分子动力学计算代码解读

1. 头文件部分

- `cstdlib`：包含`atoi`和`atof`等函数的库。
- `iomanip`：用于设置输出格式。
- `vector`：包含向量容器的库。
- `list`：包含链表容器的库。
- `algorithm`：包含算法函数的库。
- `"function.h"`：自定义的函数头文件。
- `"plotter.h"`：自定义的绘图库头文件。

2. 主要函数

- `LennardJones`函数：计算Lennard-Jones势函数对应的力。
- `Force`函数：根据位置计算力。
- `Initialize`函数：初始化位置和速度。
- `velocityVerlet`函数：实现Velocity-Verlet算法进行时间演化。
- `instantaneousTemperature`函数：计算瞬时温度。
- `Print`函数：输出位置、速度和加速度。
- `cmp`函数：用于对原子根据其z坐标进行排序。
- `Draw`函数：进行绘图，显示模拟结果。
- `main`函数：程序的主函数，包含命令行参数解析、初始化、模拟循环和绘图部分。

3. 主要函数详解

`LennardJones`函数

- 功能：计算Lennard-Jones势函数对应的力。
- 参数：
    - `r`：包含三个元素的数组，表示原子之间的距离。
    - `f`：包含三个元素的数组，表示计算得到的力。
- 这个函数实现了Lennard-Jones势函数的力计算公式。

```
inline void LennardJones(const double r[3], double f[3])
{
  double r2 = sqr(r[0])+sqr(r[1])+sqr(r[2]);
  double fc = 24 * (2 * pow(r2, -7) - pow(r2, -4));
  f[0] = r[0]*fc;
  f[1] = r[1]*fc;
  f[2] = r[2]*fc;
}
```

`Force`函数：

- 功能：根据原子的位置计算相互作用力。
- 参数：
    - `r`：包含原子位置的二维数组。
    - `a`：用于存储计算得到的加速度的二维数组。

- 这个函数使用嵌套的循环遍历所有原子对，计算原子之间的相互作用力，并更新对应的加速度。

```
void Force(const function2D<double>& r, function2D<double>& a)
{
  a=0;
  for (int i=0; i<r.size_N()-1; i++)     // loop over all distinct pairs i,j
    for (int j=i+1; j<r.size_N(); j++){
      double rij[3], fij[3];            // position of i relative to j and force
      for (int k=0; k<3; k++) rij[k] = r(i,k) - r(j,k);
      LennardJones(rij,fij);
      for (int k=0; k < 3; k++) {
    a[i][k] += fij[k];
    a[j][k] -= fij[k];
      }
    }
}
```

`Initialize`**函数**:

- 功能：初始化原子的位置和速度。
- 参数：
    - `L`：线性尺寸。
    - `Vmax`：最大初始速度分量。
    - `r`：用于存储位置的二维数组。
    - `v`：用于存储速度的二维数组。
- 这个函数将原子在立方体中均匀分布，并为每个原子分配随机的初始速度。

```
void Initialize(double L, double Vmax, function2D<double>& r, function2D<double>&
v)
{
  int N = r.size_N();
  int n = int(ceil(pow(N, 1./3.))); // number of atoms in each direction
  double a0 = L/n;                  // lattice spacing
  // initialize positions
  int p = 0; // particles are not placed close to the boundary but rather 0.5*a0
from the boundary!
  for (int ix = 0; ix < n; ix++)
    for (int iy = 0; iy < n; iy++)
      for (int iz = 0; iz < n; iz++) {
    if (p < N) {
      r(p,0) = (ix + 0.5) * a0;
      r(p,1) = (iy + 0.5) * a0;
      r(p,2) = (iz + 0.5) * a0;
    }
    ++p;
      }

  // initialize velocities
  for (int p = 0; p < N; p++)
    for (int i = 0; i < 3; i++)
      v(p,i) = Vmax * (2 * drand48()-1);
}
```

`velocityVerlet`**函数**:

- 功能：使用Velocity-Verlet算法进行时间演化。
- 参数：
    - `N`：原子数量。
    - `M`：坐标维度。
    - `dh`：时间步长。
    - `r`：包含位置的二维数组。
    - `v`：包含速度的二维数组。
    - `a`：包含加速度的二维数组。
    - `F`：函数对象，用于计算力。
- 这个函数根据给定的时间步长，使用Velocity-Verlet算法更新原子的位置和速度，并调用函数对象`F`来计算力。

```
// Velocity-Verlet algorithm written for very general system of variables
template <class storage, class functor>
void velocityVerlet(int N, int M, double dh, storage& r, storage& v, storage& a,
functor& F)
{
  F(r,a);// Computes acceleration "a" using function "F" which depends solely on
posions "r".
  for (int i=0; i<N; i++){
    for (int k=0; k<M; k++){
      v[i][k] += 0.5*a[i][k]*dh;
      r[i][k] += v[i][k]*dh;
    }
  }
  F(r,a);
  for (int i=0; i<N; i++)
    for (int k=0; k<M; k++)
      v[i][k] += 0.5*a[i][k]*dh;
}
```

`instantaneousTemperature`**函数**:

- 功能：计算瞬时温度。
- 参数：
    - `v`：包含速度的二维数组。
- 这个函数根据速度计算瞬时温度。

```
double instantaneousTemperature(function2D<double>& v){
  double sum = 0;
  for (int i=0; i<v.size_N(); i++)
    for (int k=0; k<v.size_Nd(); k++)
      sum += sqr(v[i][k]);
  return sum/(3*(v.size_N() - 1));
}
```

`Print`函数：

- 功能：输出原子的位置、速度和加速度。
- 参数：
  - `r`：包含位置的二维数组。
  - `v`：包含速度的二维数组。
  - `a`：包含加速度的二维数组。
- 这个函数使用循环遍历所有原子，并按照一定的格式输出其位置、速度和加速度。

```
void Print(const function2D<double>& r, const function2D<double>& v, const
function2D<double>& a)
{
  using namespace std;
  for (int i=0; i<r.size_N(); i++){
    cout<<setw(25)<<r(i,0)<<" "<<setw(25)<<r(i,1)<<" "<<setw(25)<<r(i,2)<<"    ";
    cout<<setw(25)<<v(i,0)<<" "<<setw(25)<<v(i,1)<<" "<<setw(25)<<v(i,2)<<"    ";
    cout<<setw(25)<<a(i,0)<<" "<<setw(25)<<a(i,1)<<" "<<setw(25)<<a(i,2)<<"    ";
    cout<<endl;
  }
}
```

`cmp`函数：

- 功能：用于对原子根据其z坐标进行排序。
- 参数：
  - `r1`：包含三个元素的数组，表示原子1的位置。
  - `r2`：包含三个元素的数组，表示原子2的位置。

```
int cmp(const vector<double>& r1, const vector<double>& r2)// Only for sorting
atoms according to their z coordinate
{ return r1[2]>r2[2];}
```

`Draw`函数：

- 功能：进行绘图，显示模拟结果。
- 参数：
  - `r`：包含位置的二维数组。
- 这个函数使用自定义的绘图库进行绘图，将原子的位置可视化。

```
void Draw(double alpha, double theta, int pixsize, Plotter& plotter, const
function2D<double>& r)
{
  // 3D->2D projection (Cavalier projection)
  double c1 = cos(theta)/tan(alpha);
  double c2 = sin(theta)/tan(alpha);

  plotter.erase();
  plotter.filltype(1);
  // atoms that have larger z coordinate should be plotted first to have the
right stacking order in 3D plot
  // need to sort coordinates
```

```
  list<vector<double> > coord;
  for (int i=0; i<r.size_N(); i++){
    vector<double> r0(3);
    r0[0] = r[i][0];
    r0[1] = r[i][1];
    r0[2] = r[i][2];
    coord.push_back(r0);
  }
  coord.sort(cmp);

  // We will draw more distant atoms by dark color and closer by bright color
  // Need the minimum and maximum z coordinate
  double zmin = (*coord.begin())[2];
  double zmax = (*(--coord.end()))[2];

  for (list<vector<double> >::iterator ri=coord.begin(); ri!=coord.end(); ri++){
    double xp = (*ri)[0]+(*ri)[2]*c1;// projected coordinates
    double yp = (*ri)[1]+(*ri)[2]*c2;
    int color = static_cast<int>(((*ri)[2]-zmin)/(zmax-zmin)*65530);
    plotter.fillcolor(color,color,color);// only grey colors used
    plotter.circle(static_cast<int>(xp*pixsize),static_cast<int>
(yp*pixsize),static_cast<int>(pixsize));
  }
  plotter.flushpl();// to flush output, otherwise some atoms might be missing on
the display
}
```

`main`函数:

- 功能：程序的主函数，包含命令行参数解析、初始化、模拟循环和绘图部分。
- 主要步骤：
    - 解析命令行参数，如粒子数、系统大小、最大步数等。
    - 初始化位置和速度。
    - 进行模拟循环，每步调用`velocityVerlet`函数进行时间演化，并输出模拟结果。
    - 每隔一段时间调用`Draw`函数进行绘图展示。

```
int main(int argc, char *argv[], char *env[])
{
  int N=64;// Number of particles
  double L=10;// Linear size of cubic volume
  double Vmax=0.2;// Maximum initial velocity component
  int MaxSteps=2000;
  double dt = 0.01; // Time-step
  int i=0;
  while (++i<argc){
    std::string str(argv[i]);
    if (str=="-N" && i<argc-1)  N = atoi(argv[++i]);
    if (str=="-L" && i<argc-1)  L = atof(argv[++i]);
    if (str=="-Vmax" && i<argc-1) Vmax = atof(argv[++i]);
    if (str=="-dt" && i<argc-1) dt = atof(argv[++i]);
    if (str=="-Ms" && i<argc-1) MaxSteps = atoi(argv[++i]);
    if (str=="-h" || str=="--help"){
      std::clog<<"********* Molecular dynamics for argon **************\n";
      std::clog<<"**                                                **\n";
```

```
      std::clog<<"**      Copyright Kristjan Haule, 26.09.2005      **\n";
      std::clog<<"************************************************\n";
      std::clog<<"\n";
      std::clog<<"dla [-N int] [-h]\n" ;
      std::clog<<"Options:   -N       Total number of particles ("<<N<<")\n";
      std::clog<<"           -L       Linear system size ("<<L<<")\n";
      std::clog<<"           -Vmax    Maximum initial velocity ("<<Vmax<<")\n";
      std::clog<<"           -Ms      Maximum number of steps ("
<<MaxSteps<<")\n";
      std::clog<<"           -dt      Time-step ("<<dt<<")\n";
      std::clog<<"************************************************\n";
      return 0;
    }
  }
  /************** Initialization of plotter ************************/
  PlotterParams params; // set a Plotter parameter
  params.setplparam ("PAGESIZE", (char *)"letter");
  XPlotter plotter(cin, cout, cerr, params); // declare Plotter
  if (plotter.openpl () < 0){ // open Plotter
    cerr << "Couldn't open Plotter\n";
    return 1;
  }
  int pixsize=500;
  plotter.fspace (-pixsize*L, -pixsize*L, pixsize*L*3, pixsize*L*3); // specify
user coor system
  /*****************************************************************************/

  // Data structures to store position, velocity and acceleartion
  function2D<double> r(N,3), v(N,3), a(N,3);

  /************** Initialization of random number gen. *********************/
  int random_seed = time(0);
  srand48(random_seed);
  /*****************************************************************************/
  Initialize(L,Vmax,r,v);// Initialize the position and velocities of atoms

  for (int i=0; i<MaxSteps; i++){
    std::cout<<i<<" "<<instantaneousTemperature(v)<<std::endl;
    velocityVerlet(N, 3, dt, r, v, a, Force);// Actual simulation
    if  (i%50==0){
      Draw(0.25*M_PI, 0.25*M_PI, pixsize, plotter, r);
    }
  }

  /************** Plotter Done**************************/
  clog<<"DONE"<<endl;
  if (plotter.closepl () < 0){ // close Plotter
    cerr << "Couldn't close Plotter\n";
    return 1;
  }
  /*****************************************************/
  return 0;
}
```