计算物理第12次作业

吴远清 2018300001031

第一题

题目分析

题目要求中说明了, 应当应用周期性边界条件, 为此, 对于两个粒子间的距离, 我们定义了粒子坐标类`coord`, 并实现了考虑周期性边界条件计算粒子间距离的类内方法. 在`coord`类的基础上, 我们抽象出任意时刻的粒子坐标体系`configuration`类, 并在内部实现了对n个粒子的储存, 计算和输出. 最后, 我们抽象出`md`类, 实现分子动力学模拟的完整过程.

代码实现

首先定义了相关的类和方法:

```
#include "cmath"
#include <iostream>
#include <vector>
#include <random>
#include <ctime>
#include "fstream"

#define random ((double)rand()/RAND_MAX)
#define pi 3.141592653

using namespace std;

typedef struct dist
{
    double distant;
    double cost;
    double sint;
};

class coord
{
public:
    double x;
    double y;
    coord(double _x, double _y);
    coord operator+(coord p1);
    coord operator-(coord p1);
    dist distance(coord p1, double _l);
};

class configuration
{
public:
    int n_moles;
    double system_size, sigma, epsilon;
```

```
    configuration(int _n_moles, double _system_size, double _sigma, double
_epsilon);
    vector<coord> moles;
    void init_moles_position();
    void add_random_perturbation(double _quantity);
    void dump_data(ofstream& fp);
};

class md
{
public:
    int n_moles;
    double system_size, sigma, epsilon, tot_time, time_step, r_cutoff;
    vector<configuration> configurations;
    md(int _n_moles, double _system_size, double _sigma, double _epsilon, double
_tot_time, double _time_step, double _v0, double _r_cutoff);
    void update();
    void dump_data();
    void calcV(int start, int end, int status);
};
```

具体实现

```
#include "Q1.h"

coord::coord(double _x, double _y)
{
    x = _x;
    y = _y;
}

coord coord::operator+(coord p1)
{
    coord new_coord(x+p1.x, y+p1.y);
    return new_coord;
}

coord coord::operator-(coord p1)
{
    coord new_coord(x - p1.x, y - p1.y);
    return new_coord;
}

dist coord::distance(coord p1, double _l)
{
    double minR = INFINITY;
    int min_r_idx = 0;
    vector<dist> condition(9);
    for(auto i = -1; i < 2; i++)
    {
        for(auto j = -1; j < 2; j++)
        {
            condition[(i + 1) * 3 + (j + 1)].distant = sqrt(pow(p1.x -
(x+i*_l),2) + pow(p1.y - (y+j*_l),2));
            if (condition[(i + 1) * 3 + (j + 1)].distant < minR)
            {
                minR = condition[(i + 1) * 3 + (j + 1)].distant;
```

```
                condition[(i + 1) * 3 + (j + 1)].cost = ((x + i * _l - p1.x) /
condition[(i + 1) * 3 + (j + 1)].distant);
                condition[(i + 1) * 3 + (j + 1)].sint = ((y + j * _l - p1.y) /
condition[(i + 1) * 3 + (j + 1)].distant);
                min_r_idx = (i + 1) * 3 + (j + 1);
            }

        }
    }
    return condition[min_r_idx];
}

configuration::configuration(int _n_moles, double _system_size, double _sigma,
double _epsilon)
{
    n_moles = _n_moles;
    system_size = _system_size;
    sigma = _sigma;
    epsilon = _epsilon;
}

void configuration::init_moles_position()
{
    int n_moles_per_row = ((int)sqrt(n_moles) == sqrt(n_moles)) ?
(int)sqrt(n_moles) : (int)sqrt(n_moles) + 1;
    //cout << n_moles_per_row << " molecules per row." << endl;
    double distance_between_moles = system_size / (n_moles_per_row+1);
    //cout << "Distance between molecules:" << distance_between_moles << endl;
    for(auto i = 0; i < n_moles; i++)
    {
        moles.push_back(coord((i%n_moles_per_row+1)*distance_between_moles,
(int)(i/n_moles_per_row + 1)*distance_between_moles));
    }
    add_random_perturbation(0.5 * sigma);
}

void configuration::add_random_perturbation(double _quantity)
{
    for(auto i = 0; i < n_moles; i++)
    {
        double theta = 2 * pi * random;
        //cout << i << '\t' << _quantity * cos(theta) << '\t' << _quantity *
sin(theta) << endl;
        moles[i].x += _quantity * cos(theta);
        moles[i].y += _quantity * sin(theta);
    }
}

void configuration::dump_data(ofstream& fp)
{
    for (coord i : moles)
    {
        fp << i.x << "," << i.y << endl;
    }
}
```

```
md::md(int _n_moles, double _system_size, double _sigma, double _epsilon, double
_tot_time, double _time_step, double _v0, double _r_cutoff)
{
    srand(time(NULL));
    n_moles = _n_moles;
    system_size = _system_size;
    sigma = _sigma;
    epsilon = _epsilon;
    tot_time = _tot_time;
    time_step = _time_step;
    r_cutoff = _r_cutoff;
    configurations.push_back(configuration(n_moles, system_size, sigma,
epsilon));
    configurations[0].init_moles_position();
    configurations.push_back(configurations.back());
    //configurations[1].add_random_perturbation(_v0*time_step);
    for(auto i = 0; i < n_moles; i++)
    {
        configurations[1].moles[i].x += random * time_step;
    }
}

void md::update()
{
    vector<double> totfx(n_moles);
    vector<double> totfy(n_moles);
    configurations.push_back(configurations.back());
    for(auto i = 0; i < n_moles; i++)
    {
        totfx[i] = 0;
        totfy[i] = 0;
        for (auto j = 0; j < n_moles; j++)
        {
            if (i == j) continue;
            dist d =
configurations.back().moles[i].distance(configurations.back().moles[j],
system_size);
            if (d.distant>r_cutoff) continue;
            double f = 24 * (2 * pow(d.distant, -13) - pow(d.distant, -7));
            if(f > pow(time_step,-2))
            {
                cout << d.distant << ' ' << f << endl;
            }
            totfx[i] += f * d.cost;
            totfy[i] += f * d.sint;
        }
    }
}

void md::dump_data()
{
    ofstream fp("Q1.txt", ios::out);
    for (auto i = 0; i < configurations.size(); i+=20)
    {
        fp << "Iter:" << i << endl;
        configurations[i].dump_data(fp);
    }
}
```

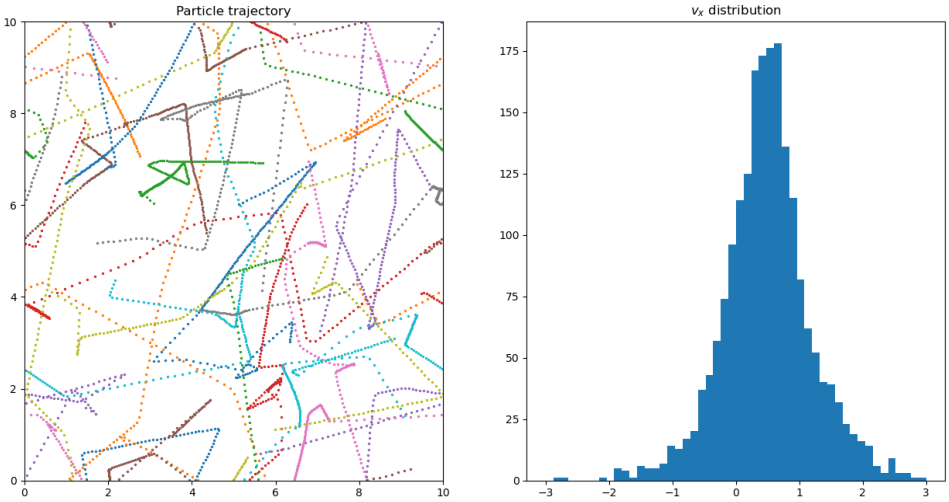
```
void md::calcV(int start, int end, int status)
{
    ofstream fp("Q1-V.txt", ios::app);
    for(auto i = 0; i < n_moles; i++)
    {
        double vx = 0, vy = 0;
        for(auto a1 = start + 1; a1 < end; a1++)
        {
            dist d = configurations[a1].moles[i].distance(configurations[a1-
1].moles[i], system_size);
            vx += d.distant * d.cost / time_step;
            vy += d.distant * d.sint / time_step;
        }
        vx = vx / (end - start);
        vy = vy / (end - start);
        double v = sqrt(pow(vx, 2) + pow(vy, 2));
        switch (status)
        {
        case 0:
            fp << v << endl;
            break;
        case 1:
            fp << vx << endl;
            break;
        case 2:
            fp << vy << endl;
            break;
        }
    }
}
```

计算主程序:

```
void Q1()
{
    const long NUM = 1000000;
    //cout << "q1 start!" << endl;
    md md1(20, 10., 1., 1., 10., 0.0001, 0,3.);
    for (auto i = 0; i < NUM; i++)
    {
        if(i%(long)(NUM/100)==0)
        {
            cout << i << "/" << NUM << '\r';
        }
        //cout << i << ' ';
        md1.update();
    }
    cout << endl;
    md1.dump_data();
    md1.calcV(0.2*NUM, 1*NUM, 1);
}
```

结果展示与讨论

如代码中所示, 我们使用0.0001的时间步长, 计算100000步, 并对最后100000步(后10%)的分子速度进行统计.结果如下:



所得结果如图所示, 从左侧的轨迹图可以看出, 分子的运动是非常杂乱无序的. 并且, 可以发现, 当两个分子非常接近的时候会产生非常突然的转向, 近似于刚性碰撞, 这也说明了LJ势在$r \to 1$的时候急剧增大的特征.

而右图的V_x分布可以看出, 与初始速度为0或者初始速度随机分布不同. 此时V_x的峰值位于0.5左右, 符合我们的预期.

第二题

题目分析

这题的计算过程与第一题一样, 我们将二维坐标的`coord`类改造成三维坐标的`vec`类, 并类似的构造相应的`configuration`类, 并构造用于计算的`melting`类.

代码实现

类与方法声明

```
#pragma once

#include <iostream>
#include <fstream>
#include <random>
#include <ctime>

#define random ((double)rand()/RAND_MAX)

using namespace std;

typedef struct dis
{
    double r;
    double cx;
    double cy;
```

```
    double cz;
};

class vec3d
{
public:
    double x, y, z;
    vec3d(double _x, double _y, double _z);
    dis distance(vec3d v1, double l);
    vec3d operator+(vec3d v1);
    vec3d operator-(vec3d v1);
    vec3d operator*(double a);
    double dot(vec3d v1);
};

class configuration
{
public:
    vector<vec3d> moles;
    int n_moles;
    double system_size;
    configuration(int _n_moles, double _system_size);
    void init_moles_position();
    void dump_data(ofstream& fp);
};

class melting
{
public:
    int n_moles;
    double system_size, sigma, epsilon, time_step, r_cutoff;
    vector < configuration > configurations;
    melting(int _n_moles, double _system_size, double _sigma, double _epsilon,
double _time_step, double _r_cutoff);
    void update(double factor);
    void dump_data(int interval);
    void calcT(int period);
};
```

类和方法的实现:

```
#include "Q2.h"

vec3d::vec3d(double _x, double _y, double _z)
{
    x = _x;
    y = _y;
    z = _z;
}

dis vec3d::distance(vec3d v1, double l)
{
    dis d;
    d.r = INFINITY;
    double r;
    for(auto ix = -1; ix < 2; ix++)
    {
```

```
        for (auto iy = -1; iy < 2; iy++)
        {
            for (auto iz = -1; iz < 2; iz++)
            {
                r = sqrt(pow(v1.x - (x + ix*l),2) + pow(v1.y - (y+iy*l),2) +
pow(v1.z - (z + iz*l),2));
                if (r < d.r)
                {
                    d.r = r;
                    d.cx = ((x + ix * l) - v1.x) / r;
                    d.cy = ((y + iy * l) - v1.y) / r;
                    d.cz = ((z + iz * l) - v1.z) / r;
                }
            }
        }
    }
    return d;
}

vec3d vec3d::operator+(vec3d v1)
{
    vec3d v_new(x+v1.x, y+v1.y, z+v1.z);
    return v_new;
}

vec3d vec3d::operator-(vec3d v1)
{
    vec3d v_new(x - v1.x, y - v1.y, z - v1.z);
    return v_new;
}

vec3d vec3d::operator*(double a)
{
    vec3d v_new(x*a, y*a, z*a);
    return v_new;
}

double vec3d::dot(vec3d v1)
{
    return (x * v1.x + y * v1.y + z * v1.z);
}


configuration::configuration(int _n_moles, double _system_size)
{
    n_moles = _n_moles;
    system_size = _system_size;
}

void configuration::init_moles_position()
{
    int n_moles_per_row = ((int)pow(n_moles, 1. / 3) == pow(n_moles, 1. / 3)) ?
(int)pow(n_moles, 1. / 3) : (int)pow(n_moles, 1. / 3) + 1;
    cout << n_moles << " " << pow(n_moles, 1. / 3) << endl;
    cout << n_moles_per_row << " molecules per row." << endl;
    double distance_between_moles = system_size / (n_moles_per_row + 1);
    cout << "Distance between molecules:" << distance_between_moles << endl;
    for(auto i = 0; i < n_moles_per_row; i++)
```

```
    {
        for (auto j = 0; j < n_moles_per_row; j++)
        {
            for (auto k = 0; k < n_moles_per_row; k++)
            {
                moles.push_back(vec3d((i+1)*distance_between_moles,
(j+1)*distance_between_moles, (k+1)*distance_between_moles));
            }
        }
    }
}

void configuration::dump_data(ofstream& fp)
{
    for (vec3d i: moles)
    {
        fp << i.x << "," << i.y << "," << i.z << endl;
    }
}

melting::melting(int _n_moles, double _system_size, double _sigma, double
_epsilon, double _time_step, double _r_cutoff)
{
    n_moles = _n_moles;
    system_size = _system_size;
    sigma = _sigma;
    epsilon = _epsilon;
    time_step = _time_step;
    r_cutoff = _r_cutoff;
    configurations.push_back(configuration(n_moles,system_size));
    configurations.back().init_moles_position();
    configurations.push_back(configurations.back());
}

void melting::update(double factor)
{
    //cout << n_moles << endl;
    vector<vec3d> totF;
    configurations.push_back(configurations.back());
    for (auto i = 0; i < n_moles; i++)
    {
        //f(i==0) cout << configurations.back().moles[i].x << " " <<
configurations.back().moles[i].y << " " << configurations.back().moles[i].z <<
endl;
        totF.push_back(vec3d(0, 0, 0));
        for (auto j = 0; j < n_moles; j++)
        {
            if (i == j) continue;
            //cout << i << " " << j << endl;
            dis d =
configurations.back().moles[i].distance(configurations.back().moles[j],
system_size);
            //cout << i << " " << j << endl;
            //if (i == 0 && j == 2) cout << configurations.back().moles[i].z <<
" " << d.r*d.cz << endl;
            if (d.r > r_cutoff) continue;
            double f = 24 * (2 * pow(d.r, -13.) - pow(d.r, -7.));
            if (f > pow(time_step, -2))
```

```
            {
                cout << d.r << ' ' << f << endl;
            }
            totF[i] = totF[i] + vec3d(d.cx, d.cy, d.cz)*f;
        }
    }
    //cout << totF[13].x << " " << totF[13].y << " " << totF[13].z << endl;
    for (auto i = 0; i < n_moles; i++)
    {
        configurations.back().moles[i] = configurations.back().moles[i] +
(configurations[configurations.size() - 2].moles[i] -
configurations[configurations.size() - 3].moles[i])*factor +
totF[i]*time_step*time_step;
        if (configurations.back().moles[i].x > system_size)
        {
            configurations.back().moles[i].x = configurations.back().moles[i].x
- floor(configurations.back().moles[i].x / system_size) * system_size;
        }
        if (configurations.back().moles[i].x < 0)
        {
            configurations.back().moles[i].x =
(floor(abs(configurations.back().moles[i].x) / system_size) + 1) * system_size +
configurations.back().moles[i].x;
        }
        if (configurations.back().moles[i].y > system_size)
        {
            configurations.back().moles[i].y = configurations.back().moles[i].y
- floor(configurations.back().moles[i].y / system_size) * system_size;
        }
        if (configurations.back().moles[i].y < 0)
        {
            configurations.back().moles[i].y =
(floor(abs(configurations.back().moles[i].y) / system_size) + 1) * system_size +
configurations.back().moles[i].y;
        }
        if (configurations.back().moles[i].z > system_size)
        {
            configurations.back().moles[i].z = configurations.back().moles[i].z
- floor(configurations.back().moles[i].z / system_size) * system_size;
        }
        if (configurations.back().moles[i].z < 0)
        {
            configurations.back().moles[i].z =
(floor(abs(configurations.back().moles[i].z) / system_size) + 1) * system_size +
configurations.back().moles[i].z;
        }
    }
}

void melting::dump_data(int interval)
{
    ofstream fp("Q2.txt", ios::out);
    for(auto i = 0; i < configurations.size(); i+=interval)
    {
        cout << i << "/" << configurations.size() << "\r";
        fp << "Iter:" << i << endl;
        configurations[i].dump_data(fp);
    }
```

```
    cout << endl;
}

void melting::calcT(int period)
{
    vector<double> TList, RList;
    for(auto i = 1; i < configurations.size(); i++)
    {
        double T = 0, rMean = 0;
        for (auto n = 0; n < n_moles; n++)
        {
            if (n != 0)
            {
                dis d =
configurations[i].moles[0].distance(configurations[i].moles[n], system_size);
                rMean += d.r/(n_moles - 1);
            }
            vec3d V(0, 0, 0);
            V = configurations[i].moles[n] - configurations[i-1].moles[n];
            T += V.dot(V);
        }
        if(i%(int)(configurations.size()/100)==0)
        cout << i << "/" << configurations.size() << '\r';
        TList.push_back(T);
        RList.push_back(rMean);
    }
    cout << endl;
    ofstream fp("Q2-T.txt", ios::out);
    for(auto i: RList)
    {
        fp << i << endl;
    }
    fp.close();
}
```

实现计算

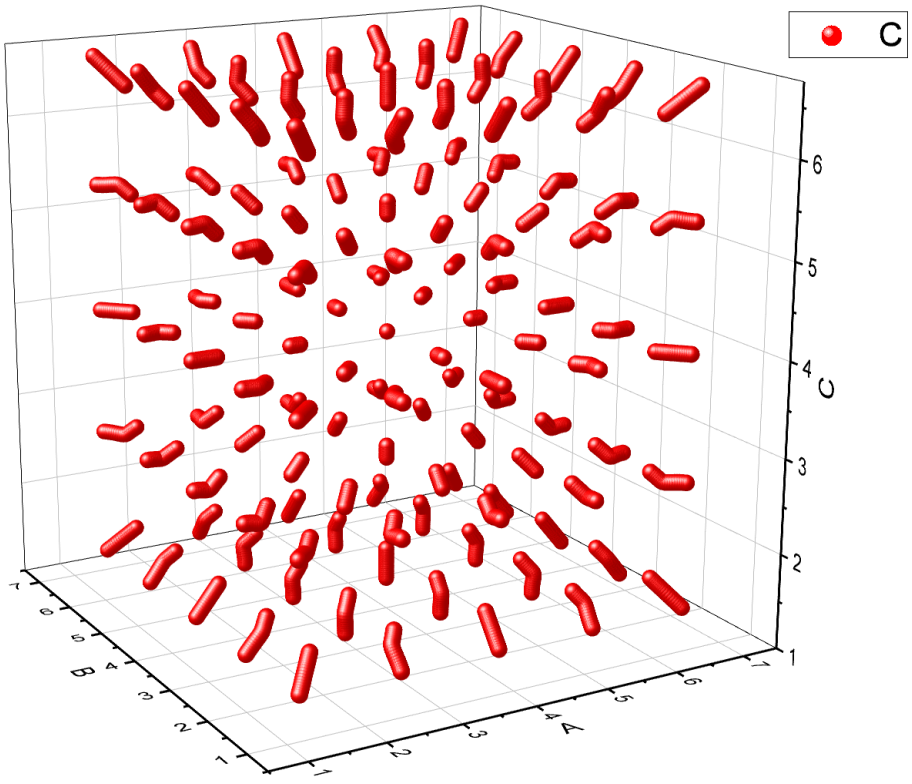
```
void Q2()
{
    const long NUM = 1000000;
    double T = 0, factor = 0.8;
    melting m1(125, 8., 1., 1., 0.0001, 3.);
    for (auto i = 0; i < NUM; i++)
    {
        if (i % (long)(NUM / 500) == 0)
        {
            cout << i << "/" << NUM << '\r';
            m1.update(factor);
            if (factor < 1.1) factor += 0.001;
            //m1.dump_data(1.);
        }
        else
        {
            m1.update(1.0);
        }
    }
    cout << endl;
```

```
    m1.dump_data(100);
    m1.calcT(20);
}
```

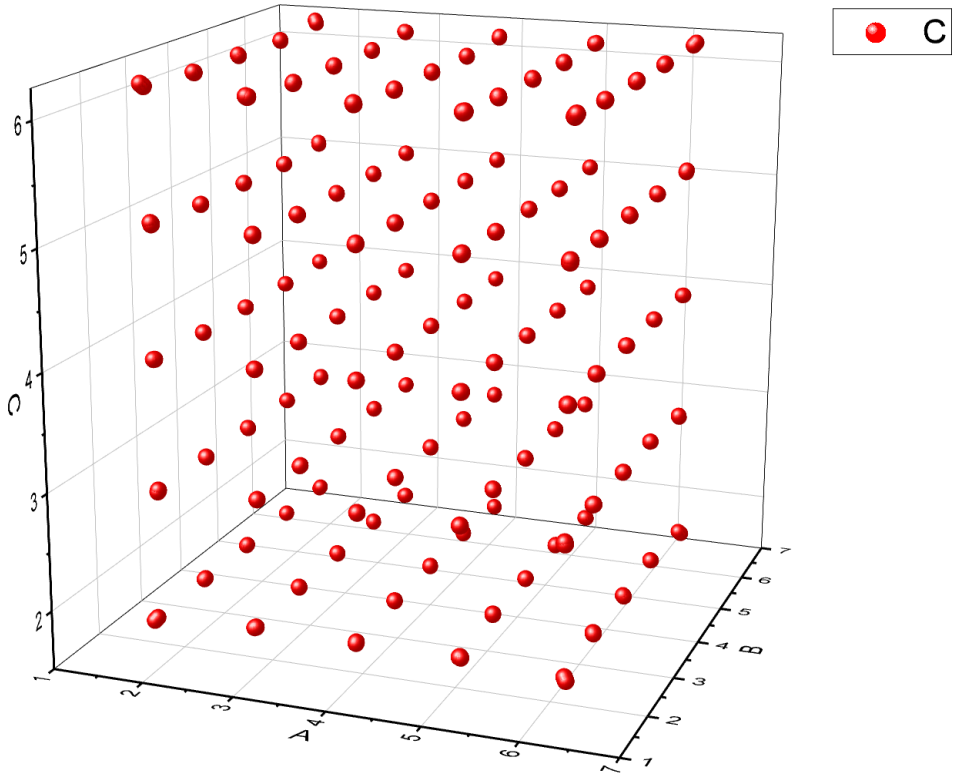
结果分析

如代码所示, 我们使用$5 \times 5 \times 5$的三维粒子构型来进行模拟, 时间步长为$0.0001s$,模拟1000000步. 并且逐步增加体系温度, 来实现模拟体系融化的过程.

首先来分析计算得到的稳定构型. 对于前大约20000代, 体系处于缓慢平衡过程,可以看到, 由于初始时我们设置的分子间距较近, 分子被排斥向外扩张.



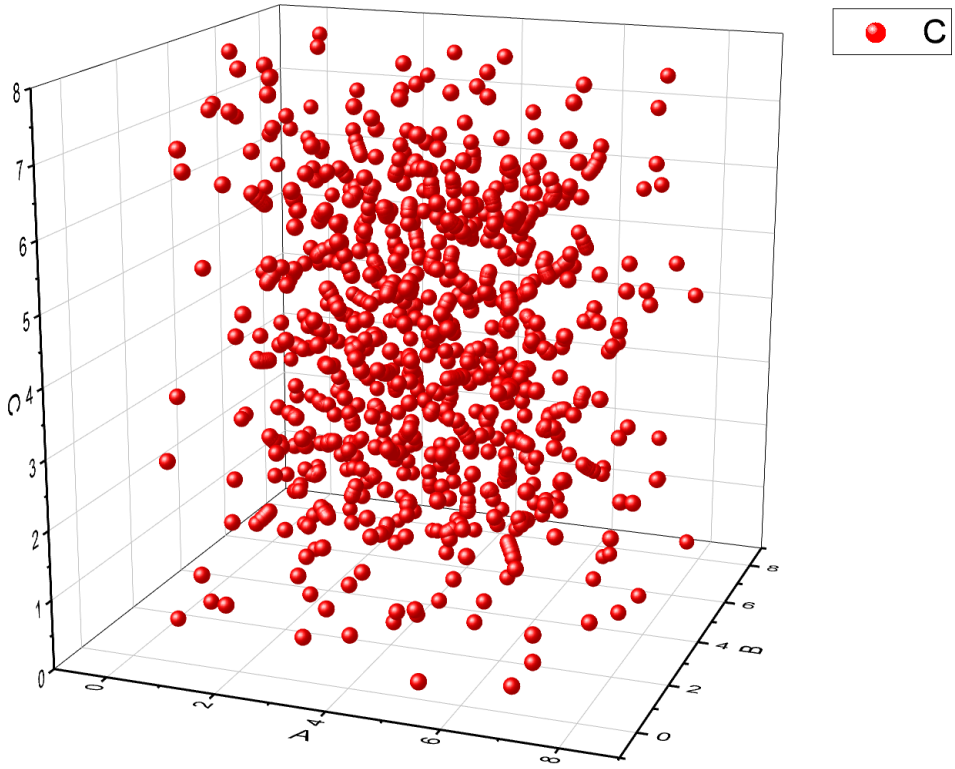
稳定一段时间后,达到平衡, 体系结构如下(同样画出20000步内的所有位置):



可以看出, 此状态下分子基本处于稳定位置, 不再振动.

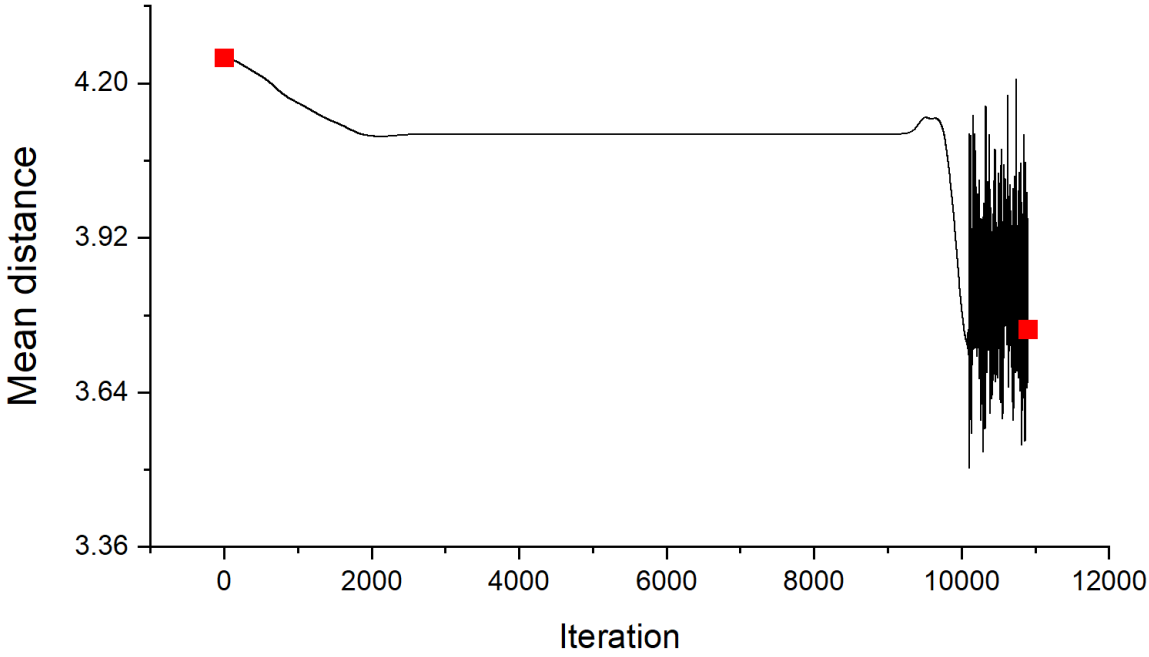
也证明了, 简单作用势下, 粒子的分布确实是面心立方结构.

之后,我们提升体系温度,增加粒子动能,发现粒子开始无规则运动(画出最后20000步的所有粒子位置):



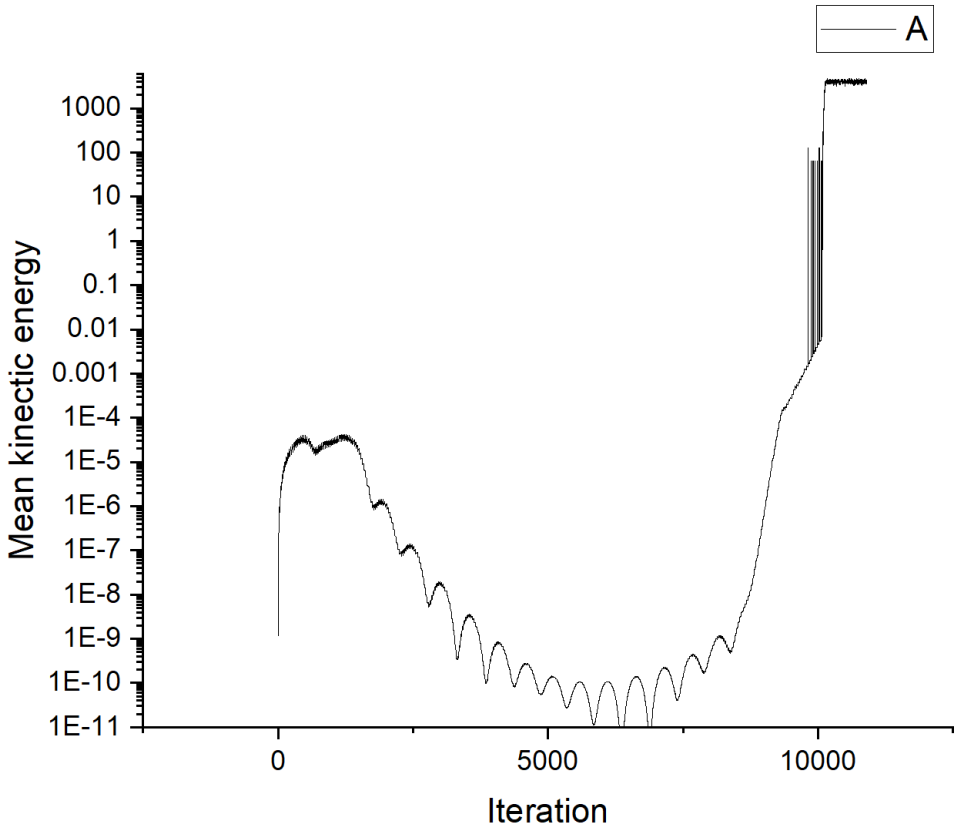
我们认为, 此时, 体系的已经完成了从固体熔化的相变过程.

下面我们来研究分子间的间距, 分子间的平均间距随迭代次数的变化规律如图所示:



可见, 最初, 粒子线运动到平衡位置, 并长时间保持在平衡位置附近. 而随着体系温度不断上升, 分子间平均间距极速无规律变化(熔融态).

最后, 我们讨论分子间的



可以判断,融化的温度应该在$100 \sim 1000 \frac{\epsilon}{k_B}$左右.