Project 1, FYS4460

Fredrik E Pettersen
f.e.pettersen@fys.uio.no

February 20, 2013

Abstract

In this project we will look at a simple linear second order differential equation.

Contents

1 About the problem **3**

2 The algorithm **3**

3 Analytic solution **3**

4 Results **3**

5 Stability and precision **3**

6 Final comments **3**

A Source code **3**

1 About the problem

The goal of this project is to model systems of Argon atoms using the Lennard-Jones potential.

2 The algorithm

3 Analytic solution

4 Results

First of all, the initial distribution of Argon is visualized in figure 1. This is a so called face centered cubic lattice, where one cube consists of $5 \times 5 \times 5 = 25$ Argon atoms.

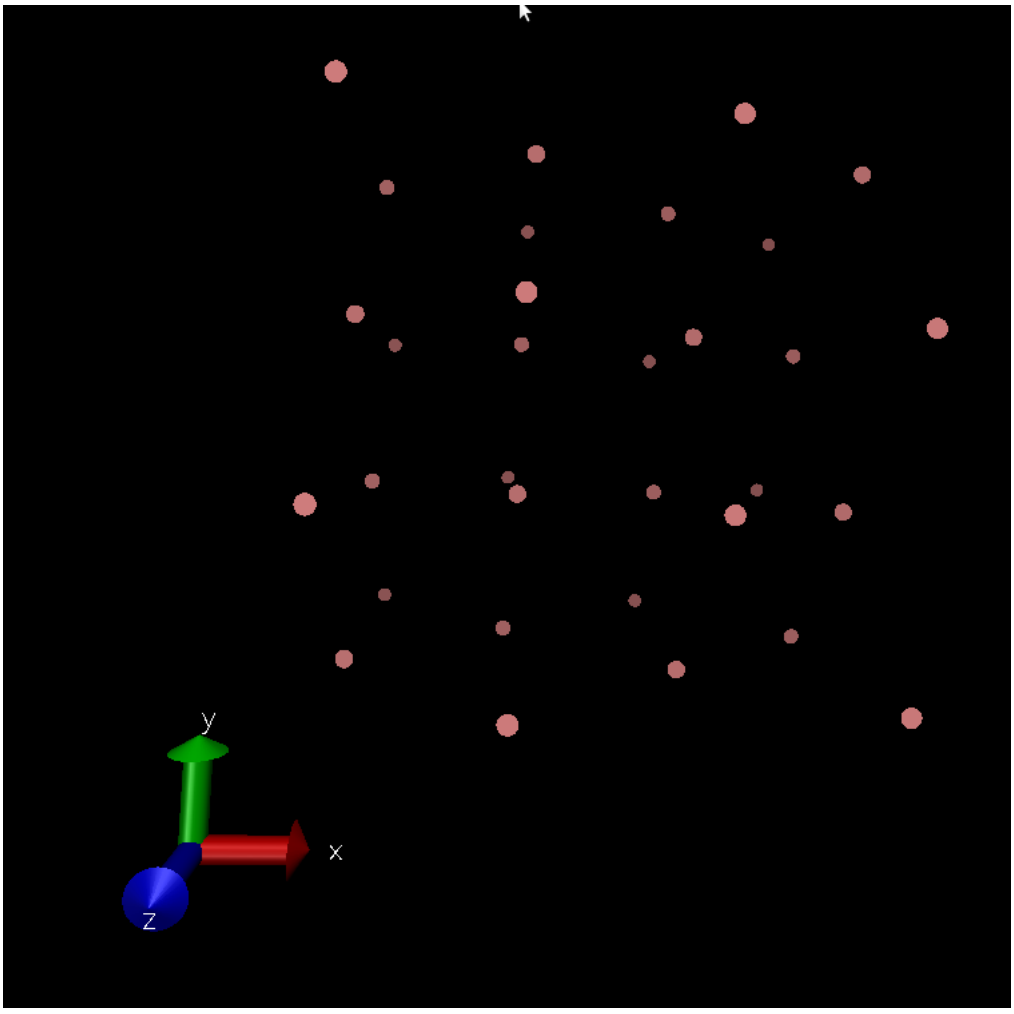

Figure 1: The initial configuration of Argon atoms makes a face-centered cubic lattice

Having placed the atoms in the correct formation, we give each atom an initial random velocity from the Bolzmann distribution depending on the temperature.
For the temperature $T = 119.8K = 1$ in MD units the velocity distribution has 0 mean and standard deviation 1. We can check this for the first resultfile (see figure **??**).

Figure 2: Velocity distribution at t=0.

5 Stability and precision

6 Final comments

A Source code

```
#include "project1.h"

using namespace std;
using namespace arma;

System::System(int ncells, int Timesteps, double Temperature)
{
    timesteps = Timesteps;
    T = Temperature;
    particles = 4*ncells*ncells*ncells;
    b = 5.260/3.405; //Aangstroms
    L = ncells*b;
    r_cut = 3;
    Ncells = ncells;
    xcells = L/r_cut;
    cells = xcells*xcells*xcells;
    particle = new Particle[particles];
    U = 0;
    res = zeros<vec>(timesteps+1);

    for(int i = 0; i < cells; i++) {
        cell.push_back(new Cell());
    }
    Initialize();
}

void System::Initialize()
{
    InitializePositions();
    InitializeVelocities(T);
    setupCells();
}

void System::InitializePositions(){
    double xCoors[] = {0.25,0.75,0.25,0.75};
    double yCoors[] = {0.25,0.75,0.75,0.25};
    double zCoors[] = {0.25,0.25,0.75,0.75};
    vec tmp;
    tmp = zeros<vec>(3);
    int counter = 0;
    for(int x=0; x<Ncells; x++){
        for(int y=0; y<Ncells; y++){
            for(int z=0; z<Ncells; z++){
                for(int k=0; k<4; k++){
                    if(counter<particles){
                        tmp(0) = b*(x+xCoors[k]); tmp(1) = b*(y+yCoors[k]); tmp(2) = b*(z+
                        particle[counter].r = tmp;
                        particle[counter].r_tmp = tmp;
                        particle[counter].r0 = tmp;
                    }
                    counter ++;
                }
            }
```

```
        }
    }
}

void System::InitializeVelocities(double T){
    vec3 sumvec = zeros(3);
    for(int i=0;i<particles; i++){
        particle[i].v = sqrt(T)*randn<vec>(3);
        sumvec += particle[i].v;
    }
    sumvec /=particles;
    for(int i=0;i<particles; i++){
        particle[i].v -=sumvec;
    }
}

void System::setupCells(){

    /*Give cellnumbers*/
    for(int i=0;i<cells;i++){
        cell[i]->setCell_no(i);
        cell[i]->setLenght(L/((double) xcells));
    }

    /*Give cells positions*/
    vec3 tmp;
    tmp = zeros(3);
    double len = cell[0]->getLength();

    int counter = 0;
    for(int x=0; x<xcells; x++){
        for(int y=0; y<xcells; y++){
            for(int z=0; z<xcells; z++){
                tmp(0) = x*len; tmp(1) = y*len; tmp(2) = z*len;
                cell.at(counter)->setPos(tmp);
                for(int b=0;b<3;b++){
                    cell[counter]->pos2(b) = fmod(tmp(b),len);
                }
                counter++;
            }
        }
    }
    vec3 dr = zeros(3);
    vec3 r1 = zeros(3);
    vec3 r2 = zeros(3);
    double length = L/((double) xcells)+0.001;
    double limit = sqrt(3)*L/((double) xcells)+0.001;
    int dummy;

    /*Find neighbours*/
    for(int i=0; i<cells;i++){
        r1 = cell[i]->getPos();
        dummy = 0;
        for(int j=0; j<i;j++){
            r2 = cell[j]->getPos();
            dr = r2-r1;
            for(int k=0;k<3;k++) {
                if(dr(k) > L/2.0){
                    dr(k) -= L;
                }
                else if(dr(k)< -L/2.0){
```

```
                    dr(k) += L;
                }
            }

            length = norm(dr,2);
            if(length <=limit){
                cell[i]->neighbours[dummy] = cell[j]->getCell_no();
                dummy++;
            }
        }
        for(int j=i+1; j<cells;j++){
            r2 = cell[j]->getPos();
            dr = r2-r1;
            for(int l=0;l<3;l++) {
                if(dr(l) > L/2.0){
                    dr(l) -= L;
                }
                else if(dr(l)< -L/2.0){
                    dr(l) += L;
                }
            }

            length = norm(dr,2);
            if(length <=limit){
                cell[i]->neighbours[dummy] = cell[j]->getCell_no();
                dummy++;
            }
        }
    }

    /*Place particles in cells*/

    for(int i=0; i<cells;i++){
        for(int j=0; j<particles; j++){
            if(cell[i]->isincell(&particle[j])){
                particle[j].cellID = cell[i]->getCell_no();
                cell[i]->addParticle(&particle[j]);
            }
        }
    }
}

void System::output(int nr){
    /*Loops through all particles and writes their positions to a numbered .xyz file*/

    char* buffer = new char[60];
    sprintf(buffer,"results_%03d.xyz",nr);
    ofstream outfile;
    cout<<buffer<<endl;
    outfile.open(buffer);
    outfile<<particles<<endl;
    outfile<<"Argon atoms using Lennard - Jones potential. timestep "<<nr<<"  "<<U<<setpre
    for(int i=0;i<particles;i++){
        outfile<<particle[i].gettype()<<" "<<particle[i].getpos()<<" "<<particle[i].getvel
              <<" "<<particle[i].cellID<<"  "<<particle[i].getForce()<<"  "<<endl;
    }
      outfile.close();
}
void System::update(double dt){
    //vec F;
    for(int i=0; i<particles; i++){
```

```
        particle[i].v = particle[i].v + particle[i].F*(dt/2.0);
        particle[i].r_tmp = particle[i].r +particle[i].v*dt;

        if(i==0){
            cout << particle[i].F << endl;
        }
    }
    accept();
    for(int i=0;i<particles;i++){
        particle[i].F = grad_U(i);
        particle[i].v = particle[i].v + particle[i].F*(dt/2.0);

    }
}
void System::update_all(double dt){
    //This will eventually be the update function utilizing neighbour lists
    U = 0;
    int index = 0;
    int n = 0;
    double length = cell[0]->getLength();
    for(int a=0;a<cells;a++){
        for(vector<Particle*>::iterator it1 = cell[a]->particles.begin(); it1 != cell[a]->
            /*SRSLY you guys, I hate you guys so much!*/
            /*Legg en god forklaring paa dette et setd!*/
            (*it1)->v = (*it1)->v + (*it1)->F*(dt/2.0);
            (*it1)->r_tmp = (*it1)->r + (*it1)->v*dt;
            (*it1)->delta_r += distance((*it1)->r_tmp,(*it1)->r);
        }
    }
    accept();
    PlaceInCells();

    for(int j=0; j<cells;j++){
        index = 0;
        for(vector<Particle*>::iterator it1 = cell[j]->particles.begin(); it1 != cell[j]->
            (*it1)->F = grad_U_new(cell[j],*it1);
            index++;
            for(int k=0; k<cell[j]->number_of_neighbours;k++){ //FIXXXX
                n = cell[j]->neighbours[k];
                (*it1)->F += grad_U_new(cell[n],*it1);
            }
            (*it1)->v = (*it1)->v + (*it1)->F*(dt/2.0);
        }
    }
    cout<<cell[0]->particles[0]->r<<endl;
}

vec3 System::force(vec dr){
//     if (norm(dr)>r_cut){
//         return zeros(3);
//     }
    double r2 = dot(dr,dr);
    double r6 = r2*r2*r2;
    double r12 = r6*r6;
    //vec F = (24*eps*pow(sigma,6)/pow(r,7))*(2*pow((sigma/r),6)-1)*(dr/r);
//     U += 4*(1/r12 -1/r6);
    vec3 F = 24*(2.0/r12 -1.0/r6)*(dr/r2);
    return  F;
}

vec3 System::grad_U(int i){
```

```
    vec3 F = zeros(3);
    for(int j=0;j<i;j++){
        F += force(particle[i].distanceToAtom(&particle[j],L));
    }
    for(int j=i+1;j<particles;j++){
        F += force(particle[i].distanceToAtom(&particle[j],L));
    }
    return -F;
}

vec3 System::grad_U_new(Cell *box,Particle *thisParticle){
    //l p over partiklene i en celle
    vec3 F = zeros(3);
    for(vector<Particle*>::iterator it2 = box->particles.begin(); it2 != box->particles.en
        if(*it2 != thisParticle){
            F += force(thisParticle->distanceToAtom(*it2,L));
        }
    }
    return -F;
}

void System::accept(){
    for(int i=0; i<particles;i++){
        particle[i].r = particle[i].r_tmp;
        particle[i].checkpos(L);
    }
}

void System::PlaceInCells(){
    for(int i=0; i<cells;i++){
        cell[i]->particles.clear();
        for(int j=0; j<particles; j++){
            if(cell[i]->isincell(&particle[j])){
                cell[i]->addParticle(&particle[j]);
            }
        }
    }
}

void System::mean_square(int nr){
    for(int i=0; i<particles; i++){
        U += dot(particle[i].delta_r,particle[i].delta_r);

    }
    res(nr)=U;
}
void System::outputMeanSquare(){
    char* buffer = new char[60];
    sprintf(buffer,"total_movement_.txt");
    ofstream outfile;
    outfile.open(buffer);
    for(int i=0;i<=timesteps;i++){
        outfile<<res(i)<<setprecision(12)<<endl;
    }
    outfile.close();
}

vec3 System::distance(vec3 r_new, vec3 r_old){
    vec3 dr = r_new-r_old;
    for(int i=0;i<3;i++) {
        if(dr(i) > L/2.0){
```

```
            dr(i) -= L;
        }
        else if(dr(i)< -L/2.0){
            dr(i) += L;
        }
    }
    return dr;
}
```

```
#include "project1.h"

using namespace std;
using namespace arma;

Cell::Cell()
{
    cell_no = 0;
    number_of_neighbours = 26;
    neighbours = new int[number_of_neighbours];
    for(int i=0; i<number_of_neighbours; i++){
        neighbours[i] = 0;
    }
    pos = zeros(3);
    pos2 = zeros(3);
    cellLength = 0;

}
vec3 Cell::distanceToCell(Cell *cell, double L){
      vec3 dr = cell->getPos()-this->pos;
    for(int i=0;i<3;i++) {
        if(dr(i) > L/2.0){
            dr(i) -= L;
        }
        else if(dr(i)< -L/2.0){
            dr(i) += L;
        }
    }
    return dr;
}
int Cell::isincell(Particle *atom){
    int x = 0;
    int y = 0;
    int z = 0;
    vec3 dr = atom->r-pos;
    if(dr[0]<cellLength && dr[0]>0){
        x = 1;
    }
    if(dr[1]<cellLength && dr[1]>0){
        y = 1;
    }
    if(dr[2]<cellLength && dr[2]>0){
        z = 1;
    }
    return (x+y+z)/3;
}
void Cell::addParticle(Particle *atom){
//     cout << particles.capacity() << "  " << particles.size() << "   " << this->getCell_n
    particles.push_back(atom);
```

```
}
/*
void Cell::FindNeighbours(Cell *cell,double r_cut,double L,int j){
    vec3 dr = zeros(3);
    cout<<"Balle nummer "<<this->cell_no<<" har pos "<<this->pos<<endl;
    int x,y,z;
    x = y = z = 0;
    dr = distanceToCell(cell,L);
    if(dr(0)<=(cellLength+0.033)){
        x=1;
    }
    if(dr(1)<=(cellLength+0.033)){
        y=1;
    }
    if(dr(2)<=(cellLength+0.033)){
        z=1;
    }
    if((x+y+z)/3){
        neighbours[j] = cell->getCell_no();
    }


}
*/
void Cell::FindNeighbours(Cell *cell,double L,int j){
    vec3 dr = cell->getPos()-pos;
    for(int i=0;i<3;i++) {
        if(dr(i) > L/2.0){
            dr(i) -= L;
        }
        else if(dr(i)< -L/2.0){
            dr(i) += L;
        }
    }
    double length = cellLength+0.001;
    int x,y,z;
    x = y = z = 0;
    if(dr(0) <= length){
        x = 1;
    }
    if(dr(1) <= length){
        y = 1;
    }
    if(dr(2) <= length){
        z = 1;
    }
    if((x+y+z)/3 == 1){
        neighbours[j] = cell->getCell_no();
    }
}




#include "project1.h"

using namespace std;
using namespace arma;

Particle::Particle()
    {
```

```
    particlename = "Ar";
    mass = 1.0; //atomic units *1.66053886e-27
    r = zeros(3);
    v = zeros(3);
    F = zeros(3);
    delta_r = zeros(3);
    r_tmp = zeros(3);
    cellID = 99;
    };
char *Particle::getpos()
{
    char* buffer = new char[60];
    sprintf(buffer, "%.12g  %.12g  %.12g", r(0), r(1), r(2));
    return buffer;
}

char *Particle::getvel()
{
    char* buffer = new char[60];
    sprintf(buffer, "%.12g  %.12g  %.12g", v(0), v(1), v(2));
    return buffer;
}
char *Particle::getForce()
{
    char* buffer = new char[60];
    sprintf(buffer, "%.12g  %.12g  %.12g", F(0), F(1), F(2));
    return buffer;
}
void Particle::checkpos(double L){
    r(0) = fmod(r(0),L) + L*(r(0) < 0);
    r(1) = fmod(r(1),L) + L*(r(1) < 0);
    r(2) = fmod(r(2),L) + L*(r(2) < 0);
}

vec3 Particle::distanceToAtom(Particle *atom, double L) {
    vec3 dr = atom->r-r;
    for(int i=0;i<3;i++) {
        if(dr(i) > L/2.0){
            dr(i) -= L;
        }
        else if(dr(i)< -L/2.0){
            dr(i) += L;
        }
    }

//     for(int i=0;i<3;i++){
//         dr(i) = (dr(i)/fabs(dr(i)))*max(dr(i),0.8);
//     }
    return dr;
}
vec3 Particle::NewdistanceToAtom(Particle *atom, double cell_length, double L) {
    vec3 dr = atom->r-r;
    for(int i=0;i<3;i++) {
        if(fabs(dr(i))> 2*cell_length){
        dr(i) = fmod(r(i) +100*cell_length,cell_length);
        }
    }

//     for(int i=0;i<3;i++){
//         dr(i) = (dr(i)/fabs(dr(i)))*max(dr(i),0.8);
//     }
```

```
    return dr;
}
char *Particle::distanceMoved(){
//     vec3 g = delta_r-r0;
    char* buffer = new char[60];
    sprintf(buffer, "%.12g  %.12g  %.12g", delta_r(0), delta_r(1), delta_r(2));
    return buffer;
}
```