Molecular Dynamics with C++

Lars Pastewka, Wolfram G. Nöhring

April 22, 2021

Department of Microsystems Engineering
University of Freiburg

Contents

Chapter 1

Introduction

Context: We start by introducing the concept of the potential energy and the interatomic force. Those are the central ingredients to the molecular dynamics simulation method.

1.1 Structure of matter at the atomic scale

All matter is build out of quark and leptons, or perhaps even smaller particles, but for the sake of modeling the real material world the atom is the fundamental unit. Atoms can be described by nuclei and electrons or through "coarse-grained" models that ignore the fact that there are electrons. Both types of models are useful for describing materials, and the latter ones will be extensively used in this class.

Atoms in solids can arrange in different configurations that are called crystals when there is long-ranged order or glasses when there is not. (All solid matter typically has short-ranged order that is determined by the chemical bonds between atoms.) Atoms in solids are immobile and self-diffusion is limited. Conversely, liquids and gases are disordered (like glasses) but have mobile constituent atoms. Macroscopic object typically contain a lot of atoms – on the order of Avogadro's constant $N_A \approx 6 \times 10^{23}$. The atomic-scale simulation techniques discussed in this class can at the time of this writing (2020) treat on order of $\sim 10^6$ atoms, $10^8 - 10^9$ if you use the biggest computers available to us. Of course, this boundary is pushed towards larger systems as computer technology evolves.

We can nowadays even observe matter at atomic scales and "see" individual atoms. The collaborative research center 103 has produced an extremely

instructive video on the structure of specific type of alloys, dubbed "superalloy", that is used in e.g. turbine blades. Enjoy the ride from the blade to the atom. This class is about modeling matter at the smallest scales that you see in this video.

1.2 Interatomic forces and the potential energy

Atoms interact via forces. As Feynman put it in his famous lectures on physics, the fundamental truth about man's understanding of the physical world is "that all things are made of atoms – little particles that move around in perpetual motion, attracting each other when they are a little distance apart, but repelling upon being squeezed into one another". Indeed this is the essence of the molecular dynamics simulation method.

As the simplest example why atoms attract each other, let us consider the example of simple salt, e.g. Na-Cl that we all have sitting in our kitchen. Na-Cl in its solid form is an ionic crystal. Na atoms have approximately a charge of $q_{\mathrm{Na}} = +1|e|$, where e is the electron charge, and Cl atoms have a charge of approximately $q_{\mathrm{Na}} = -1|e|$. The Coulomb interaction between these atoms is a fundamental force of nature. Basic physical principles tell us, that the interaction energy between a Na and a Cl atom is given by

$$V_{\mathrm{Coulomb}}(r; q_{\mathrm{Na}}, q_{\mathrm{Cl}}) = \frac{1}{4\pi\varepsilon_0}\frac{q_{\mathrm{Na}}q_{\mathrm{Cl}}}{r}. \tag{1.1}$$

We also know that this energy is pair-wise additive, allowing us to write down the Coulomb interaction energy for Na-Cl consisting of N atoms,

$$E_{\mathrm{Coulomb}}(\{\vec{r}_i\}) = \sum_{i=1}^{N}\sum_{j=i+1}^{N} V_{\mathrm{Coulomb}}(r_{ij}; q_i, q_j) = \frac{1}{4\pi\varepsilon_0}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\frac{q_i q_j}{r_{ij}} \tag{1.2}$$

where q_i is the charge on atom i and $r_{ij} = |\vec{r}_i - \vec{r}_j|$ the distance between atom i and j. Note that we have introduced — in passing — a central quantity of the molecular dynamics method, the atomic positions $\vec{r}_i$ and Eq. (1.2) indicates that the interaction energy depends on the positions of all atoms.

The Coulomb interaction has a singularity at $r \to 0$. The attractive force between opposite charges becomes infinitely large. The salt crystal does not collapse because atoms are, as Feynman puts it, "repelling upon being squeezed into one another". While the attraction between our Na and Cl atoms are described by a fundamental force of nature, it is more difficult to

understand the origin of this repulsion. Hand-wavingly, it goes back to the fact that electrons are Fermions and electrons from the electron shells of Na and Cl therefore cannot exist at the same point in space (and the same spin state). This is the Pauli exclusion principle and the resulting repulsive force is called Pauli repulsion.

Different models for the Pauli repulsion exist. While the Coulomb interaction is a fundamental force of nature, these models are approximations to the true quantum physics that is the origin of the repulsive form. Two common forms are exponential repulsion,

$$E_{\text{rep,exp}}(\{\vec{r}_i\}) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} A e^{-r/\rho}, \tag{1.3}$$

or an algebraic repulsion of the form

$$E_{\text{rep,12}}(\{\vec{r}_i\}) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} A r^{-12}. \tag{1.4}$$

Note that A and ρ are *parameters*, that need to be somehow determined. This can be done with the help of either experimental data or *first-principles* calculations, that treat the electrons explicitly. These parameters depend on the atom types under consideration and in contrast to the parameter that show up in the Coulomb interaction (the permittivity ε_0), they are not universal.

For our Na-Cl model, we combine Coulomb interaction with an exponential repulsion, to give the total energy

$$E_{\text{pot}}(\{\vec{r}_i\}) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \left(A_{ij} e^{-r_{ij}/\rho_{ij}} + \frac{1}{4\pi\varepsilon_0} \frac{q_i q_j}{r_{ij}} \right). \tag{1.5}$$

This energy is called the *potential energy* and is the central property of an atomic-scale model. With Eq. (1.5), we have also encountered our first atomic-scale model for a real material. Potentials that can be decomposed as Eq. (1.5) into pair-wise terms are called *pair potentials*. They are often written as

$$E_{\text{pot}}(\{\vec{r}_i\}) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} V(r_{ij}), \tag{1.6}$$

with

$$V(r_{ij}) = A_{ij} e^{-r_{ij}/\rho_{ij}} + \frac{1}{4\pi\varepsilon_0} \frac{q_i q_j}{r_{ij}} \tag{1.7}$$

for the above potential. The quantity $V(r_{ij})$ is called the pair interaction energy.

Likely the most famous pair-potential is the Lennard-Jones potential. Its pair interaction energy is given by

$$V(r_{ij}) = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]. \tag{1.8}$$

The repulsive term $\propto r^{-12}$ is one of the models for Pauli repulsion discussed above. The attractive term $\propto r^{-6}$ arises from *London dispersion interactions*. Dispersion forces exist between all atoms, even uncharged molecules or noble gases. They are widely employed for the nonbonded portion of valence force-fields. Simple Lennard-Jones systems are often used to study generic phenomena found in real materials, e.g. for example the glass transition or plasticity of amorphous materials. There are limitations of pair potentials and more sophisticated potential energy models have been developed over the past decades. We will discuss a few of those in Chapter **??**.

Note: A repulsive term of the form r^{-12} is more attractive from a simulation point of view since it is faster to compute than an exponential. This has helped popularize the Lennard-Jones potential in the early days of molecular dynamics simulations.

The significance of the potential energy is that we can derive forces from it,

$$\vec{f}_k = -\frac{\partial}{\partial \vec{r}_k} E_{\text{pot}}(\{\vec{r}_i\}). \tag{1.9}$$

These forces are the essential ingredient to *molecular dynamics*, as they determine the motion of the atoms.

The potential energy itself describes what is called the *potential energy landscape*. The potential energy landscape depends on $3N$ degrees of freedom (as compared to the landscape we experience while walking, which depends on 2 degrees of freedom); it is therefore an object that is complex to visualize. Simplifying aspects of it is the core of *molecular statics*. For example, it is typically important to at least identify the ground-state of a system; this is the most stable configuration of a material and has the lowest possible potential energy. There is usually some crystal that is lower in energy than the energy of a glass with the same stoichiometry. Yet, in many real-world engineering applications the materials are not in their crystalline ground-state, the most common material we encounter with this property may be window

glass. In molecular statics we therefore seek to enumerate those *local minima* of the potential energy landscape and energy barriers between them.

Since the dynamics of a molecular system is determined by the forces, we only need to specify the potential energy up to a constant which disappears in the derivative Eq. (1.9). We can therefore measure the potential energy with respect to any reference configuration. This reference configuration is often the atomized state of the material, where all of the constituent atoms sit individually in vacuum and are not interacting with each other. If this reference situation is assigned the energy 0, then the potential energy is generally negative, because if it was positive the system would spontaneously atomize. (Remember, any physical system evolves to a state of lower energy.)

Chapter 2

Molecular dynamics

Context: Molecular *dynamics* follows the motion of individual atoms through a solution of Newton's equations of motion. We need integration algorithms to be able to solve this set of coupled differential equations on a computer.

2.1 Newton's equations of motion

We have now (almost) all the ingredients to carry out a molecular dynamics simulation. From our or potential energy expression $E_{\text{pot}}(\{\vec{r}_i\})$ discussed in the previous chapter, we obtain the force

$$\vec{f}_i = \partial E_{\text{pot}} / \partial \vec{r}_i \tag{2.1}$$

on each of the N atoms. Once we know the forces, we can obtain the accelerations $\vec{a}_i$ through Newton's third law,

$$\vec{f}_i = m_i \vec{a}_i. \tag{2.2}$$

We are therefore assuming that atom i can be described as a point of mass m_i! The mass can be obtained from the periodic table of elements. Note that the mass listed in the periodic table is usually the average over all isotopes weighted by their occurrence on earth, and this mass is used for most practical purposes. For some application, in particular to understand the different behavior of Hydrogen and Deuterium, it can be necessary to actually model the individual isotopes by using their respective mass.

We further have $\vec{a}_i = \dot{\vec{v}}_i$, where $\vec{v}_i$ is the velocity of atom i, and $\vec{v}_i = \dot{\vec{r}}_i$. The dot superscript indicates derivative with respect to time. The set of

linear differential equations to solve is therefore

$$\dot{v}_i(t) = \vec{f}_i(t)/m_i \quad \text{and} \quad \dot{\vec{r}}_i(t) = \vec{v}_i(t) \tag{2.3}$$

with the initial (boundary) conditions $\vec{r}_i(0) = \vec{r}_0$ and $\vec{v}_i(0) = \vec{v}_0$. Note that the boundary condition is an integral part of the differential Eq. (2.3). The state of the system is therefore fully and uniquely determined by the positions $\vec{r}_i$ and the velocities $\vec{v}_i$ of all atoms. This set of positions $\vec{r}_i$ and momenta $\vec{p}_i = \vec{v}_i/m_i$ defines a point in *phase-space* $\vec{\Gamma} = \{\vec{r}_i, \vec{p}_i\}$. The evolution of position and velocities given by Eq. (2.3) can therefore be thought of as a single point moving in the $6N$ dimensional phase-space. The concept of a phase-space will become important in the next chapter when we talk about statistical mechanics.

Code example: For a molecular dynamics code, it is useful to have a data structure that represents the state of the simulation and stores at least positions and velocities. This data structure could also store element names (or atomic numbers), masses and forces. An example that uses Eigen arrays as the basic array container is shown below.

```
using Positions_t = Eigen::Array3Xd;
using Velocities_t = Eigen::Array3Xd;
using Forces_t = Eigen::Array3Xd;

class Atoms {
public:
    Positions_t positions;
    Velocities_t velocities;
    Forces_t forces;

    Atoms(Positions_t &p) :
            positions{p}, velocities{3, p.cols()}, forces
   {3, p.cols()} {
        velocities.setZero();
        forces.setZero();
    }

    size_t nb_atoms() {
        return positions.cols();
    }
};
```

As a general rule, the data structure should be designed in a way that data that is processed consecutively is also stored in memory in a continuous manner. This ensures cache coherenece. For example, we could be tempted

to create a class `Atom` that contains the positions, velocities, etc. of a single atom and than use an array (e.g. `std::vector<Atom> atoms`) of that class as the basic data structure. However, positions are then no longer consecutive in memory. A function (e.g. computing forces) does not need the velocities would still load them into the cache, as the cache line size for all modern CPUs is 64 bytes. For high-performance numerical code, it is therefore *always* preferable to use structures of arrays rather than arrays of structure.

2.2 Kinetic energy and energy conservation

In addition to the potential energy $E_{\text{pot}}(\{\vec{r}_i\})$, the dynamical state of a system is characterized by its kinetic energy,

$$E_{\text{kin}}(\{\vec{p}_i\}) = \sum_i \frac{1}{2}\frac{p_i^2}{m_i}. \tag{2.4}$$

Note that the total energy

$$H(\{\vec{r}_i\},\{\vec{p}_i\}) = E_{\text{kin}}(\{\vec{p}_i\}) + E_{\text{pot}}(\{\vec{r}_i\}) \tag{2.5}$$

is a conserved quantity during the motion of the atoms. This can be seen by showing that the derivative of the total energy with respect to time vanishes,

$$\dot{H} = \dot{E}_{\text{kin}} + \dot{E}_{\text{pot}} = \sum_i \frac{\vec{p}_i\dot{\vec{p}}_i}{m_i} + \sum_i \frac{\partial E_{\text{pot}}}{\partial \vec{r}_i}\dot{\vec{r}}_i = \sum_i \vec{v}_i\vec{f}_i - \sum_i \vec{v}_i\vec{f}_i = 0. \tag{2.6}$$

H is also called the *Hamiltonian* of the system.

Note: Measuring the total energy (or any other conserved quantity!) and checking whether it is constant in a molecular dynamics simulation is a way of testing if the time step Δt used in the numerical integration is small enough. We will discuss numerical integration in detail below.

A generalization of Newton's equations of motion are *Hamilton's equation of motion*,

$$\dot{\vec{r}}_i = \frac{\partial H}{\partial \vec{p}_i} \tag{2.7}$$

$$\dot{\vec{p}}_i = -\frac{\partial H}{\partial \vec{r}_i}, \tag{2.8}$$

and it is straightforward to show that this reduces to Newton's equations of motions for the Hamiltonian given by Eq. (2.5). Hamilton's equation of motion remain valid when positions $\vec{r}_i$ and momenta $\vec{p}_i$ are replaced by generalized coordinates that consider constraints, such as for example the angle of a (rigid) pendulum. These equations will become important when we discuss statistical mechanics and temperature control in molecular dynamics simulations using *thermostats*, where a generalized degree of freedom is the internal state of the heat bath that controls the temperature. A full derivation of Hamilton's equations of motion is given in Chap. **??**.

Note: The *temperature* is simply a measure of the kinetic energy of the system, $\frac{3}{2}Nk_BT = E_{\text{kin}}$ where N is the number of atoms. In other words, E_{kin} measures the variance of the velocity distribution, which is Gaussian. We will learn more about this when discussing the basics of statistical mechanics.

2.3 Integration algorithms

The main ingredient in any molecular dynamics simulation, regardless of the underlying model, is the numerical solution of Eqs. (2.3). A plethora of algorithms have been developed over the years, but for most practical purposes the Velocity-Verlet algorithm is used nowadays. For instructive purposes we will start out with a simple integration method, the Euler integration, before discussing Velocity-Verlet.

2.3.1 Euler integration

In order to follow the trajectories of all atoms we need to integrate the above differential equation. On a computer, a continuous differential equation needs to be replaced by a discrete equation. Equations (2.3) are continuous in time and hence need to be discretized. (Note that our system is already discrete spatially since we are dealing with mass points, but each of these points corresponds to a physical object so this is not the result of a discretization procedure.) The simplest integration is the Euler algorithm in which forces and velocities are assumed to be constant over time intervals Δt.

To see this, we write the above differential equation as

$$d\vec{v}_i = \frac{\vec{f}_i(t)}{m_i} \quad \text{and } d\vec{r}_i(t) = \vec{v}_i(t)\, dt \tag{2.9}$$

i.e., we move the differential dt of $\dot{\vec{v}}_i = d\vec{v}/dt$ to the right hand side of the equation. We can now straightforwardly integrate the equation from time t to time $t + \Delta t$ while assuming that $\vec{f}_i$ and $\vec{v}_i$ remain constant. This yields

$$\vec{v}_i(t + \Delta t) - \vec{v}_i(t) = \frac{\vec{f}_i(t)}{m_i}\Delta t \tag{2.10}$$

$$\vec{r}_i(t + \Delta t) - \vec{r}_i(t) = \vec{v}_i(t)\Delta t \tag{2.11}$$

which is obviously only a good approximation for small Δt! This algorithm is called Euler integration.

Same equation can be derived by Taylor-expanding $\vec{r}_i(t + \Delta t)$ up to first order in Δt. The algorithm is hence $O(\Delta t^2)$. The Euler algorithm is not reversible, i.e. starting from time $t + \Delta t$ and integrating backwards one ends up with a different result at time t. Applying the Euler algorithm with timestep $-\Delta t$ gives

$$\vec{v}_i(t) - \vec{v}_i(t + \Delta t) = -\frac{\vec{f}_i(t + \Delta t)}{m_i}\Delta t \tag{2.12}$$

$$\vec{r}_i(t) - \vec{r}_i(t + \Delta t) = -\vec{v}_i(t + \Delta t)\Delta t \tag{2.13}$$

These equations cannot be re-arranged to give Eqs. (2.10) and (2.11). Euler integration is generally not a good algorithm and requires very small time steps.

2.3.2 Leap-frog integration

Leap-frog stores position at times t_i and velocities at times $t_i + \Delta t/2$ and can be derived from a argument similar to the one given above. Specifically, we combine the results of a Taylor expansion $\pm\Delta t/2$, yielding

$$\vec{v}_i(t + \Delta t/2) - \vec{v}_i(t - \Delta t/2) = \frac{\vec{f}_i(t)}{m_i}\Delta t \tag{2.14}$$

$$\vec{r}_i(t + \Delta t) - \vec{r}_i(t) = \vec{v}_i(t + \Delta t/2)\Delta t. \tag{2.15}$$

Note that Eq. (2.14) is similar to Eq. (2.10), except the force is evaluated at the mid-point. The resulting algorithm is reversible. Applying the Leap-frog algorithm with timestep $-\Delta t$ gives

$$\vec{v}_i(t - \Delta t/2) - \vec{v}_i(t + \Delta t/2) = -\frac{\vec{f}_i(t)}{m_i}\Delta t \tag{2.16}$$

$$\vec{r}_i(t) - \vec{r}_i(t + \Delta t) = -\vec{v}_i(t + \Delta t/2)\Delta t \tag{2.17}$$

Bring the terms on the left hand side to the right and vice-versa, and you arrive at the original equations for forward integration. Leap-frog is therefore *reversible*.

2.3.3 Verlet integration

Let us now Taylor expand $\vec{r}_i(t \pm \Delta t)$ up to third order in $\pm\Delta t$,

$$\vec{r}_i(t \pm \Delta t) = \vec{r}_i(t) \pm \vec{v}_i(t)\Delta t + \frac{1}{2m_i}\vec{f}_i(t)\Delta t^2 \pm \frac{1}{6}\dddot{\vec{r}}_i(t)\Delta t^3 + O(\Delta t^4). \quad (2.18)$$

Note that only the odd exponents see the sign of $\pm\Delta t$. The sum of this equation for expansion in $+\Delta t$ and $-\Delta t$ gives the positions update,

$$\vec{r}_i(t + \Delta t) + \vec{r}_i(t - \Delta t) = 2\vec{r}_i(t) + \frac{1}{m_i}\vec{f}_i(t)\Delta t^2 + O(\Delta t^4). \quad (2.19)$$

Eq. (2.19) is called the Verlet algorithm. Instead of requiring the positions $\{\vec{r}_i(t)\}$ and velocities $\{\vec{v}_i(t)\}$ it requires the positions of the current $\{\vec{r}_i(t)\}$ and past $\{\vec{r}_i(t - \Delta t)\}$ times for the integration.

The difference between the expansion for $+\Delta t$ and $-\Delta t$ yields the velocities,

$$\vec{r}_i(t + \Delta t) - \vec{r}_i(t - \Delta t) = 2\vec{v}_i(t)\Delta t + O(\Delta t^3). \quad (2.20)$$

Note that in order to compute the velocities at time t in the regular Verlet algorithm, we need to know the positions at time $t + \Delta t$. Verlet and Leap-Frog are identical algorithms, since Leap-Frog stores the velocities at the intermediate time $t + \Delta t/2$. It is usually useful to be able to know both, positions and velocities, at time t. This problem is solved by the Velocity-Verlet algorithm, described in the following section.

2.3.4 Velocity-Verlet integration

Let us now also Taylor expand $\vec{r}_i(t)$ up to third order in Δt at $\vec{r}_i(t + \Delta t)$, i.e. we integrate backwards in time from $t + \Delta t$ to t. This gives

$$\vec{r}_i(t) = \vec{r}_i(t + \Delta t) - \vec{v}_i(t + \Delta t)\Delta t + \frac{1}{2m_i}\vec{f}_i(t + \Delta t)\Delta t^2 - \frac{1}{6}\dddot{\vec{r}}_i(t)\Delta t^3 + O(\Delta t^3) \quad (2.21)$$

Equation (2.18) is the positions update of the Velocity-Verlet algorithm. The sum of Eq. (2.18) and Eq. (2.21) gives the velocity update in the Velocity-Verlet algorithm:

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \vec{v}_i(t)\Delta t + \frac{1}{2m_i}\vec{f}_i(t)\Delta t^2 \quad (2.22)$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \frac{1}{2m_i}\left(\vec{f}_i(t) + \vec{f}_i(t + \Delta t)\right)\Delta t, \quad (2.23)$$

Note that this algorithm is often split in the form of a predictor-corrector scheme since this saves computation time and the necessity to keep past forces around. The predictor step is

$$\vec{v}_i(t + \Delta t/2) = \vec{v}_i(t) + \frac{1}{2m_i}\vec{f}_i(t)\Delta t \tag{2.24}$$

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \vec{v}_i(t + \Delta t/2)\Delta t \tag{2.25}$$

where $\vec{v}_i(t+\Delta t/2)$ is the predicted velocity. After this we compute new forces, $\vec{f}_i(t + \Delta t)$. We then correct the velocities via

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t + \Delta t/2) + \frac{1}{2m_i}\vec{f}_i(t + \Delta t)\Delta t \tag{2.26}$$

The Velocity-Verlet algorithm is the integration algorithm used in most molecular dynamics codes. It has the additional properties that is it *symplectic*, which means it conserves phase-space volume. We will come back to what this mean when talking about statistical mechanics.

Code example: We can implement the velocity-verlet algorithm in a few lines of C++ code using vectorized `Eigen` operations. The prediction step

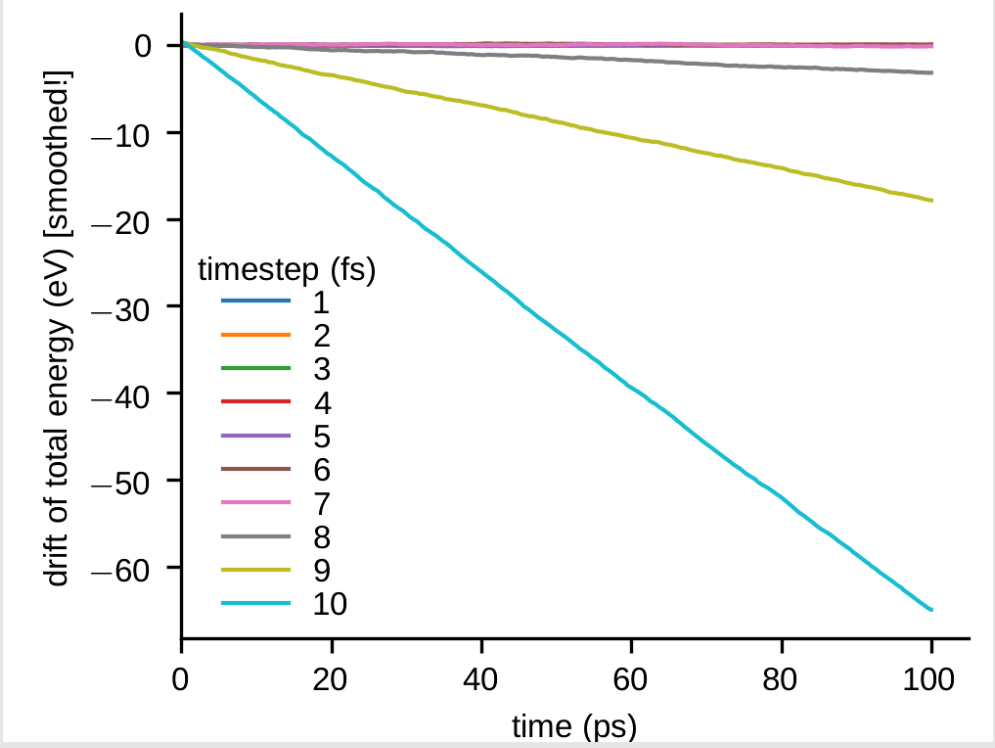
```
void verlet_step1(Atoms &atoms, double timestep, double
    mass) {
     atoms.velocities += 0.5 * atoms.forces * timestep /
    mass;
     atoms.positions += atoms.velocities * timestep;
}
```

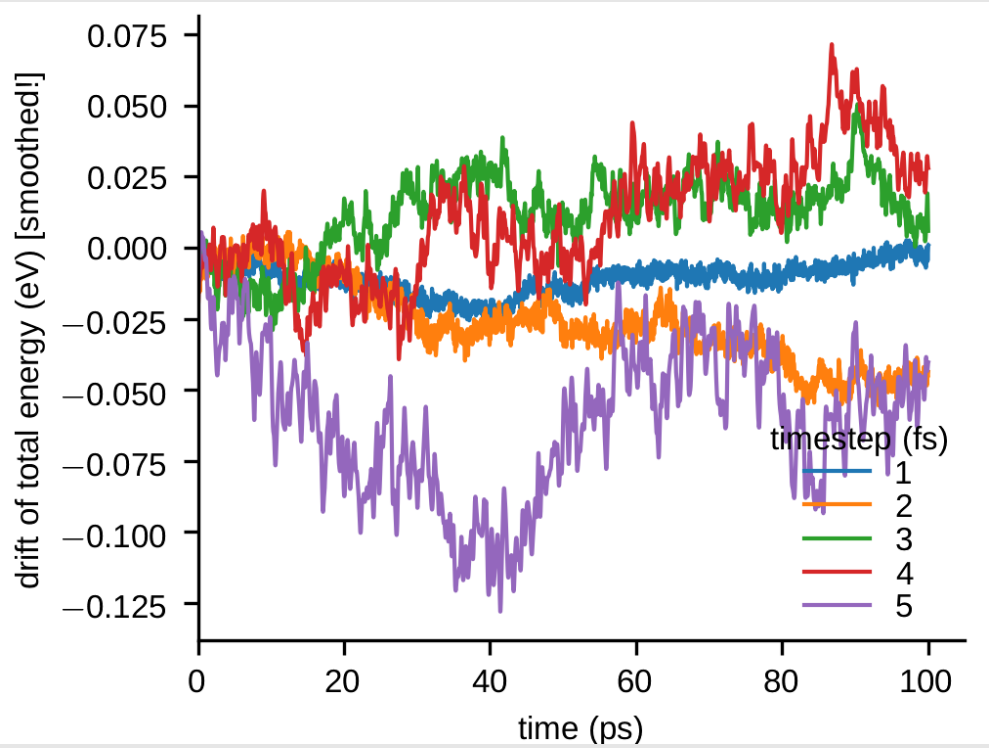
implements Eq. (2.24). We then compute new forces and correct the velocities via

```
void verlet_step2(Atoms &atoms, double timestep, double
    mass) {
     atoms.velocities += 0.5 * atoms.forces * timestep /
    mass;
}
```

Note: The timestep in MD simulations has to be on the order of femtoseconds, in order to resolve the fastest atomic vibrations. For example, in simulations with metals and Embedded Atom Method (EAM) potentials, $\Delta t = 1$ fs is typically a safe choice. How can we check that the timestep is sensible? One possibility is to simply propage a configuration in time

using the Velocity-Verlet algorithm. This is sometimes called the micro-canonical or NVE ensemble. (NVE because number of atoms, volume and energy is constant.) We then record the evolution of the total (kinetic plus potential) energy, which should be constant. The discrete time integration scheme will introduce numerical errors. If Δt is too large, there will be noticeable drift of the total energy. The figures below show the results of such a simulation. A system of 108000 Au atoms was simulated for 100 ps with various values of Δt. The y-axis shows the difference between the current and initial values of the total energy. The data was smoothened to suppress high-frequency fluctuations in the figure. For this system, even 5 fs would still be an acceptable time step.




0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
drift of total energy (eV) [smoothed!]
0
20
40
60
80
100
time (ps)
timestep (fs)
1
2
3
4
5