Simulations in Statistical Physics

Course for MSc physics students

János Kertész

Lecture 9

MOLECULAR DYNAMICS

Deterministic simulation.
The task is to calculate the trajectory of an *N*-particle system in phase space, i.e., to solve the **Newton's equation** for it.
For simplicity we will start with point-like particles interacting through a simple pair potential. In this case we have a set of coupled simple differencial equations.

$$m_i \dot{\mathbf{v}}_i = \mathbf{f}_i = m_i \mathbf{g} + \sum_j \mathbf{f}_{ij} \quad \text{for} \quad i, j = 1,2...,N$$

The force between the interacting particles can be caculated from the pair potenctial:

$$\mathbf{f}_{ij} = -\mathbf{f}_{ji} = -\nabla u(r_{ij})$$

where the we assume that the pair potential depends only on the distance between particles *i* and *j*. Usually ***g*** ignorable.

Here we do not ask about the origin of the potential (QM).

This method is widely used from the calculation of basic properties of condensed matter to the formation of galaxies, the interaction of biopolimers etc.

Conservative Newtonian dynamics:

- Total energy (Hamiltonian time-independent)
- Totla momentum (Space homogeneous)
- Total angular momentum (Space isotropic)

Characteristic quantitites

Length:

i) Potential minimum (energeticaly favorable distance btw particles)

ii) Average distance between particles (in dense systems ~ i))

iii) Range of the potential

iv) Size of the sample

Time

i) Collision time

ii) Time between collisions

Velocity Equipartition: $v = \sqrt{\frac{3k_B T}{m}}$

from which the intercollision time can be estimated as $\frac{\sqrt[3]{\frac{V}{N}}}{\sqrt{\frac{3k_B T}{m}}}$

For the estimation of the collision time we start from the energy conservation and assume binary collisions. The two-body problem can be transformed into that of scattering on a central force potential.

$$E = \frac{1}{2} m \dot{r}^2 + u(r)$$

$$\frac{dr}{dt} = \sqrt{\frac{2}{m}\left(E - u(r)\right)}$$

$$t_c \approx 2 \int_{r_{\min}}^{r_{\max}} \frac{1}{\sqrt{\frac{2}{m}\left(E - u(r)\right)}} dr$$

In a dense fluid the order of magnitude of t_c and t_{ic} is 10^{-12}s (meaning that the binary collision picture does not work).

If the potential is smooth, the collisions take considerable time, a finite difference method can be used.

The method has to be cheap and efficient, i.e.,

-Relatively large Δt with relatively high **accuracy**
-**Stability**
-The iteration step should be **fast**

What does accuracy mean?
The initial conditions uniquely determine the trajectory in the phase space. Accuracy, however, does not mean close to this (ideal) trajectory, because the system is chaotic! Little deviations (unavoidably there because of rounding) grow exponentially fast. How can we trust MD?
The calculated trajectory does not correspond to any real one – but it is „typical" if the calculation is accurate.
However, the calculated trajectories are non-reversible.

http://www.math.buffalo.edu/306/Euler3a.html

Stability

We would like to have a method, which – though it may become inaccurate – does not go wild if Δt is increased. E.g., Euler is not good!

Iteration speed

The main loop contains the calculation of forces. This is very time consuming. Methods, which calculate forces several times within one iteration are slow.

E.g., the „default” solver is fourth order Runge-Kutta (RK4), a stable and very accurate method:

$$y' = f(t, y), \quad y(t_0) = y_0.$$

$$y_{n+1} = y_n + \tfrac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

$$t_{n+1} = t_n + h$$

$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_1)$$
$$k_3 = f(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2)$$
$$k_4 = f(t_n + h, y_n + hk_3)$$

4 times calculating forces!

MD solvers:

1. Verlet method

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t \mathbf{v}(t) + \frac{1}{2}\Delta t^2 \dot{\mathbf{v}}(t) + \ldots$$

$$\mathbf{r}(t-\Delta t) = \mathbf{r}(t) - \Delta t \mathbf{v}(t) + \frac{1}{2}\Delta t^2 \dot{\mathbf{v}}(t) + \ldots$$

$$\mathbf{r}(t+\Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\Delta t) + \Delta t^2 \ddot{\mathbf{r}}(t) + \ldots$$

From the force

The velocity is not needed to calculate the positions. It can be calculated:

$$\mathbf{v}(t) = \frac{\mathbf{r}(t+\Delta t) - \mathbf{r}(t-\Delta t)}{2\Delta t}$$

-Fast

-Third order algorithm: error $\mathcal{O}((\Delta t)^4)$ – accurate, but in **v** only $\mathcal{O}((\Delta t)^2)$

-Not self-starting (store positions at t and $t - \Delta t$) – no real problem

-Addition of $\mathcal{O}((\Delta t)^0)$ and $\mathcal{O}((\Delta t)^2)$ terms

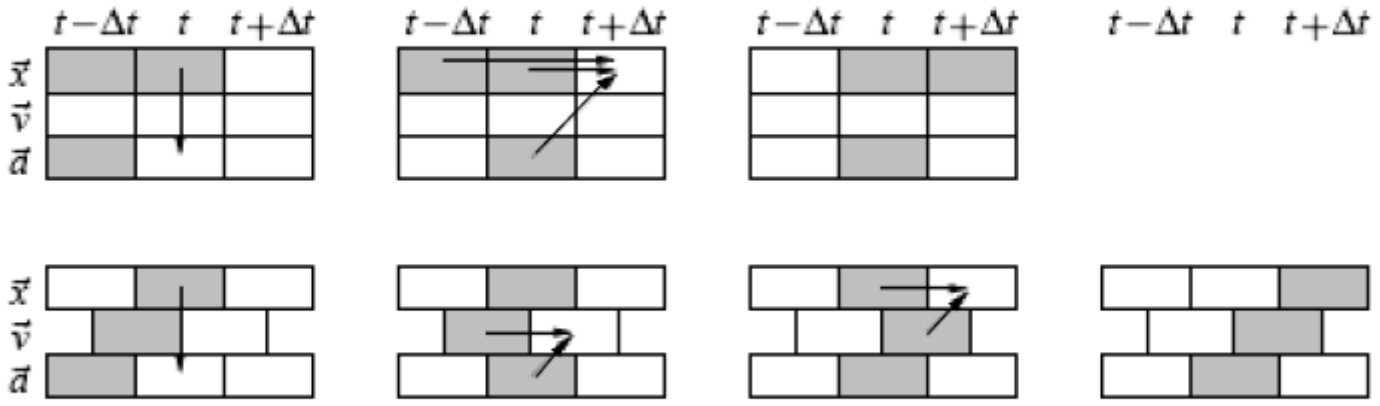
2. Leap frog method

$$\mathbf{v}(t+\frac{1}{2}\Delta t) = \mathbf{v}(t-\frac{1}{2}\Delta t) + \Delta t\dot{\mathbf{v}}(t)$$

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t+\frac{1}{2}\Delta t)$$

-Direct calculation of the velocity
-Second order both in **r** and **v**
-No addition of zeroth and second order terms
-Not self-starting

Verlet



Leap frog

3. Predictor-corrector

The idea is to use known values for **r** and its derivatives for a polynomial extrapolation to the unknown value (predictor step). Then the derivatives at the predictor value are used to interpolate to an improved solution (corrector step).

We use dimensionless variables (units m, $r_{\min}$, t_c).

In the simplest case

$$y_{n+1}^p = y_n + h f(y_n, t_n) \quad \text{Predictor}$$

Euler

$$y_{n+1} = y_n + \frac{h}{2}\left[f(y_{n+1}^p, t_{n+1}) + f(y_n, t_n)\right] \quad \text{Corrector.}$$

Higher order PC method: Predictor:

$$\mathbf{r}^P(t+\Delta t) = \mathbf{r}(t) + \Delta t\mathbf{v}(t) + \frac{1}{2}(\Delta t)^2\dot{\mathbf{v}}(t) + \frac{1}{6}(\Delta t)^3\ddot{\mathbf{v}}(t) + \ldots$$

$$\mathbf{v}^P(t+\Delta t) = \mathbf{v}(t) + \Delta t\dot{\mathbf{v}}(t) + \frac{1}{2}(\Delta t)^2\ddot{\mathbf{v}}(t) + ..$$

$$\dot{\mathbf{v}}^P(t+\Delta t) = \dot{\mathbf{v}}(t) + \Delta t\ddot{\mathbf{v}}(t) + \ldots$$

$$\ddot{\mathbf{v}}^P(t+\Delta t) = \ddot{\mathbf{v}}(t) + \ldots$$

Corrector:

$$\dot{\mathbf{v}}^C = \mathbf{f}/m$$

$$\delta\mathbf{a} = \dot{\mathbf{v}}^C - \dot{\mathbf{v}}^P$$

$$\mathbf{r}^C = \mathbf{r}^P + c_1\delta\mathbf{a}$$

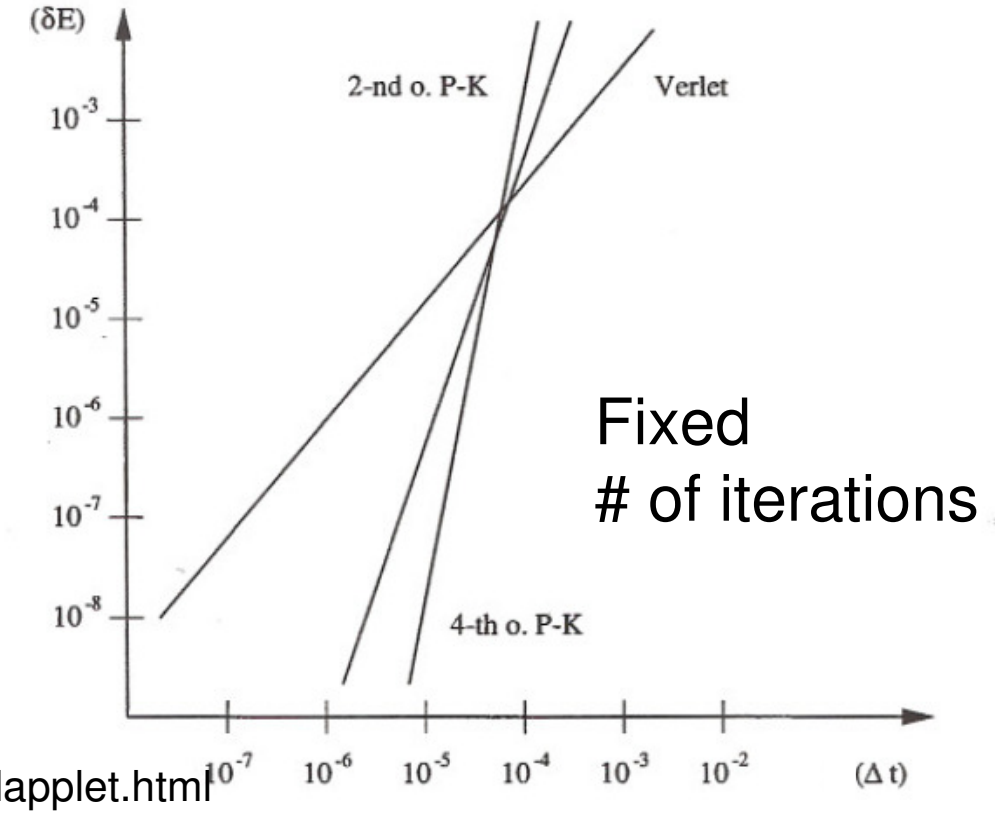
$$\mathbf{v}^C = \mathbf{v}^P + c_2\delta\mathbf{a}$$

$$\ddot{\mathbf{v}}^C = \ddot{\mathbf{v}}^P + c_4\delta\mathbf{a}$$

C_i-s are the Gear coefficients. Their values are determined byminimizing the error: $c_1 = 1/6, \quad c_2 = 5/6, \quad c_4 = 1/3$. (Trivially, c_3=0.)
Rule of tumb: Use $\Delta t < t_c / 20$.

Comparison of methods: Usually Verlet/Leap frog does the job
If higher accuracy is needed (smaller Δt) then PC overtakes.

In a closed system energy should be conserved. Due to inaccuracies there are fluctuations. The quantity $\delta E = \sqrt{< E^2 > - < E >^2}$ is a measure of the accuracy of the method.



Fixed
# of iterations

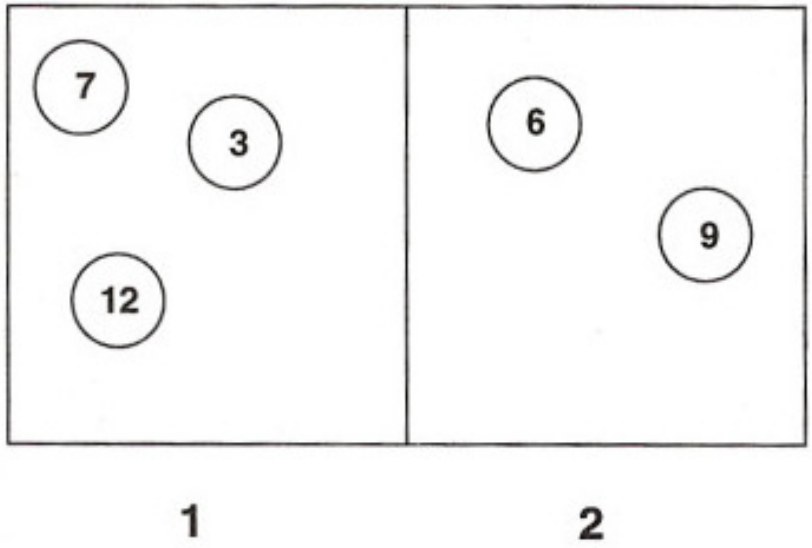
http://physics.weber.edu/schroeder/software/mdapplet.html

Starting with a few hundred atoms in the 60'-s, today billions of atoms can be simulated and hydrodynamics understood from the molecular level. The dream to simulate macroscopic objects is within reach: Avogadro project (~2012). Massive parallel computing, petaflop (10^{15}/s) machines are needed.

Some tricks have to be used on the smaller scale too.
1. Don't calculate the distance (square root of the sum of the squares of the coordinate differences) but use r^2 instead. If the potential is too complicated, use pre-tabulated values.
2. The short range pair potentials (like LJ) have to be truncated.
3. Nevertheless, if we don't know the positions of the particles, N^2 operations have to be carried out in the core of the program. This can be reduced by appropriate book-keeping. Two methods: Verlet tables and linked cells.

YouTube demos

Here we discuss the linked cell method (VT-s are based on similar ideas). Let us deal with the 2d case. The box is divided covered by a mesh where the mesh size is such that a particle cannot have interaction with another one if it is not in one of the particles 8 neighboring cells (> r_{max}). The cells are numbered and the particles too. There is a pointer which shows for each cell, which is the first particle in it. Then there is an array which runs through the particles in the cell and indicates if the last one was taken. Update needed after particle leaves a cell



BEG(1)=7
BEG(2)=6

LIST(7)=3
LIST(3)=12
LIST(12)=0
LIST(6)=9
LIST(9)=0

LAST(7)=0

LAST(3)=7

LAST(12)=3…