数。

对势函数求导，可以得到任一粒子，任意分量上的受力

$$F_{i,x}=48(\frac{\varepsilon}{\sigma^2})\sum_{j\neq i}^{N}\sum_{j=1}(x_i-x_j)[(\frac{\sigma}{r})^{14}-\frac{1}{2}(\frac{\sigma}{r})^{8}] \tag{4}$$

同时我们采用周期性边界条件，并且在考虑分子之间的相互作用的时候采用最小像力的约定。我们认为每一个粒子只和与它距离范围在$\pm\frac{L}{2}$范围内的粒子间有相互作用力。

## 2.2 设定初始条件

我们采用以下方式给定初始条件。在设定初始条件的时候，我们把所有的6×6×6的粒子等距放置在给定的L×L×L的晶胞里面，之后，我们随机给所有的粒子赋予初始的速度，并且利用标度因子将由此产生的温度控制在给定的温度下。初始速度各个分量的分布服从计算机伪随机数均匀分布。

## 2.3 数值求解运动方程

我们采用速度verlet算法进行运动方程的求解。速度verlet算法分为以下三个步骤（已做无量纲化处理）：

第一步：计算出n+1步时所有粒子的速度（程序中velocityverlet1函数的一部分）：

$$r_i^{(n+1)}=r_i^{(n)}+hv_i^{(n)}+\frac{1}{2}h^2F_i^{(n)} \tag{5}$$

第二步：计算出相应的受力（程序中force函数）：

$$F_{i,x}=48\sum_{j\neq i}^{N}\sum_{j=1}(x_i-x_j)[(\frac{1}{r})^{14}-\frac{1}{2}(\frac{1}{r})^{8}] \tag{6}$$

第三步：计算出n+1步时所有粒子的速度（程序中velocityverlet1函数的一部分和velocityverlet2函数）：

$$v_i^{(n+1)}=v_i^{(n)}+h(F_i^{(n)}+F_i^{(n+1)})/2 \tag{7}$$

并且计算出相应的动能

$$E_k=\frac{1}{2}\sum_i(v_i^{(n+1)})^2 \tag{8}$$

和相应的标度因子

$$\beta = \sqrt{\frac{(3N-4)kT}{\sum_i (v_i^{(n+1)})^2}} \tag{9}$$

对速度进行标度调整，作为下一次的计算值

$$\beta (v_i^{(n+1)})^2 \to (v_i^{(n+1)})^2 \tag{10}$$

### 2.4 利用位置涨落衡量融化过程

为了衡量融化过程，我们建立一个宏观统计量——平均位置涨落来观测样品是否融化。

$$\langle \Delta r^2 \rangle = \sum_i \frac{(r_i - r_{0i})^2}{L^2} \tag{11}$$

在本例中，当$\langle \Delta r^2 \rangle > \frac{1}{36}$时，即平均涨落已经使它容易到达周围粒子的平衡位置时，我们认为，样品已经融化。

## 3 程序源代码

```
1  //
2  //  main.c
3  //  Molecule Dynamics
4  //
5  //  Created by ZHANG CH on 14-6-8.
6  //  Copyright (c) 年2014 NJU. All rights reserved.
7  //
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <time.h>
12 #include <math.h>
13
14 #define GRID_NUM (6)
15 #define L (10)
16 #define H (0.001)
17 #define STEPMAX (5000)
18 #define TEMPERATURE  (0.01)
19
20 ////////////////////////vecter functions/////////////////////////
21 typedef struct {
22     double x;
23     double y;
24     double z;
```

```
25 } vecter;
26
27 vecter mkvecter(double x, double y, double z)
28 {
29     vecter ans = {x,y,z};
30     return ans;
31 }
32
33 vecter sqrvecter(vecter v)
34 {
35     return mkvecter(v.x*v.x, v.y*v.y, v.z*v.z);
36 }
37
38 vecter sqrtvecter(vecter v)
39 {
40     return mkvecter(sqrt(v.x), sqrt(v.y), sqrt(v.z));
41 }
42
43 vecter addvecter(vecter v1, vecter v2)
44 {
45     return mkvecter(v1.x+v2.x, v1.y+v2.y, v1.z+v2.z);
46 }
47
48 vecter add3vecter(vecter v1, vecter v2, vecter v3)
49 {
50     return mkvecter(v1.x+v2.x+v3.x, v1.y+v2.y+v3.y, v1.z+v2.z+v3.z);
51 }
52
53 vecter substractvecter(vecter v1, vecter v2)
54 {
55     return mkvecter(v1.x-v2.x, v1.y-v2.y, v1.z-v2.z);
56 }
57
58 vecter multiplyvecter(vecter v, double d)
59 {
60     return mkvecter(v.x*d, v.y*d, v.z*d);
61 }
62
63 double sum(vecter v)
64 {
65     return (v.x+v.y+v.z);
66 }
67
68 vecter sectionvecter(vecter v,double smin,double smax)
69 {
70     double s = smax - smin;
```

```
71      return mkvecter(v.x-floor((v.x-smin)/s)*s, v.y-floor((v.y-smin)/s)*
            s, v.z-floor((v.z-smin)/s)*s);
72  }
73
74  vecter dotmulvecter(vecter v1, vecter v2)
75  {
76      return mkvecter(v1.x*v2.x, v1.y*v2.y, v1.z*v2.z);
77  }
78
79  ////////////////////veter functions////////////////////
80
81  vecter r[GRID_NUM][GRID_NUM][GRID_NUM] = {0};
82  vecter r0[GRID_NUM][GRID_NUM][GRID_NUM] = {0};
83  vecter v[GRID_NUM][GRID_NUM][GRID_NUM] = {0};
84  vecter f[GRID_NUM][GRID_NUM][GRID_NUM] = {0};
85  vecter *pr = (vecter *)r;
86  vecter *pr0 = (vecter *)r0; //record the equilibrium position of
        particles
87  vecter *pv = (vecter *)v;
88  vecter *pf = (vecter *)f;
89  FILE *fp;
90
91  void init();
92  void force();
93  void volecityverlet1();
94  void volecityverlet2();
95  void printinit();
96  void print();
97  void printend();
98  double fluctuation();
99  vecter calBeta(); //return a vecter whose components are total kenetic
        energy, beta, 0
100 vecter calcenterV();
101
102
103 int main(int argc, const char * argv[])
104 {
105     srand((unsigned)time(NULL)); //use time as the seed of random
            function
106     int t=0;
107     double fct;
108     printinit();                //initialize file I/O stream
109     init();                             //set the initial position and
            initial velocity of particles
110     for (t=0; t<STEPMAX; t++)
111     {
```

```
112         volecityverlet1();
113         force();
114         volecityverlet2();
115         fct = fluctuation();            //calculate the position
                fluctuation of particles
116         fprintf(fp, "%lf , ", fct);
117     }
118
119     printend();
120     return 0;
121 }
122
123 void init()
124 {
125     int i,j,k;
126     double beta;
127     vecter centerV;
128
129     for (i=0; i<216; i++)
130     {
131         pr[i] = mkvecter(0.0, 0.0, 0.0);
132         pv[i] = mkvecter(0.0, 0.0, 0.0);
133     }
134
135     for (i=0; i<GRID_NUM; i++) {
136         for (j=0; j<GRID_NUM; j++) {
137             for (k=0; k<GRID_NUM; k++) {
138                 r[i][j][k] = mkvecter(((1.0/12.0)+(1.0/6.0)*i)*L,
                        ((1.0/12.0)+(1.0/6.0)*j)*L, ((1.0/12.0)+(1.0/6.0)*k)
                        *L);
139                 r0[i][j][k] = r[i][j][k];
140                 v[i][j][k] = mkvecter((double)(rand())/RAND_MAX - 0.5,
                        (double)(rand())/RAND_MAX - 0.5, (double)(rand())/
                        RAND_MAX - 0.5);
141             }
142         }
143     }
144     beta = calBeta().y;
145     centerV = calcenterV();
146     for (i=0; i<GRID_NUM; i++) {
147         for (j=0; j<GRID_NUM; j++) {
148             for (k=0; k<GRID_NUM; k++) {
149                 v[i][j][k] = multiplyvecter(substractvecter(v[i][j][k],
                        centerV), beta);
150             }
151         }
```

```
152     }
153 }
154
155 void force()
156 {
157     int i,j;
158     vecter rel;
159     double rsqri,r6thi,ff;
160     for (i=0; i<216; i++) {
161         pf[i] = mkvecter(0.0, 0.0, 0.0);
162     }
163     for (i=0; i<216; i++) {
164         for (j=0; j<i; j++) {
165             rel = substractvecter(pr[i], pr[j]);
166             rel = sectionvecter(rel,-L/2.0,L/2.0);
167             rsqri = 1.0/sum(sqrvecter(rel));
168             r6thi = pow(rsqri, 3);
169             ff = 48.0*rsqri*r6thi*(r6thi - 0.5);
170             pf[i] = addvecter(pf[i], dotmulvecter(mkvecter(rel.x<L
                    /2?1.0:0.0, rel.y<L/2?1.0:0.0, rel.z<L/2?1.0:0.0),
                    multiplyvecter(rel, ff)));
171             pf[j] = substractvecter(pf[j], dotmulvecter(mkvecter(rel.x<
                    L/2?1.0:0.0, rel.y<L/2?1.0:0.0, rel.z<L/2?1.0:0.0),
                    multiplyvecter(rel, ff)));
172         }
173     }
174 }
175
176 void volecityverlet1()
177 {
178     int i;
179     for (i=0; i<216; i++) {
180         pr[i] = add3vecter(pr[i], multiplyvecter(pv[i], H),
                multiplyvecter(pf[i], 0.5*H*H));
181         pv[i] = addvecter(pv[i], multiplyvecter(pf[i], 0.5*H));
182         pr[i] = sectionvecter(pr[i], 0, L);
183     }
184 }
185
186 void volecityverlet2()
187 {
188     int i;
189     double beta;
190     for (i=0; i<216; i++) {
191         pv[i] = addvecter(pv[i], multiplyvecter(pf[i], 0.5*H));
192     }
```

```
193     beta = calBeta().y;
194     for (i=0; i<216; i++) {
195         pv[i] = multiplyvecter(pv[i], beta);
196     }
197 }
198
199 double fluctuation()
200 {
201     int i;
202     double ans = 0.0;
203     for (i=0; i<216; i++) {
204         ans += sum(sqrvecter(substractvecter(pr[i], pr0[i])))/(L*L);
205     }
206     ans /= 216;
207     return ans;
208 }
209
210 vecter calBeta() //return a vecter whose components are total kenetic
        energy, beta, 0
211 {
212     double sumV2 = 0;
213     int i;
214     for (i=1; i<216; i++) {
215         sumV2 += sum(sqrvecter(pv[i]));
216     }
217     sumV2 /= 216;
218     return mkvecter(sumV2,sqrt(3*TEMPERATURE/sumV2),0);
219 }
220
221 vecter calcenterV()
222 {
223     vecter centerV = mkvecter(0, 0, 0);
224     int i;
225     for (i=1; i<216; i++) {
226         centerV = addvecter(centerV, pv[i]);
227     }
228     return multiplyvecter(centerV, (double)1.0/216);
229 }
230
231 void printinit()
232 {
233     fp = fopen("output.txt", "w+");
234 }
235
236 void print(vecter *v)
237 {
```

```
238      int i;
239      for (i=0; i<216; i++)
240      {
241          fprintf(fp, "%lf   ", v[i].x);
242          fprintf(fp, "%lf   ", v[i].y);
243          fprintf(fp, "%lf   ", v[i].z);
244          fprintf(fp, "\n");
245      }
246  }
247
248  void printend()
249  {
250      fclose(fp);
251  }
```

# 4 计算结果

我们在晶胞中建立了网格，并且把216个粒子放置在网格中，如图1所示。

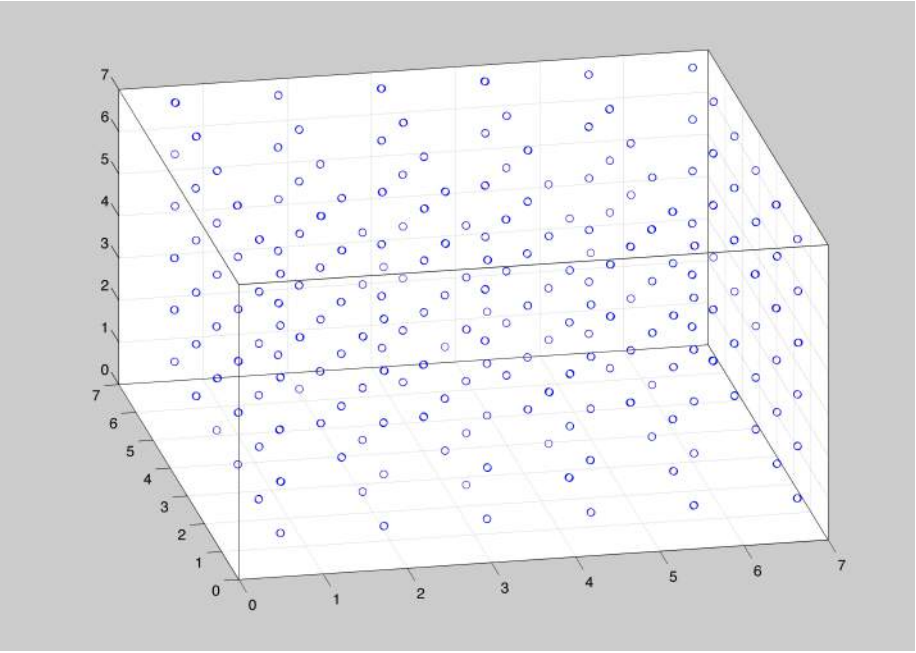


Figure 1: 按照位置建立网格，（重影代表初速度方向）

程序为每一个粒子建立初始位置和初始速度之后，粒子就按照各自的力学规律运动。如图2所属就是粒子前20个MD步子中的运动情况。

在模拟5000步的过程中，我们观察到了样品Melting的过程。其立体视图和x-y视图分别如图3和图4所示。

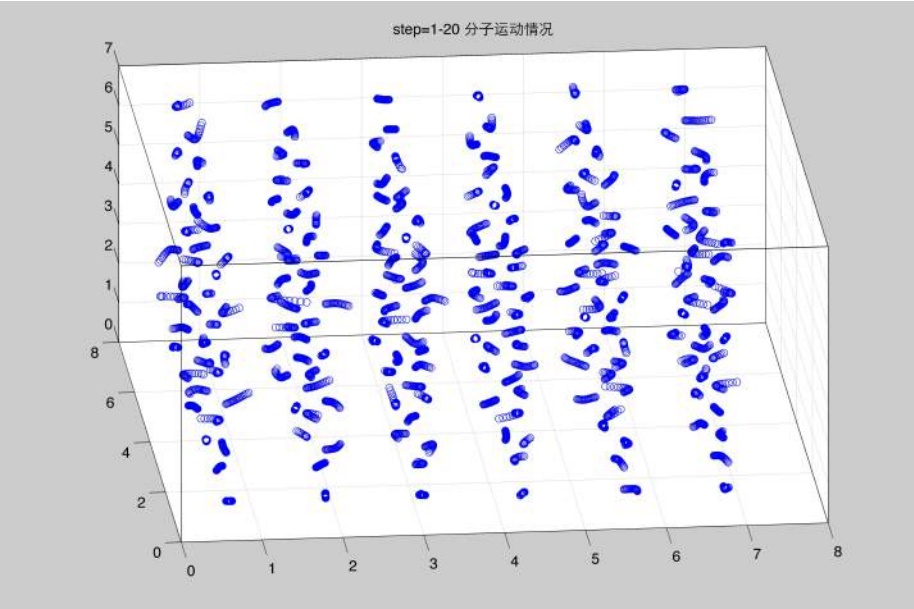


Figure 2: 粒子在前20个MD步子中的运动情况（步长h=0.001）

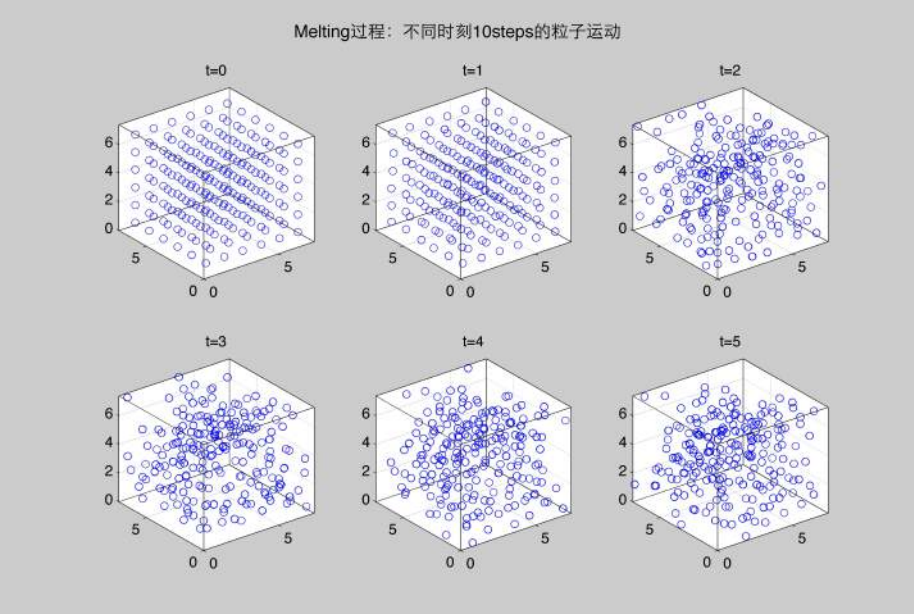


Figure 3: 融化过程：立体视图（步长h=0.001）

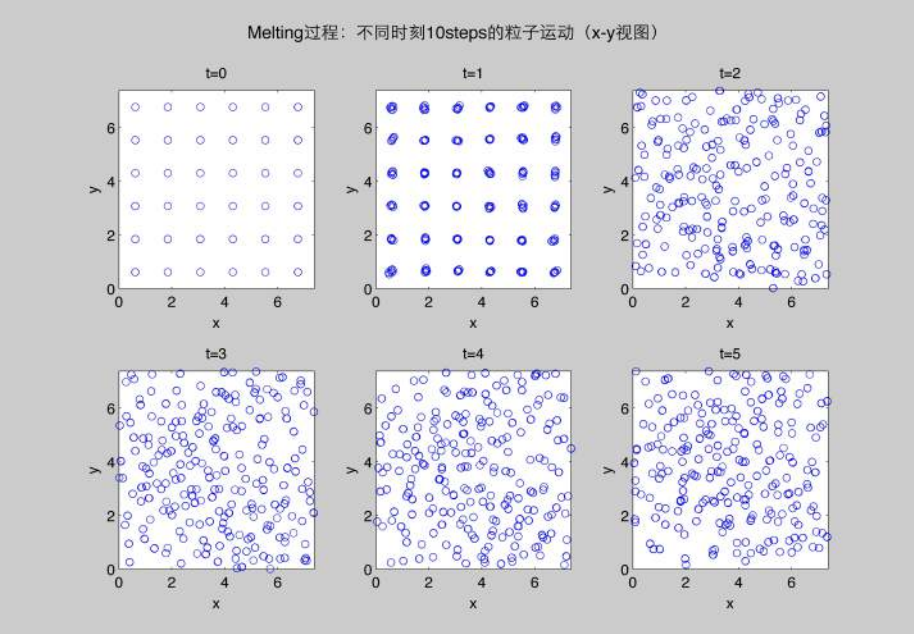


Figure 4: 融化过程：x-y视图（步长h=0.001）

样品的融化，说明在该状态下，样品存在的状态应该是非固态。如图5所示，样品在模拟的时间足够长的情况下，达到稳态。

为了研究融化温度随随密度的影响，我们绘制了在密度下（通过调节晶胞长度实现），涨落随温度的变化关系，当涨落$\frac{\langle \Delta r^2 \rangle}{L^2} > \frac{1}{36}$时我们认为其对应的温度可以使得样品融化。如图6、图7、图8和图9所示。我们可以发现，密度越小，融化所需要的温度越高。

此外，我们在L=7时对温度进行扫描，得到不同温度下5000MD步之后的的涨落大小，如图10所示。从图中可以看到样品的融化温度大约在temperature=0.012左右。

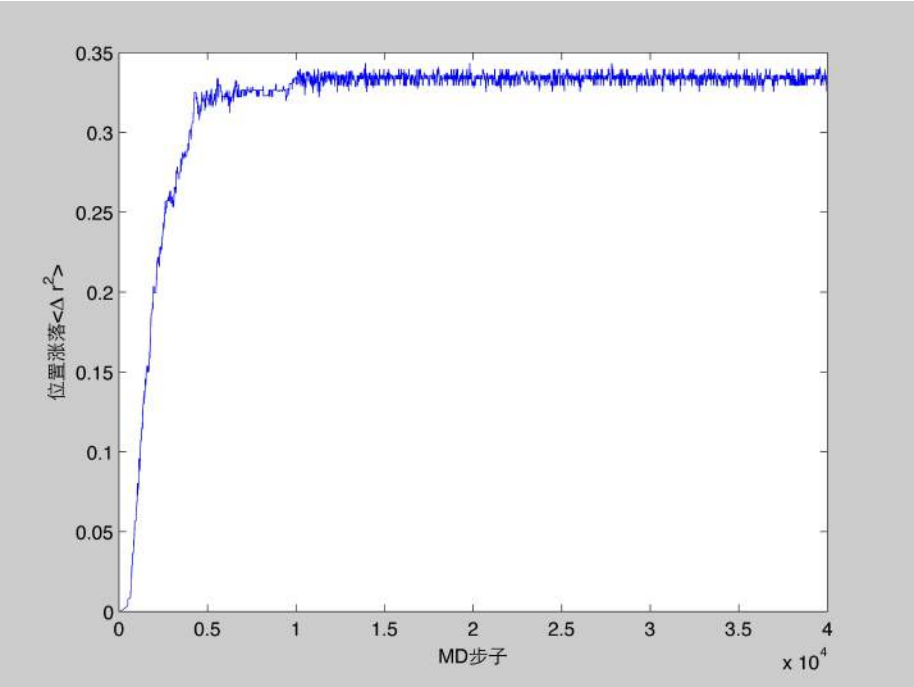


Figure 5: 样品在10000MD步之后，达到稳定状态

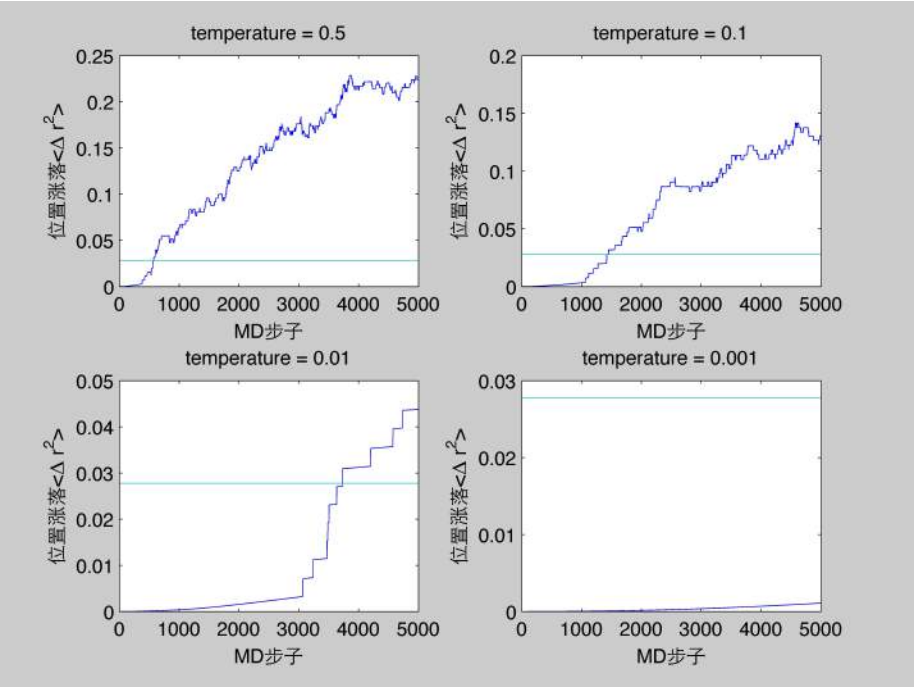


Figure 6: L=7时，涨落随温度的变化（步长h=0.001）

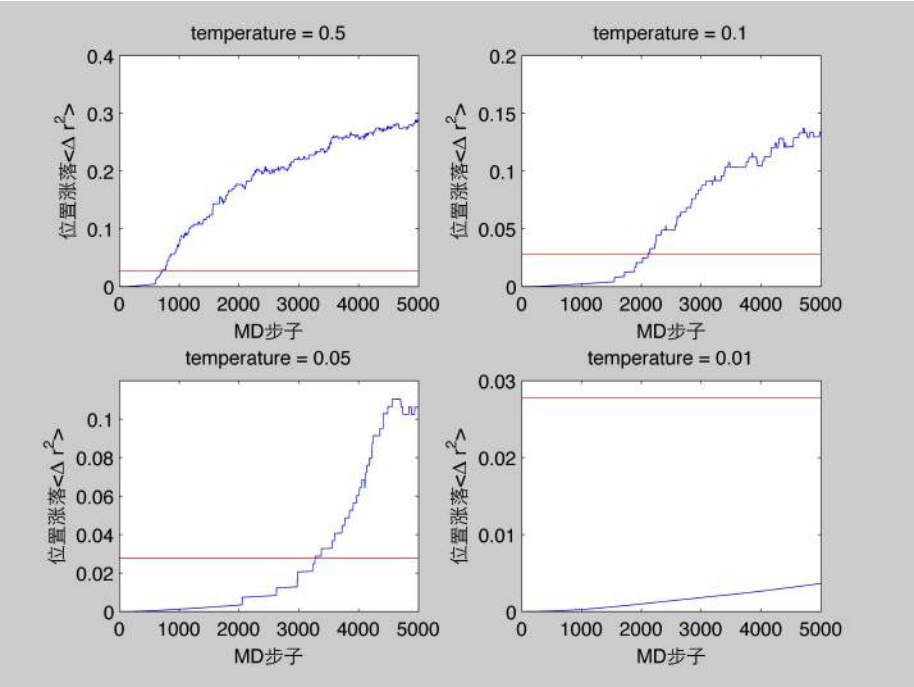


Figure 7: L=10时，涨落随温度的变化（步长h=0.001）

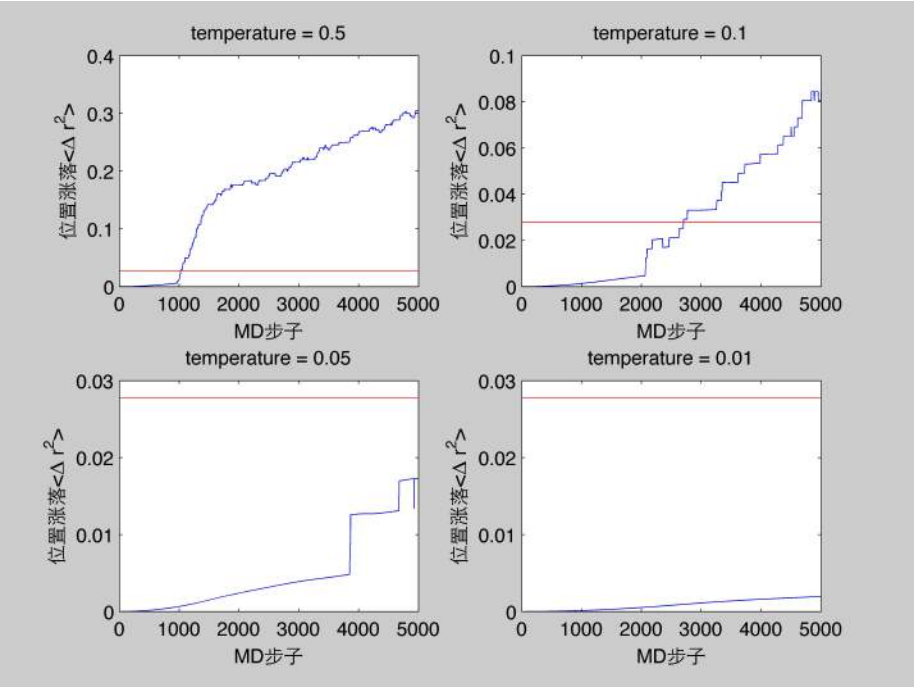


Figure 8: L=15时，涨落随温度的变化（步长h=0.001）

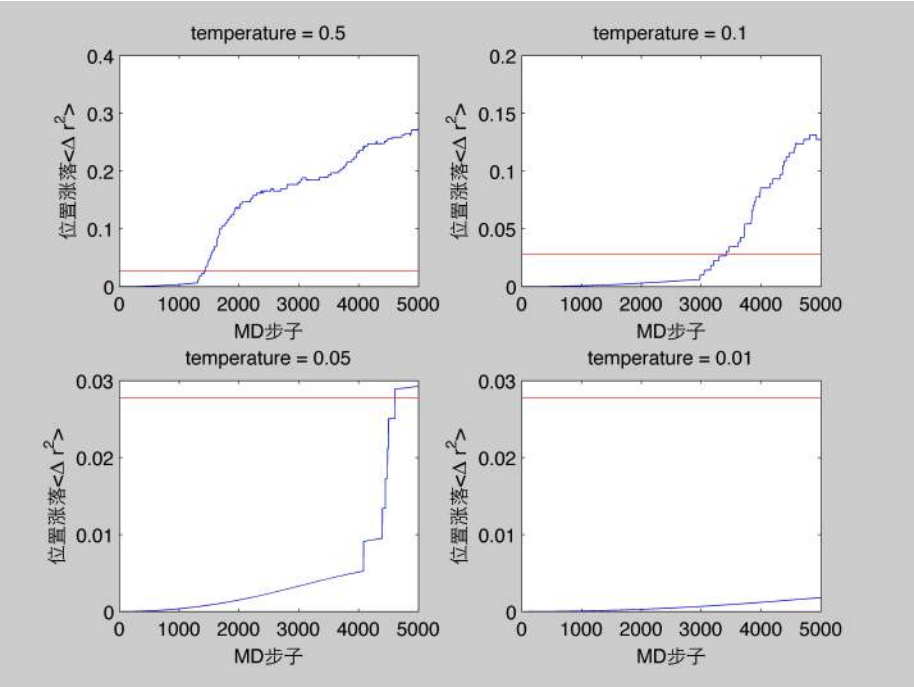


Figure 9: L=20时，涨落随温度的变化（步长h=0.001）

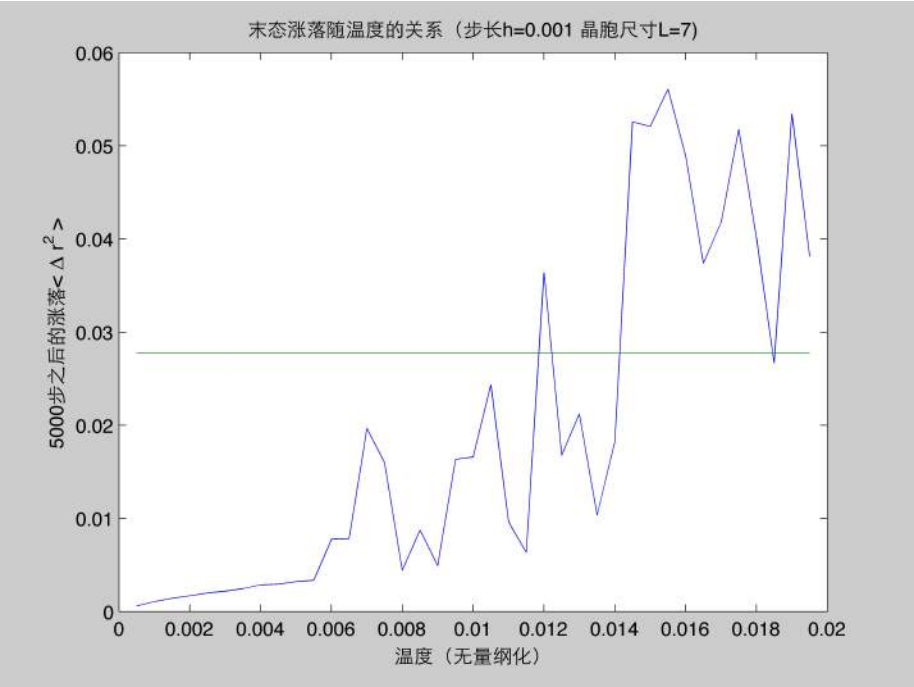


Figure 10: L=7时，不同温度下5000MD步之后的涨落（步长h=0.001）