dr hab. inż. Mirosław Łątka
mgr inż. Klaudia Kozłowska
Metody numeryczne
Semestr zimowy 2017/18

Laboratorium 13

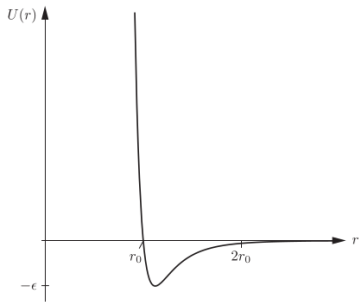
Dynamika molekularna (część 1)

Oddziaływanie pomiędzy dwoma atomami gazów szlachetnych może być modelowana za pomocą potencjału Lenarda-Jonesa:

$$u(r)=4\epsilon\left[\left(\frac{r_0}{r}\right)^{12}-\left(\frac{r_0}{r}\right)^{6}\right]. \quad (1)$$

Energia potencjalna jest zależna od odległości r pomiędzy atomami. r_0 to średnica atomu a ε jest parametrem, który determinuje siłę oddziaływania. Przykładowe wartości parametrów r_0 i ε podano w poniższej tabelce:

	r_0 (Å)	ϵ (eV)
helium	2.65	0.00057
neon	2.76	0.00315
argon	3.44	0.0105



Z wykresu energii potencjalnej wynika, że atomy odpychają się kiedy odległość pomiędzy nimi jest mniejsza od r_0. Dla odległości większych od r_0 atomy przyciągają się. W praktyce oddziaływanie pomiędzy atomami zaniedbujemy kiedy odległość pomiędzy nimi jest większa od arbitralnie wybranej wartości r_{cutoff} (większej od 2 r_0).

Całkowitą energię potencjalną układu N oddziałujących atomów obliczamy w następujący sposób

$$U=\sum_{i=1}^{N}\sum_{j>i}^{N}u(r_{ij}), \quad (2)$$

gdzie wektor $\boldsymbol{r}_i$ określa chwilowe położenie i-tego atomu, $\boldsymbol{r}_{ij}=\boldsymbol{r}_i-\boldsymbol{r}_j$ a r_{ij} to odległość pomiędzu atomem i i j.

Dynamika molekularna polega na numerycznym całkowaniu równań ruchu Newtona:

$$m\frac{d^2\boldsymbol{r}_i}{dt^2}=\boldsymbol{F}_i(\boldsymbol{r}_1,\boldsymbol{r}_2,...,\boldsymbol{r}_N), i=1,2,...,N, \quad (3)$$

gdzie $\boldsymbol{F}_i$ jest wypadkową siłę działającą na i-ty atom:

$$\boldsymbol{F}_i=\sum_{i\neq j}^{N}\boldsymbol{f}_{ij},\ \boldsymbol{f}_{ij}=\frac{-du(r_{ij})}{dr_{ij}}\frac{\boldsymbol{r}_{ij}}{r_{ij}}. \quad (4)$$

Zwróć, uwagę, że siła to pochodna energii potencjalnej *u* oddziaływania. Dla potencjału Lenarda-Jonesa siły $\boldsymbol{f}_{ij}$ mają następującą postać:

$$\boldsymbol{f}_{ij}=\frac{48\epsilon}{r_{ij}^2}\left[\left(\frac{r_0}{r_{ij}}\right)^{12}-\frac{1}{2}\left(\frac{r_0}{r_{ij}}\right)^{6}\right]\boldsymbol{r}_{ij}. \quad (5)$$

Z trzeciej zasady dynamiki Newtona wynika, że $\boldsymbol{f}_{ij}=-\boldsymbol{f}_{ji}$.

Zadania do wykonania:

1) Korzystając ze szkieletu klasy MD napisanej na zajęciach i zamieszczonego w dodatku napisz metodę **verletStep(….)** oraz pomocniczą metodę obliczającą energię kinetyczną atomów. Użyj tej drugiej metody do obliczenia energii kinetycznej na końcu metody **verletStep.** Dla uproszczenia przyjmujemy, że *m*, r_0 i ε są równe 1.

2) Napisz tester, który sprawdzi poprawność kodu dla przypadku czołowego zderzenia dwóch cząstek (osobno dla ruchu wzdłuż osi *x* i *y*) . Wykonaj wykresy:

- położenia i prędkości obu cząstek,
- energii kinetycznej, potencjalnej i całkowitej.

3) Zbadaj wpływ długości kroku całkowania na wynik symulacji zderzenia czołowego.

4) **Dla entuzjastów**: zamiast wykonywać wykresy wykonaj animację zderzenia korzystając z umieszczonego na eportalu projektu TDemo.

DODATEK 1

```java
import java.util.Arrays;

public class MD {

    private  int nAtoms;

    private  double  x[];
    private  double  y[];

    private  double  vx[];
    private  double  vy[];

    private  double  ax[];
    private  double  ay[];




    private int stepCounter;
    private double ePot;
    private double eKin;
    private double  boxWidth;

    private final double  rCut2=16.0;
    private final double wallStifness=50;



    public MD (double [] xStart, double [] yStart, double [] vxStart, double[] vyStart, int boxWidth){

        nAtoms=xStart.length;
        ax= new double[nAtoms];
        ay= new double[nAtoms];


        x= Arrays.copyOf(xStart,xStart.length);
        y= Arrays.copyOf(yStart,yStart.length);
        vx= Arrays.copyOf(vxStart,vxStart.length);
        vy= Arrays.copyOf(vyStart,vyStart.length);


        calculateAcceleration();
        calculateKineticEnergy();

    }
```

```java
    }

    public double getX(int i){
        return x[i];
    }

    public double getY(int i){
        return y[i];
    }


    public int getStepCounter() {
        return stepCounter;
    }

    public double getePot() {
        return ePot;
    }

    public double geteKin() {
        return eKin;
    }
```

```
81      private void calculateAcceleration(){
82
83          // zero acceleration vector
84
85          for (int i=0; i<nAtoms; i++){
86              ax[i]=0;
87              ay[i]=0;
88          }
89
90
91          ePot=0;
92          for (int i=0; i<nAtoms-1; i++)
93              for (int j=i+1;j<nAtoms;j++){
94
95              double dx=x[i]-x[j];
96              double dy=y[i]-y[j];
97
98              double rij2=dx*dx+dy*dy;
99
100             if (rij2<rCut2){
101                 double fr2=1./rij2;
102                 double fr6=fr2*fr2*fr2;
103                 double  fr= 48.0*fr6*(fr6-0.5)/rij2;
104                 double fxi=fr*dx;
105                 double fyi=fr*dy;
106
107                 ax[i]+=fxi; ax[j]-=fxi;
108                 ay[i]+=fyi; ay[j]-=fyi;
109                 ePot+=4*fr6*(fr6-1.0);
110
111             }
112
113             }
114
115     }
116
117     private void calculateKineticEnergy(){...}
125
126
127     public void verletStep(double dt){...}
157
158 }
159
```