КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ФИЗИКИ
Кафедра вычислительной физики и моделирования физических процессов

Д.Т. ЯРУЛЛИН, Б.Н. ГАЛИМЗЯНОВ

ПРОВЕДЕНИЕ КЛАСТЕРНОГО АНАЛИЗА НА ОСНОВЕ РЕЗУЛЬТАТОВ МОЛЕКУЛЯРНО-ДИНАМИЧЕСКИХ РАСЧЕТОВ

Учебно-методическое пособие

Казань – 2022

УДК 538.9; 536.3; 53.03
ББК 22.311

Издается по решению Ученого совета Института физики
Казанского федерального университета

Рецензенты:
кандидат физико-математических наук,
доцент кафедры вычислительной физики КФУ **Р.М. Хуснутдинов**;
кандидат физико-математических наук,
доцент кафедры вычислительной физики КФУ **И.И. Файрушин**

Яруллин Д.Т., Галимзянов Б.Н.
Проведение кластерного анализа на основе результатов молекулярно-динамических расчетов /
Д.Т. Яруллин, Б.Н. Галимзянов – Казань: Казан. ун-т, 2022. – 27 с.

В настоящем учебно-методическом пособии представлен теоретический и практический материал по проведению кластерного и структурного анализа результатов молекулярно-динамических расчетов. Учебное пособие предназначено для студентов и аспирантов физических специальностей высших учебных заведений при изучении дисциплин, связанных с компьютерным моделированием. Пособие составлено с целью повышения эффективности организации самостоятельной работы и аудиторных занятий студентов очного отделения.

Методическое пособие выполнено при поддержке фонда развития теоретической физики и математики «БАЗИС» (Проект No 20-1-2-38-3).

ОГЛАВЛЕНИЕ

Предисловие

В настоящем учебно-методическом пособии представлен теоретический и практический материал по проведению кластерного и структурного анализа результатов молекулярно-динамических расчетов. Пособие состоит из двух частей. В первой части приводится краткий теоретический материал с инструкциями и справочной информацией по работе с вычислительным пакетом LAMMPS. Представленные инструкции позволяют выполнить численное моделирование процесса кристаллизации модельной системы Леннарда-Джонса.

Во второй части представлены коды на языке C# для программной реализации кластерного и структурного анализа результатов молекулярно-динамических расчетов.

Учебное пособие предназначено для студентов и аспирантов физических специальностей высших учебных заведений при изучении дисциплин, связанных с компьютерным моделированием. Пособие составлено с целью повышения эффективности организации самостоятельной работы и аудиторных занятий студентов очного отделения.

1. МОЛЕКУЛЯРНО-ДИНАМИЧЕСКОЕ МОДЕЛИРОВАНИЕ ПРОЦЕССА КРИСТАЛЛИЗАЦИИ В ВЫЧИСЛИТЕЛЬНОМ ПАКЕТЕ LAMMPS

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) - один из наиболее известных вычислительных пакетов для выполнения молекулярно-динамического моделирования. Широкая известность LAMMPS обусловлена тем, что данный вычислительный пакет включает в себя большую обновляемую библиотеку потенциалов межчастичного взаимодействия, позволяющих моделировать поведение различных систем: металлов, полупроводников, биомолекул, полимерных систем и др. Важным достоинством LAMMPS является возможность распараллеливания расчетов в соответствии с MPI (Message Passing Interface), что позволяет многократно повысить производительность вычислений на многопроцессорных компьютерах, вычислительных кластерах и суперкомпьютерах. Возможна компиляция исходных кодов под различные операционные системы (Linux, OS X, Windows), а также имеется большое количество различных опций компиляции, которые позволяют создавать исполняемый файл под конкретную архитектуру процессора и архитектуру графического процессора. Учебник с описанием всех возможностей данного пакета опубликован на официальном сайте разработчиков по ссылке: https://www.lammps.org/tutorials.html. Дополнительная полезная информация об основах проведения молекулярно-динамических расчетов с помощью вычислительного пакета LAMMPS представлена в работах [1-4].

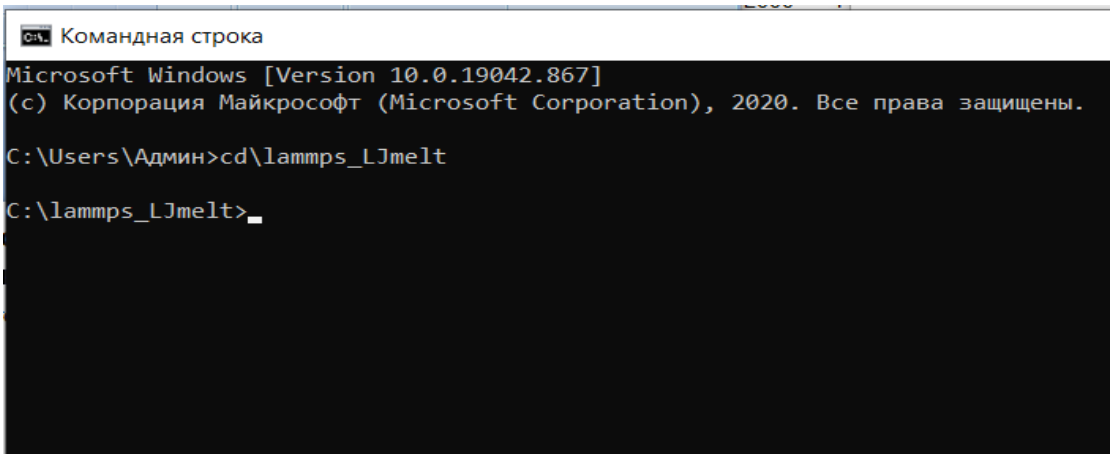
1.1 Запуск расчетов

Для запуска расчетов в вычислительном пакете LAMMPS понадобятся следующие файлы:

- Исполняемый файл lmp_serial.exe (в случае однопоточных вычислений)
- script-файл, содержащий детали моделирования (примеры находятся по следующему пути: корневая папка LAMMPS\Examples)

Данные файлы должны находиться в одной папке. LAMMPS не имеет графического интерфейса (только командный режим), поэтому запуск расчетов осуществляется через командную строку. Командную строку можно запустить через диалоговое окно «Выполнить» - для этого необходимо зажать сочетание клавиш «WIN + R», в открывшемся окне ввести *cmd*. Альтернативный способ открытия командной строки - в строке поиска (находится рядом с кнопкой «Пуск») ввести *cmd*.

В командной строке для выбора директории следует набрать команду *cd* и указать путь к папке, где располагаются script-файл и исполняемый файл. Далее необходимо набрать *lmp_serial.exe* (название исполняемого файла) и *<in.melt* (символ < + название скрипт-файла с символом) и нажать кнопку Enter. Пример запуска расчетов с помощью командной строки представлен на рисунке 1.



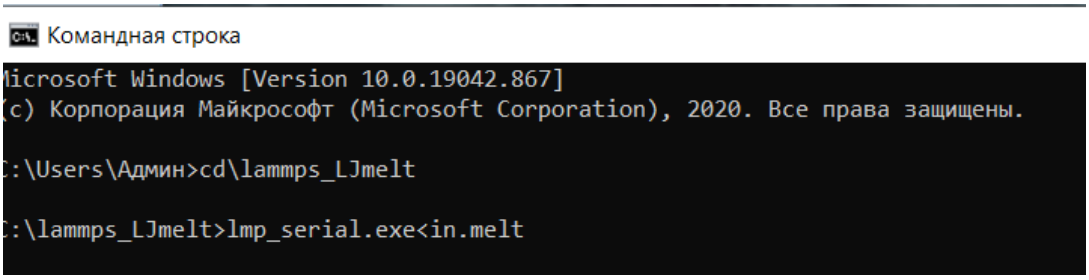


Рис. 1. Пример запуска расчетов с помощью командной строки. «lamps_LJmelt» – название папки, в которой располагаются script-файл и исполняемый файл «lmp_serial.exe». «in.melt» – название script-файла

1.2 Скрипт-файлы для моделирования кристаллизации

Далее будут представлены скрипт-файлы, позволяющие смоделировать кристаллизацию переохлажденной однокомпонентной системы Леннарда-Джонса в LAMMPS.

Этап 1. Получение исходного образца

```
# 3d Lennard-Jones melt

units       lj
atom_style atomic

lattice          fcc 0.8442
region           box block 0 15 0 15 0 15
create_box 1 box
create_atoms     1 box
mass        1 1.0

velocity   all create 2.5 87287

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

timestep         0.005
fix 1 all npt temp 2.5 2.5 $(100*dt) iso 2.0 2.0 $(1000*dt)

thermo_style custom step temp pe press

restart          10000 restart_lj_T2.5_PT.equil

thermo           10
run              100000
```

Скрипт-файл: *in_equi.melt*

Данный скрипт-файл позволяет получить систему 13500 частиц, взаимодействующих на основе потенциала Леннарда-Джонса. Температура и давление системы составляют 2.5 ϵ/k_B и $2.0\epsilon/\sigma^3$, соответственно. В ходе выполнения расчетов каждые 10’000 временных шагов будет создаваться конфигурационный рестарт-файл, содержащий координаты всех частиц в исходном жидком образце. Таким образом, в случае успешного завершения расчетов, состоящих из 100’000 временных шагов, в папке будет создано 10 рестарт-файлов (как показано на рисунке 2).

Наличие рестарт-файлов необходимо для выполнения независимых симуляций и проведения статистической обработки получаемых результатов.

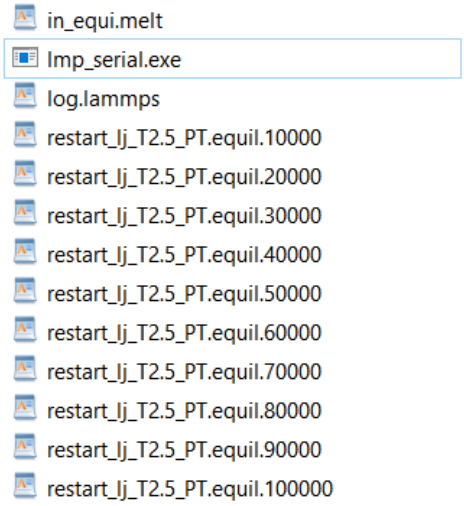


Рис. 2. Результат выполнения скрипт-файла *in_equi.melt*

Отчет о выполненных расчетах помещается в файл «log.lammps». В этот же файл записывается информация об ошибках, в случае их возникновения.

Следующий рестарт-файл «in_cool.melt» задает охлаждение жидких образцов, полученных на начальном этапе.

Этап 2. Охлаждение исследуемого образца

```
units       lj
atom_style atomic

read_restart restart_lj_T2.5_PT.equil.10000

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

timestep        0.005
fix 1 all npt temp 2.5 0.5 $(100*dt) iso 2.0 2.0 $(1000*dt)

thermo_style custom step temp pe press density

restart         10000 restart_S1_lj_T0.5_PT.equil

thermo          10
run             10000
```

Скрипт-файл: *in_cool.melt*

В результате выполнения данного скрипт-файла будет получен рестарт-файл, содержащий информацию о положении частиц после охлаждения до температуры 0.5 ϵ/k_B при постоянном давлении 2.0 ϵ/σ^3. Обратите внимание, что в случае, когда в «read_restart» указывается только название рестарт-файла, данный файл должен находиться в одной папке с исполняемым и скрипт файлами. В некоторых случаях более удобно указывать полный путь к файлу, содержащий корневой каталог и все сопутствующие папки. Полный путь к файлу может указываться и в строках «restart», «dump». Пример полного пути к файлу: «C:\Lammps\ restart_lj_T2.5_PT.equil.10000».

Этап 3. Получение dump-файлов кристаллизующейся системы

```
units       lj
atom_style atomic

read_restart restart_lj_T0.5_PT.equil.20000

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

timestep        0.01
fix 1 all npt temp 0.5 0.5 $(100*dt) iso 2.0 2.0 $(1000*dt)

thermo_style custom step temp pe press density

dump            1 all custom 1 S1_dump_ljmelt_T0.5_file.txt id
type x y z

thermo          10
run             10000
```

Скрипт-файл: *in_dump.melt*

После выполнения скрипт-файла in_dump.melt будет создан dump-файл «S1_dump_ljmelt_T0.5_file.txt». Содержание дамп-файла представлено на рисунке 3. Впоследствии на основе dump-файла производится идентификация наличия упорядоченных структур в системе.

```
ITEM: TIMESTEP
20000
ITEM: NUMBER OF ATOMS
13500
ITEM: BOX BOUNDS pp pp pp
3.2962753333486106e-01 2.4864315337402701e+01
3.2962753333486106e-01 2.4864315337402701e+01
3.2962753333486106e-01 2.4864315337402701e+01
ITEM: ATOMS id type x y z
3208 1 1.15771 0.692858 0.657684
3096 1 3.04806 1.66185 0.951313
3800 1 2.19125 0.739244 0.947275
9805 1 2.09108 1.67647 1.35297
5164 1 3.01862 0.934522 1.68315
7254 1 3.55156 1.74829 1.85332
2278 1 3.84537 0.849668 0.971744
7253 1 4.65815 1.61053 1.75003
2923 1 4.67935 0.468645 0.605677
7317 1 4.38995 0.523764 1.6931
6727 1 5.12433 1.37458 0.811539
3580 1 5.53384 0.905423 1.72646
7955 1 6.21854 1.26853 1.00934
273 1 5.64447 0.697759 0.337093
10051 1 7.9611 1.54979 0.524975
3528 1 7.05309 0.473805 0.936071
8357 1 7.50309 1.27069 1.50591
562 1 8.37268 0.617107 1.04542
5130 1 9.28826 0.994668 1.23273
4423 1 10.1971 1.41733 1.27447
9227 1 9.71976 1.55227 0.372786
4156 1 9.75769 0.419611 0.333261
1659 1 11.2766 0.964279 0.960084
837 1 11.9531 1.71099 0.897925
2636 1 12.0469 1.21548 1.85281
6685 1 12.4238 0.579092 1.02855
...
```

Рис. 3. Структура dump-файла

Как видно из рисунка, в dump-файле содержится детальная информация об исследуемой ячейке моделирования: количество выполненных шагов моделирования, количестве частиц в системе, их координаты, параметры исследуемой ячейки моделирования.

1.3 Идентификация кристаллических структур на основе результатов молекулярно-динамических расчетов

Одним из наиболее часто применяемых методов при проведении кластерного и структурного анализа является метод, основанный на расчете параметров локального ориентационного порядка:

$$q_m^l(i) = \frac{1}{N_b(i)} \sum_{j=1}^{N_b(i)} Y_m^l(r_{ij}). \qquad (1)$$

Здесь $N_b(i)$ - количество ближайших соседей у частицы i, l - целочисленный параметр, m - параметр, принимающий значения от $m = -l$ до $m = +l$. $Y_m^l(\boldsymbol{r}) = Y_m^l\big(\theta(\boldsymbol{r}), \boldsymbol{\phi}(\boldsymbol{r})\big) -$ сферические гармоники, $\theta(\boldsymbol{r})$ и $\boldsymbol{\phi}(\boldsymbol{r}) -$ зенитный и азимутальный углы, соответственно.

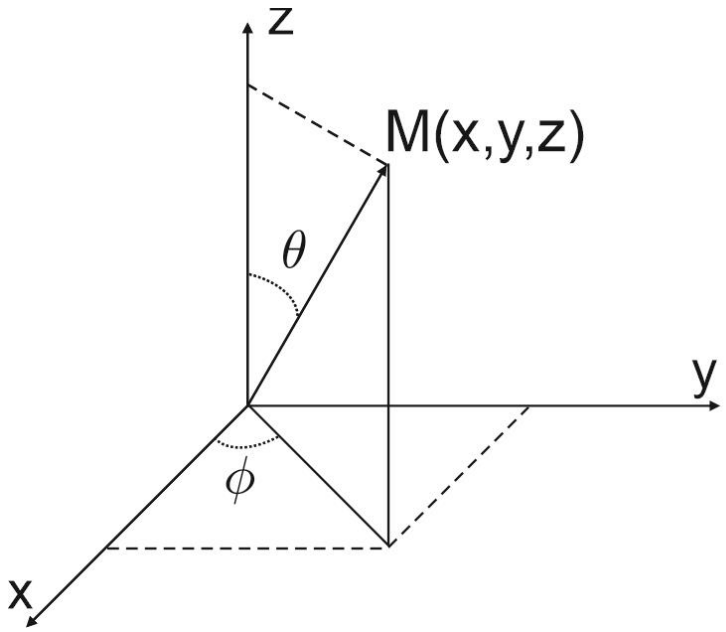


Рис. 4. Сферические координаты θ и $\boldsymbol{\phi}$ для произвольной точки M

Связь между сферическими координатами θ и $\boldsymbol{\phi}$ и декартовыми координатами x, y, z описывается следующими соотношениями:

$$\begin{aligned} \boldsymbol{\phi} &= \boldsymbol{arctg}\frac{\boldsymbol{y}}{\boldsymbol{x}} \\ \boldsymbol{\theta} &= \mathbf{arccos}\frac{\boldsymbol{z}}{\sqrt{\boldsymbol{x^2 + y^2 + z^2}}} \end{aligned} \qquad (\mathbf{2})$$

Расчет сферических гармоник производится на основе следующего выражения:

$$Y_m^l(\theta,\phi) = \sqrt{\frac{2m+1}{4\pi}\frac{(m-l)!}{(m+l)!}}P_m^l(cos\theta)e^{il\phi}, \quad (3)$$

где P_m^l - присоединенные полиномы Лежандра.

Идентификация частиц, участвующих в формировании кристаллических структур, осуществляется на основе значения скалярного произведения усредненных параметров локального ориентационного порядка q_m^l для частиц i и j:

$$S_{ij} = \sum_{m=-6}^{6} q_{6m}(i)q_{6m}^*(j) \quad (4)$$

Частицы i и j являются «*соединенными*» в случае, если $S_{ij} > 0.5$. Частица считается *solidlike,* то есть участвующей в формировании кристаллических структур, если в её окружении имеется как минимум 6 частиц с выполняющимся условием (4) [10,11].

2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ОБРАБОТКИ РЕЗУЛЬТАТОВ МД РАСЧЕТОВ

В данном параграфе представлена программная реализация основных процедур, необходимых при проведении кластерного анализа результатов молекулярно-динамических расчетов. Представленные программные коды написаны на языке C#.

2.1 Подготовка скрипт-файлов для проведения кластерного анализа

Для задания условий проведения кластерного и структурного анализа результатов молекулярно-динамических расчетов, как правило, применяется скрипт-файл. На рисунке далее представлен пример скрипт-файла. В представленном скрипт-файле содержится информация о количестве частиц в исследуемой ячейке моделирования, количество временных итераций, содержащихся в dump-файле, путь к dump-файлу, а также задан шаг, с которого начинается проведение кластерного и структурного анализа. Содержание скрипт-файла может изменяться в зависимости от специфики проводимого кластерного анализа.

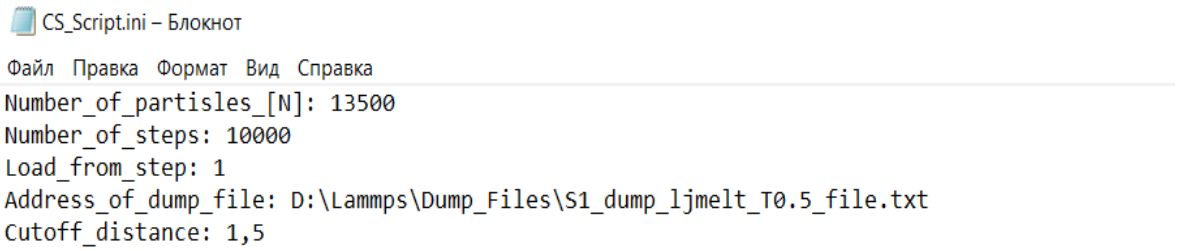


Рис. 5. Пример скрипт-файла «*CS_Script.ini*»

Считывание скрипт-файла реализуется при помощи кода, представленного далее.

```
#region Load_Script
            Console.Write("Load parameters from script file: ");
            FileStream fscript = new FileStream("CS_Script.ini", FileMode.Open);
```

```
                StreamReader rscript = new StreamReader(fscript);
      st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0, idx + 1);
                _ddl.N = Convert.ToInt32(sts);
                st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
                _ddl.Step = Convert.ToInt32(sts);
                st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
                fstep = Convert.ToInt32(sts);
                st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
               st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
                pos_address = sts;
                st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
                _ddl.cutoff = Convert.ToSingle(sts);
                st = rscript.ReadLine(); idx = st.IndexOf(" "); sts = st.Remove(0,
idx + 1);
                Console.WriteLine("Completed");
                Console.WriteLine("Start the cluster analysis:");
                Console.WriteLine("--------------------------");
        #endregion
```

В представленном коде для чтения скрипт-файла применяются классы «FileStream» и «StreamReader», входящие в состав пространства имен «System.IO». (Детальная информация о пространстве имён «System.IO» располагается по адресу: https://docs.microsoft.com/ru-ru/dotnet/api/system.io?view=net-6.0). Метод ReadLine () считывает строку из указанного в FileStream входного потока (файла). Извлечение интересующих численных значений происходит при помощи метода IndexOf(). Обратите внимание, что файл «CS_Script.ini» должен располагаться в той же папке, что и исполняемый файл программы кластерного анализа.

2.2 Считывание координат частиц из dump-файлов

На следующем этапе необходимо считать координаты частиц, содержащихся в dump-файле. Представленный далее программный код позволяет решить поставленную задачу.

```
FileStream fsp = new FileStream(pos_address, FileMode.Open);
    StreamReader strp = new StreamReader(fsp);
    for (int step = 1; step <= _ddl.Step; step++)
    {
        _ddl.VMDid_count++;
        _ddl.currentStep++;
        #region load_positions
        st = strp.ReadLine();
        st = strp.ReadLine();
        st = strp.ReadLine();
        st = strp.ReadLine();
        st = strp.ReadLine();
        st = strp.ReadLine(); sts = st; idx = sts.IndexOf(" ");
        st1 = st.Substring(0, idx);
        sts = st.Remove(0, idx + 1); dfg = sts.Replace(".", ",");
        _ddl.box_x = Convert.ToSingle(dfg);
        // Переменные box_x, box_y, box_z несут информацию о параметрах
ячейки моделирования
        st = strp.ReadLine(); sts = st; idx = sts.IndexOf(" ");
        st1 = st.Substring(0, idx);
        sts = st.Remove(0, idx + 1); dfg = sts.Replace(".", ",");
        _ddl.box_y = Convert.ToSingle(dfg);
        st = strp.ReadLine(); sts = st; idx = sts.IndexOf(" ");
        st1 = st.Substring(0, idx);
        sts = st.Remove(0, idx + 1); dfg = sts.Replace(".", ",");
        _ddl.box_z = Convert.ToSingle(dfg);
        st = strp.ReadLine();
        for (int i = 0; i < _ddl.N; i++)
        {
            st = strp.ReadLine(); sts = st; idx = sts.IndexOf(" ");
            st1 = st.Substring(0, idx);
            sts = st.Remove(0, idx + 1); idx = sts.IndexOf(" ");
            dfg = sts.Substring(0, idx); dfg = dfg.Replace(".", ",");
            st1 = dfg;
```

```
            st = sts.Remove(0, idx + 1); idx = st.IndexOf(" ");
            dfg = st.Substring(0, idx); dfg = dfg.Replace(".", ",");
            _ddl.pos[i, 0] = Convert.ToSingle(dfg);
            // В массив pos[][] записываются координаты частиц
            // Столбец pos[][0] - содержит координаты частиц по оси X,
           //pos[][1] - по оси Y, pos[][2] - по оси Z
            sts = st.Remove(0, idx + 1); idx = sts.IndexOf(" ");
            dfg = sts.Substring(0, idx); dfg = dfg.Replace(".", ",");
            _ddl.pos[i, 1] = Convert.ToSingle(dfg);
            st = sts.Remove(0, idx + 1); idx = st.IndexOf(" ");
            st = st.Replace(".", ",");
            _ddl.pos[i, 2] = Convert.ToSingle(st);
        }
        #endregion
        Console.WriteLine("Step " + step.ToString() + ": ");
```

Для информирования пользователя о ходе выполнения программы реализован вывод номера шага работы программы через метод Console.WriteLine(). Двумерный массив _ddl.pos[][] содержит информацию о координатах всех частиц системы. Схематическое представление массива _ddl.pos[][] представлено на рисунке 6.

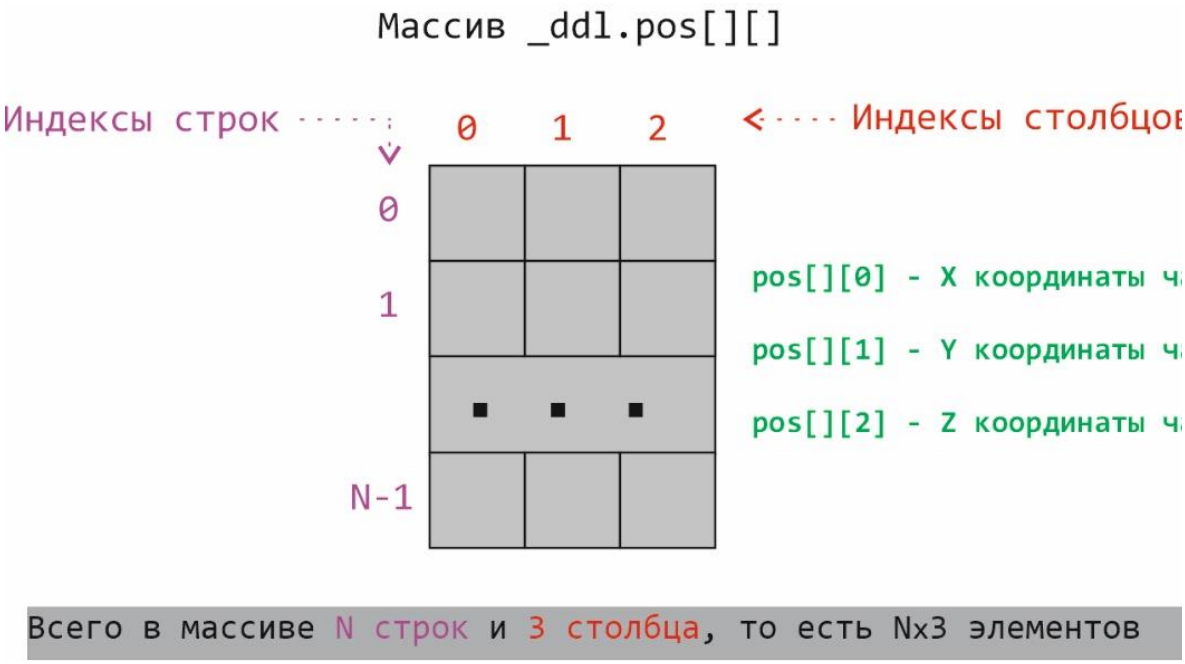


Рис. 6 Схематическое представление массива _ddl.pos[][]

Обратите внимание на то, что в языке *C#* индексация элементов массива начинается с 0.

2.3 Определение «частиц-соседей»

Для того, чтобы рассчитать параметр ориентационного порядка, необходимо составить список «ближайших соседей» каждой частицы исследуемой системы. Две частицы считаются «соседями», если они лежат на расстоянии $1.2r_0$, где r_0 определяется минимумом потенциала Леннарда-Джонса. В более общем случае r_0 может определяться положением первого пика функции радиального распределения $g(r)$. При таком определении все частицы первой координационной сферы будут считаться «ближайшими соседями».

Реализация поиска «частиц-соседей» может быть выполнена при помощи процедуры **genPair()**, представленной далее.

```
void genPair()
        {
          for (int k=0; k<num; k++) { nnum[k] = 0; }
          // Элемент массива nnum[i] определяет количество "соседей" у частицы i
            for (int i = 0; i < num; i++)
            {
                for (int j = 0; j < num; j++)
                {
                    if (i == j) { j++; }

                double dr2, drx, dry, drz;
                drx = q[i].x - q[j].x;
                dry = q[i].y - q[j].y;
                drz = q[i].z - q[j].z;
                #region block_distans

                _ddl.Sij1 = (float)(drx / _ddl.box_x);
                          // Sij1 определяет расстояние между частицами i и j;
                _ddl.Sij2 = (float)(dry / _ddl.box_y);
                _ddl.Sij3 = (float)(drz / _ddl.box_z);
                if (Math.Abs(_ddl.Sij1) > 0.5) { _ddl.Sij1 = _ddl.Sij1 -
Math.Sign(_ddl.Sij1); }
      // Определение расстояний между частицами с учетом граничных условий
                if (Math.Abs(_ddl.Sij2) > 0.5) { _ddl.Sij2 = _ddl.Sij2 -
Math.Sign(_ddl.Sij2); }
                if (Math.Abs(_ddl.Sij3) > 0.5) { _ddl.Sij3 = _ddl.Sij3 -
Math.Sign(_ddl.Sij3); }
                dr2 = _ddl.box_x * _ddl.Sij1 * _ddl.box_x * _ddl.Sij1
                    + _ddl.box_y * _ddl.Sij2 * _ddl.box_y * _ddl.Sij2
```

```
                        + _ddl.box_z * _ddl.Sij3 * _ddl.box_z * _ddl.Sij3;
        // dr2 - квадрат расстояния между частицами
                    #endregion
                    if (dr2 < cutoff * cutoff)

                    {
                        neighbor[i][nnum[i]++] = j;
                        neighbor[j][nnum[j]++] = i;
        // Элементы массива neighbor[i][] несут информацию об ID-номерах «соседей» i-ой
частицы
                    }
                }
            }
                }
```

2.4 Расчет параметров локального ориентационного порядка

Ниже представлена функция «sphHarmonic», позволяющая рассчитывать сферические гармоники, необходимые при расчете параметров локального ориентационного порядка. Входными параметрами являются показатели l и m, а также $cos\theta$ и угол ϕ.

```
        complex sphHarmonic(int l, int m, double cosTheta, double phi)
                {

                    int m1 = Math.Abs(m);
                    double c = Math.Sqrt((2 * l + 1) * factorial(l - m1) / (4 * Math.PI *
factorial(l + m1)));
                    c *= legendre(l, m1, cosTheta);
                    // legendre выполняет расчет присоединенных полиномов Лежандра; '*'
применяется для сокращения записи c=c*legendre;
                    complex y = new complex();
                    double simp;
                    if (m == 1) { simp = -1.0f; } else { simp = 1.0f; }
                    if (m < 0)
                    {
                        y.r = Math.Cos(m1 * phi) * simp;
                        y.i = -Math.Sin(m1 * phi) * simp;
                        // разложение экспоненты exp(i*m*phi)на реальную и виртуальную ча-
сти в случае m<0;
                    }
                    else
                    {
                        y.r = Math.Cos(m1 * phi);
                        y.i = Math.Sin(m1 * phi);
                        // разложение экспоненты exp(i*m*phi)на реальную и виртуальную ча-
сти в случае m>0;
                    }
                    complex cmp = new complex();
```

```
                cmp.r = y.r * c;
                cmp.i = y.i * c;
                // cmp содержит итоговые значений реальной и мнимой частей сферической
гармоники ;
                return cmp;
            }
```

Расчет параметров локального ориентационного порядка производится при помощи функции «w», представленной далее.

```
        public double w(int l, Vector[] p, int n)
                // p содержит координаты частицы, n - определяет количество "соседей";
            {
                if (n < 1) { return 1.0e-138f; }
                // Исключение из рассмотрения частиц без "соседей"
                cosTheta = new double[n];
                phi = new double[n];
                for (long i = 0; i < n; i++)
                {
                    if (fzero(p[i].x))
                    {
                        if (fzero(p[i].y))
                        {
                            phi[i] = 0; // определение угла phi в случае x=0;y=0;
                        }
                        else
                        {
                            if (p[i].y > 0) //
                            {
                                phi[i] = Math.PI / 2.0;
     // Определение угла phi в случае x=0, y>0
                            }
                            else
                            {
                                phi[i] = 3.0 * Math.PI / 2.0;
    // Определение угла phi в случае x=0; y<0
                            }
                        }
                    }
                    else
                    { // Определение phi в случае ненулевого значения x
                        phi[i] = Math.Atan(p[i].y / p[i].x);
                        if (p[i].x < 0.0) { phi[i] += Math.PI; }
```

```
                    else { if (p[i].y < 0) { phi[i] += 2.0 * Math.PI; } }
                }
                cosTheta[i] = p[i].z / Math.Sqrt(p[i].x * p[i].x + p[i].y * p[i].y
+ p[i].z * p[i].z);
                // Определение cosTheta
            }
            Q = new complex[2 * l + 1];
            // Q имеет размерность 2l+1, поскольку суммирование ведется от -l до
l, включая 0
            Q[l].r = 0; Q[l].i = 0;
            for (int j = 0; j < n; j++)
                // Суммирование ведется по числу "соседей" n
            {
                Q[l].r += sphHarmonic(l, 0, cosTheta[j], phi[j]).r;
                Q[l].i += sphHarmonic(l, 0, cosTheta[j], phi[j]).i;
               // Расчет реальной и мнимой частей Q при m=0;
                for (int m = 1; m <= l; m++)
                {
                    c = new complex();
                    c = sphHarmonic(l, m, cosTheta[j], phi[j]);
                    Q[m + l].r += c.r;
                    Q[m + l].i += c.i;
                    double sign;
                    if (m == 1) { sign = -1.0f; } else { sign = 1.0f; }
                    Q[-m + l].r += sign * c.r;
                    Q[-m + l].i += -sign * c.i;
                }
            }
            for (int m = -l; m <= l; m++)
            {
                Q[m + l].r /= n;
                Q[m + l].i /= n;
                // Расчет параметра qlm
            }

            double sum = 0;
            for (int m = -l; m <= l; m++)
            {
                sum += Q[m + l].r * Q[m + l].r + Q[m + l].i * Q[m + l].i;
            }
            if (l == 6)
```

```
                {
                    for (int m = -l; m <= l; m++)
                    {
                        Q6m_tilda[iID - 1, m + l].r = Q[m + l].r / Math.Sqrt(sum);
                        Q6m_tilda[iID - 1, m + l].i = Q[m + l].i / Math.Sqrt(sum);
                    }
                }
                qLoad = Math.Sqrt((4.0 * Math.PI / (2.0 * l + 1)) * sum);
                // Расчет усредненного параметра ql
                return qLoad;
            }
```

2.5 Идентификация частиц, участвующих в формировании кристаллических структур

```
      _ddl.indexCount = 0;
                for (int i = 0; i < _ddl.N; i++) { _ddl.ClusterID[i] = 0; } // ClusterID
- принимает 0 для частиц материнской неупорядоченной фазы,
                // 1 для частиц, участвующих в формировании кристаллических структур
                for (int i = 0; i < num; i++)
                {
                    n = nnum[i] + 1; // nnum считается в genpair на основе координат
частиц;
                    if (n - 1 >= _ddl.minClSize)
      // n - число частиц, образующих кристаллический зародыш
      // minClSize - минимальный  размер зародыша, определяемый условием корреллирован-
ности
                    {
                        lq = new int[n];
                        lq[0] = i;
                        for (int j = 1; j < n; j++) { lq[j] = neighbor[i][j - 1]; }
                        idcl = true;
                        for (int j = 0; j < n; j++)
                        {
                            res = 0;
                            for (int m = -6; m <= 6; m++)
                            {
                                res += Q6m_tilda[i, m + 6].r * Q6m_tilda[lq[j], m +
6].r + Q6m_tilda[i, m + 6].i * Q6m_tilda[lq[j], m + 6].i;
      // Расчет параметра коррелированности S;
                            }
                            if (Math.Abs(res) <= 0.5) { idcl = false; }
                        }
                        if (idcl)
                        {
                            _ddl.ClusterID[i] = 1; _ddl.indexCount++;
      }
```

```
                    }
                }
                _ddl.Total_Particles = _ddl.indexCount;
        // _ddl.Total_Particles - определяет общее число частиц в системе, участвующих в
формировании кристаллических структур;
```

Библиографический список

1. Галимзянов, Б.Н. Основы моделирования молекулярной динамики: учебное пособие / А.В. Мокшин, Б.Н. Галимзянов. – М.–Ижевск: Институт компьютерных технологий, 2018. – 106 с.
2. Галимзянов, Б.Н. Молекулярная динамика при структурных трансформациях и фазовых переходах в неупорядоченных системах / Б.Н. Галимзянов, А.В. Мокшин. – Казань: Казан. ун-т, 2017. – 159 с.
3. Хуснутдинов, Р.М. Конспект лекций по курсу «Вычислительная физика« (учебно-методическое пособие) / Р.М. Хуснутдинов, А.В. Мокшин. – Казань: РИЦ Школа, 2021. – 35 с.
4. Хуснутдинов, Р.М. Сборник задач по курсу «Вычислительная физика« (учебно-методическое пособие) / Р.М. Хуснутдинов, А.В. Мокшин. – Казань: РИЦ Школа, 2021. – 47 с.
5. Stukowski, A. Visualization and analysis of atomistic simulation data with OVITO – the Open Visualization Tool Modelling / A. Stukowski. – Simul. Mater. Sci. Eng. 18. – 2010. – p. 015012.
6. Allen, M.P. Computer Simulation of Liquids / M.P. Allen and D.J. Tildesley. – Oxford: Clarendon Press, 1987. – 404 pp.
7. Гулд, Х. Компьютерное моделирование в физике / Х. Гулд, Я. Тобочник. – М.: Мир, 1990. – 350 с.
8. Товбина, Ю.К. Метод молекулярной динамики в физической химии / Под ред. проф. Ю.К. Товбина. – М.: Наука, 1996. – 334 с.
9. Браун, А.Г. Основы статистической физики: Учебное пособие / А.Г. Браун, И.Г. Левитина. – 3-е изд. – М.: НИЦ ИНФРА-М, 2015. – 120 с.
10. Steinhardt, P.J. Bond-orientational order in liquids and glasses / P.J. Steinhardt, D.R. Nelson, M. Ronchetti // Phys. Rev. B. -1983. -Vol.28. -P.784.

11. Wolde, P. Numerical Evidence for bcc Ordering at the Surface of a Critical fcc Nucleus / P. ten Wolde, M. Ruiz-Montero, and D. Frenkel // Phys. Rev. Lett. -1995. -Vol.75. -P.2714.

Приложение. Присоединенные полиномы Лежандра

Расчет присоединенных полиномов Лежандра P_n^m в случаях $n = m$ и $n = m + 1$ производится на основе следующих выражений:

$$P_l^l(x) = (-1)^l(2l-1)!!\,(1-x^2)^{\frac{1}{2}} \quad (1)$$

$$P_{l+1}^l(x) = x(2l+1)P_l^l(x). \quad (2)$$

В выражении (1) применяется двойной факториал. Двойной факториал числа n определяется произведением всех натуральных чисел в отрезке $[1, n]$, имеющих ту же четность, что и n. В случае нечетного n:

$$n!! = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot n. \quad (3)$$

В остальных случаях применяется рекуррентное соотношение:

$$(l-m)P_l^m(x) = x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x). \quad (4)$$

Расчет значений присоединенных полиномов Лежандра производится при помощи функции «legendre», представленной ниже.

```
double legendre(int l, int m, double x)
        {
            double fact, pll, pmm, pmmp1, somx2;
            pll = new double();
            int i, ll;
            pmm = 1.0f;
            if (m > 0)
            {
                somx2 = Math.Sqrt((1.0 - x) * (1.0 + x));
                // Выражение (1-x^2)^(1/2)
                fact = 1.0f; // переменная fact участвует при расчете факториала
                for (i = 1; i <= m; i++)
                {
                    pmm *= -fact * somx2;
                    fact += 2.0f;
    // каждую итерацию fact увеличивается на 2, таким образом достигается расчет двой-
ного факториала (2l-1)!!

                }
            // В цикле происходит расчет присоединенного полинома Лежандра P при l=m;
            }
```

```
                if (l == m) { return pmm; }
                else
                {
                    pmmp1 = x * (2 * m + 1) * pmm;
                    // Расчет присоединенного полинома Лежандра при l = m+1
                    if (l == (m + 1)) { return pmmp1; }
                    else
                    {
                        for (ll = m + 2; ll <= l; ll++)
                        {
                            pll = (x * (2 * ll - 1) * pmmp1 - (ll + m - 1) * pmm) /
(ll - m);
                            pmm = pmmp1;
                            pmmp1 = pll;
                        }
            // Расчет присоединенных полиномов Лежандра при помощи рекуррентных соот-
ношений
                        return pll;
                    }
                }
            }
```