Molecular Dynamics I: Principles

Basics of the molecular-dynamics (MD) method[1-3] are described, along with corresponding data structures in program, `md.c`.

Newton's Second Law of Motion

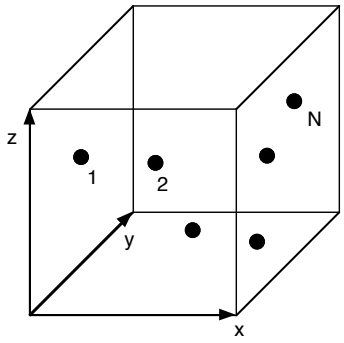
TRAJECTORY, COORDINATE, AND ACCELERATION

- Physical system = a set of atomic coordinates:

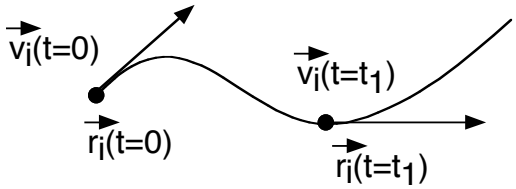
$$\{\vec{r}_i = (x_i, y_i, z_i) \mid x_i, y_i, z_i \in \Re, i = 0, ..., N-1\},$$

where $\Re$ is the set of real numbers (in the program, represented by a double precision variable) and we use a vector notation, $\vec{r}_i$.

(Data strcutures in `md.c`)
`int nAtom`: N, the number of atoms.
`NMAX`: Maximum number of atoms that can be handled by the program.
`double r[NMAX][3]`: `r[i][0]`, `r[i][1]`, and `r[i][2]` are the x, y, and z coordinates of the i-th atom, where i = 0, .., N-1.



- **Trajectory**: A mapping from time to a point in the 3-dimensional space, $t \in \Re \mapsto \vec{r}_i(t) \in \Re^3$. In fact, a trajectory of an N-atom system is regarded as a curve in 3N-dimensional space. A point on the curve is then specified by a 3N-element vector, $\vec{r}^N = (x_0, y_0, z_0, x_1, y_1, z_1, ..., x_{N-1}, y_{N-1}, z_{N-1})$.



- **Velocity**: Short-time limit of an average speed (how fast and in which direction the particle is moving),

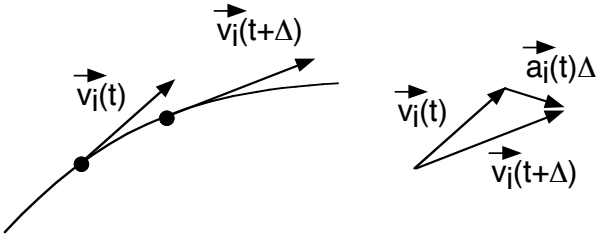
$$\vec{v}_i(t) = \dot{\vec{r}}_i(t) = \frac{d\vec{r}}{dt} \equiv \lim_{\Delta \to 0} \frac{\vec{r}_i(t+\Delta) - \vec{r}_i(t)}{\Delta}.$$

`double rv[NMAX][3]`: `rv[i][0]`, `rv[i][1]`, and `rv[i][2]` are the x, y, and z components of the velocity vector, $\vec{v}_i$, of the i-th atom.

- **Acceleration**: Rate at which a velocity changes (whether the particle is accelerating or decelerating),

$$\vec{a}_i(t) = \ddot{\vec{r}}_i(t) = \frac{d^2\vec{r}}{dt^2} = \frac{d\vec{v}_i}{dt} \equiv \lim_{\Delta \to 0} \frac{\vec{v}_i(t+\Delta) - \vec{v}_i(t)}{\Delta}.$$

`double ra[NMAX][3]`: `ra[i][0]`, `ra[i][1]`, and `ra[i][2]` are the x, y, and z components of the acceleration vector, $\vec{a}_i$, of the i-th atom.



The acceleration of a particle can be estimated from three consecutive positions on its trajectory separated by small time increments:

$$\begin{aligned}\vec{a}_i(t) &= \lim_{\Delta\to 0}\frac{\vec{v}_i(t+\Delta/2)-\vec{v}_i(t-\Delta/2)}{\Delta}\\ &= \lim_{\Delta\to 0}\frac{\dfrac{\vec{r}_i(t+\Delta)-\vec{r}_i(t)}{\Delta}-\dfrac{\vec{r}_i(t)-\vec{r}_i(t-\Delta)}{\Delta}}{\Delta}\\ &= \lim_{\Delta\to 0}\frac{\vec{r}_i(t+\Delta)-2\vec{r}_i(t)+\vec{r}_i(t-\Delta)}{\Delta^2}\end{aligned}$$

NEWTON'S EQUATION

Newton's equation states that the acceleration of a particle is proportional to the force acting on the particle,

$$m\ddot{\vec{r}}_i(t)=\vec{F}_i(t),$$

where m is the mass of the particle. For a heavier particle, the same force causes less acceleration (or deceleration).

- Initial value problem: Given initial particle positions and velocities, $\left\{\left(\vec{r}_i(0),\vec{v}_i(0)\right)\middle| i=0,...,N-1\right\}$, obtain those at later times, $\left\{\left(\vec{r}_i(t),\vec{v}_i(t)\right)\middle| i=0,...,N-1; t\geq 0\right\}$. Note that both positions and velocities must be specified in order to predict future trajectories.

Potential Energy

We consider forces which are derived from a potential energy, $V(\vec{r}^N)$, which is a function of all atomic positions.

$$\vec{F}_k=-\frac{\partial}{\partial\vec{r}_k}V\left(\vec{r}^N\right)=-\left(\frac{\partial V}{\partial x_k},\frac{\partial V}{\partial y_k},\frac{\partial V}{\partial z_k}\right).$$

Here **partial derivative** is defined as

$$\frac{\partial V}{\partial x_k}=\lim_{h\to 0}\frac{V(x_0,y_0,z_0,...,x_k+h,y_k,z_k,...x_{N-1},y_{N-1},z_{N-1})-V(x_0,y_0,z_0,...,x_k,y_k,z_k,...x_{N-1},y_{N-1},z_{N-1})}{h},$$

i.e., what is the rate of change in V when we slightly change one coordinate of an atom while keeping all the other coordinates fixed.

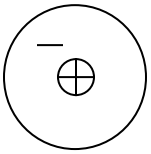
LENNARD-JONES POTENTIAL

To model certain materials such as neon and argon liquids, the Lennard-Jones potential is often used by scientists:

$$V\left(\vec{r}^{\,N}\right) = \sum_{i<j} u(r_{ij}) = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} u(|\,\vec{r}_{ij}\,|)$$

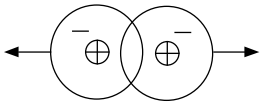
where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ is the relative position vector between atoms *i* and *j*, and

$$u(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right].$$

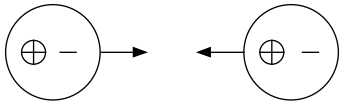


• Physical meaning of the potential: An atom consists of a nucleus and electrons surrounding it. Two atoms interact with each other in the following ways.

> Short-range repulsion (the first term): the Pauli exclusion principle states that electrons cannot occupy the same position, resulting in repulsion between the atoms. Note that for $r \to 0$, this term is dominant.



> Long-range attraction (the second term): Electrons around nuclei polarize (change distribution), creating electrostatic attraction between atoms. Note that for $r \to \infty$, this term becomes dominant.



• For Argon atoms,

> $m = 6.6 \times 10^{-23}$ gram

> $\varepsilon = 1.66 \times 10^{-14}$ erg (erg = gram•cm^2/(second^2) is a unit of energy)

> $\sigma = 3.4 \times 10^{-8}$ cm

* Don't program with these numbers (in the CGS unit); it will cause floating-point under- or over-flows.

NORMALIZATION

• We introduce normalized coordinates, $\vec{r}_i'$, potential energy, *V*', and time, *t*', that are dimensionless and are defined as follows.

$$\begin{cases} \vec{r}_i = \vec{r}_i'\sigma = 3.4\times10^{-8}[\text{cm}]\times\vec{r}_i' \\ V = V'\varepsilon = 1.66\times10^{-14}[\text{erg}]\times V' \\ t = \sigma\sqrt{m/\varepsilon}\,t' = 2.2\times10^{-12}[\text{sec}]\times t' \end{cases}$$

With these substitutions, Newton's equation becomes

$$m\frac{\varepsilon}{m\sigma^2}\sigma\frac{d^2\vec{r}_i'}{dt'^2} = -\frac{\varepsilon}{\sigma}\frac{\partial V'}{\partial\vec{r}_i'}.$$

* In the derivation above, note that

$$\begin{aligned} m\frac{d^2\vec{r}_i}{dt^2} &= m\lim_{\Delta\to0}\frac{\vec{r}_i(t+\Delta) - 2\vec{r}_i(t) + \vec{r}_i(t-\Delta)}{\Delta^2} \\ &= m\lim_{\Delta\to0}\frac{\sigma\left[\vec{r}_i'(t+\Delta) - 2\vec{r}_i'(t) + \vec{r}_i'(t-\Delta)\right]}{\left[\sigma\sqrt{m/\varepsilon}\right]^2(\Delta')^2} \end{aligned}$$

In summary, the normalized Newton's equation for atoms interacting with the Lennard-Jones potential is given below. (From now on, we always use normalized variables, and omit primes.)

$$\frac{d^2\vec{r}_i}{dt^2} = -\frac{\partial V}{\partial \vec{r}_i} = \vec{a}_i$$

$$V(\vec{r}^N) = \sum_{i<j} u(r_{ij})$$

$$u(r) = 4\left(\frac{1}{r^{12}} - \frac{1}{r^6}\right)$$

The figure in the right shows the normalized Lennard-Jones potential, $u(r)$, as a function of interatomic distance, r. The distance, r_0, at which the potential takes its minimum value, $u(r_0)$, is obtained as follows.

$$\left.\frac{du}{dr}\right|_{r=r_0} = 0 \quad \Rightarrow \quad r_0 = 2^{1/6} \approx 1.12$$

$$u(r_0) = 4\left(\frac{1}{2^{12/6}} - \frac{1}{2^{6/6}}\right) = -1$$

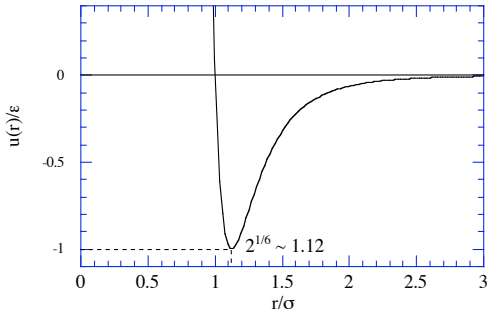


Figure: Lennard-Jones potential.

ANALYTIC FORMULA FOR FORCES

$$\vec{a}_k = -\frac{\partial}{\partial \vec{r}_k}\sum_{i<j} u(r_{ij})$$

$$= -\sum_{i<j} \frac{\partial r_{ij}}{\partial \vec{r}_k}\frac{du}{dr_{ij}}$$

Let's evaluate the two factors in the above expression:

1.

$$\frac{\partial r_{ij}}{\partial \vec{r}_k} = \left(\frac{\partial}{\partial x_k}, \frac{\partial}{\partial y_k}, \frac{\partial}{\partial z_k}\right)\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$= \frac{\left(2(x_i - x_j), 2(y_i - y_j), 2(z_i - z_j)\right)}{2\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}}\left(\delta_{ik} - \delta_{jk}\right)$$

$$= \frac{\vec{r}_{ij}}{r_{ij}}\left(\delta_{ik} - \delta_{jk}\right)$$

$$\left(\because \frac{d}{dx}[f(x)]^{1/2} = \frac{1}{2}[f(x)]^{-1/2}\frac{df}{dx}\right)$$

2.

$$\frac{du}{dr} = 4\left(-\frac{12}{r^{13}} + \frac{6}{r^7}\right) = -\frac{48}{r}\left(\frac{1}{r^{12}} - \frac{1}{2r^6}\right)$$

$$\left(\because \frac{d}{dr}r^{-n} = -nr^{-n-1}\right)$$

In the first result,

$$\delta_{ik} = \begin{cases} 1 & (i = k) \\ 0 & (i \neq k) \end{cases}$$

is called Kronecker's delta expression.

Substituting these two results back into the expression for $\vec{a}_k$, we obtain

$$\vec{a}_k = \sum_{i<j} \vec{r}_{ij} \left(-\frac{1}{r}\frac{du}{dr} \right)_{r=r_{ij}} \left(\delta_{ik} - \delta_{jk} \right)$$

where

$$-\frac{1}{r}\frac{du}{dr} = \frac{48}{r^2}\left(\frac{1}{r^{12}} - \frac{1}{2r^6} \right)$$

Summary of Molecular-Dynamics Equation System

Given initial atomic positions and velocities, $\left\{ \left(\vec{r}_i(0), \vec{v}_i(0) \right) \middle| i = 0,...,N-1 \right\}$, obtain those at later times, $\left\{ \left(\vec{r}_i(t), \vec{v}_i(t) \right) \middle| i = 0,...,N-1; t \geq 0 \right\}$, by integrating the following differential equation:

$$\ddot{\vec{r}}_k(t) = \vec{a}_k(t) = -\frac{\partial}{\partial \vec{r}_k} \sum_{i<j} u\left(r_{ij} \right) = \sum_{i<j} \vec{r}_{ij}(t) \left(-\frac{1}{r}\frac{du}{dr} \right)_{r=r_{ij}(t)} \left(\delta_{ik} - \delta_{jk} \right)$$

where

$$-\frac{1}{r}\frac{du}{dr} = \frac{48}{r^2}\left(\frac{1}{r^{12}} - \frac{1}{2r^6} \right)$$

The sum over i and *j* to evaluate the acceleration is implemented in a program as follows:

for $i = 0$ to N-1, $\vec{a}_i = 0$
for $i = 0$ to N-2
  for $j = i$+1 to N-1
    compute $\vec{\alpha} = \vec{r}_{ij} \left(-\frac{1}{r}\frac{du}{dr} \right)_{r=|\vec{r}_{ij}|}$
    $\vec{a}_i$ + = $\vec{\alpha}$
    $\vec{a}_j$ − = $\vec{\alpha}$

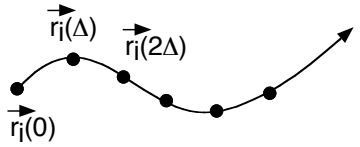
Discretization

We need to discretize the trajectories in order to solve the problem on a computer: Instead of considering

$$\left(\vec{r}_i(t), \vec{v}_i(t) \right) \quad \text{for} \quad t \geq 0$$

for continuous time, we consider a sequence of states

$$\left(\vec{r}_i(0), \vec{v}_i(0) \right) \mapsto \left(\vec{r}_i(\Delta), \vec{v}_i(\Delta) \right) \mapsto \left(\vec{r}_i(2\Delta), \vec{v}_i(2\Delta) \right) \mapsto \cdots$$



The question is: How to predict the next state, $\left(\vec{r}_i(t + \Delta), \vec{v}_i(t + \Delta) \right)$, from the current state, $\left(\vec{r}_i(t), \vec{v}_i(t) \right)$?

`double DeltaT`: Δ in `md.c`.

VERLET DISCRETIZATION

Let's apply the Taylor expansion

$$f(x_0+h)=\sum_{n=0}^{\infty}\frac{h^n}{n!}\frac{d^nf}{dx^n}\bigg|_{x=x_0}=f(x_0)+h\frac{df}{dx}\bigg|_{x=x_0}+\frac{h^2}{2}\frac{d^2f}{dx^2}\bigg|_{x=x_0}+\frac{h^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x=x_0}+\cdots,$$

which predicts a function value at a distant point, x_0+h, from derivatives at x_0, to an atomic coordinate.

$$\begin{aligned}
&\vec{r}_i(t+\Delta)=\vec{r}_i(t)+\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2+\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)\\
+\quad&\vec{r}_i(t-\Delta)=\vec{r}_i(t)-\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2-\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)\\
\hline
&\vec{r}_i(t+\Delta)+\vec{r}_i(t-\Delta)=2\vec{r}_i(t)+\vec{a}_i(t)\Delta^2+O(\Delta^4)\\
&\therefore\ \vec{r}_i(t+\Delta)=2\vec{r}_i(t)-\vec{r}_i(t-\Delta)+\vec{a}_i(t)\Delta^2+O(\Delta^4)
\end{aligned}$$

To obtain velocities,

$$\begin{aligned}
&\vec{r}_i(t+\Delta)=\vec{r}_i(t)+\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2+\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)\\
-\quad&\vec{r}_i(t-\Delta)=\vec{r}_i(t)-\vec{v}_i(t)\Delta+\frac{1}{2}\vec{a}_i(t)\Delta^2-\frac{1}{6}\dddot{\vec{r}}_i(t)\Delta^3+O(\Delta^4)\\
\hline
&\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)=2\vec{v}_i(t)\Delta+O(\Delta^3)\\
&\therefore\ \vec{v}_i(t)=\frac{\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)}{2\Delta}+O(\Delta^2)
\end{aligned}$$

The above two results constitute the Verlet discretization scheme.

(Verlet discretization scheme)

$$\begin{cases}\vec{r}_i(t+\Delta)=2\vec{r}_i(t)-\vec{r}_i(t-\Delta)+\vec{a}_i(t)\Delta^2+O(\Delta^4) & (1)\\ \vec{v}_i(t)=\dfrac{\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)}{2\Delta}+O(\Delta^2) & (2)\end{cases}$$

The Verlet algorithm for MD integration is a straight-forward application of these two equations.

(Verlet algorithm)

Given $\vec{r}_i(t-\Delta)$ and $\vec{r}_i(t)$,

1. Compute $\vec{a}_i(t)$ as a function of $\{\vec{r}_i(t)\}$,
2. $\vec{r}_i(t+\Delta)\leftarrow 2\vec{r}_i(t)-\vec{r}_i(t-\Delta)+\vec{a}_i(t)\Delta^2$,
3. $\vec{v}_i(t)\leftarrow\left[\vec{r}_i(t+\Delta)-\vec{r}_i(t-\Delta)\right]/(2\Delta)$,

A problem of this discretization scheme is that the velocity at time t cannot be calculated until the coordinate at time $t+\Delta$ is calculated. In the slight variation, called the velocity-Verlet scheme, of the above discretization scheme, we are given $\left(\vec{r}_i(t),\vec{v}_i(t)\right)$ and predict $\left(\vec{r}_i(t+\Delta),\vec{v}_i(t+\Delta)\right)$.

VELOCITY-VERLET DISCRETIZATION

(Theorem) The following algebraic equation gives the same sequence of states, $\left(\vec{r}_i(n\Delta), \vec{v}_i(n\Delta)\right)$, as that obtained by the Verlet discretization.

$$\begin{cases} \vec{r}_i(t+\Delta) = \vec{r}_i(t) + \vec{v}_i(t)\Delta + \frac{1}{2}\vec{a}_i(t)\Delta^2 & (3) \\ \vec{v}_i(t+\Delta) = \vec{v}_i(t) + \frac{\vec{a}_i(t) + \vec{a}_i(t+\Delta)}{2}\Delta & (4) \end{cases}$$

(Proof)

1. Using Eq. (3) of the velocity-Verlet discretization scheme for two consecutive time steps, t and t+Δ,

$$\begin{aligned} \vec{r}_i(t+2\Delta) &= \vec{r}_i(t+\Delta) + \vec{v}_i(t+\Delta)\Delta + \frac{1}{2}\vec{a}_i(t+\Delta)\Delta^2 \\ - \quad \vec{r}_i(t+\Delta) &= \vec{r}_i(t) + \vec{v}_i(t)\Delta + \frac{1}{2}\vec{a}_i(t)\Delta^2 \\ \hline \vec{r}_i(t+2\Delta) - \vec{r}_i(t+\Delta) &= \vec{r}_i(t+\Delta) - \vec{r}_i(t) + \left[\vec{v}_i(t+\Delta) - \vec{v}_i(t)\right]\Delta + \frac{\vec{a}_i(t+\Delta) - \vec{a}_i(t)}{2}\Delta^2 \end{aligned}$$

   Let's eliminate the velocities using Eq. (4),

$$\vec{r}_i(t+2\Delta) - \vec{r}_i(t+\Delta) = \vec{r}_i(t+\Delta) - \vec{r}_i(t) + \left[\frac{\vec{a}_i(t+\Delta) + \vec{a}_i(t)}{2}\Delta\right]\Delta + \frac{\vec{a}_i(t+\Delta) - \vec{a}_i(t)}{2}\Delta^2$$

   i.e.,

$$\begin{aligned} \vec{r}_i(t+2\Delta) - \vec{r}_i(t+\Delta) &= \vec{r}_i(t+\Delta) - \vec{r}_i(t) + \vec{a}_i(t+\Delta)\Delta^2 \\ \therefore \vec{r}_i(t+2\Delta) &= 2\vec{r}_i(t+\Delta) - \vec{r}_i(t) + \vec{a}_i(t+\Delta)\Delta^2 \end{aligned}$$

   This is nothing but the Verlet rule, Eq. (1), applied to t+2Δ instead of t+Δ. We have thus derived the first equation of the Verlet scheme from the two equations in the velocity-Verlet scheme.

2. Using Eq. (3)

$$\begin{aligned} \vec{r}_i(t+\Delta) &= \vec{r}_i(t) + \vec{v}_i(t)\Delta + \frac{1}{2}\vec{a}_i(t)\Delta^2 \\ + \quad \vec{r}_i(t) &= \vec{r}_i(t-\Delta) + \vec{v}_i(t-\Delta)\Delta + \frac{1}{2}\vec{a}_i(t-\Delta)\Delta^2 \\ \hline \vec{r}_i(t+\Delta) &= \vec{r}_i(t-\Delta) + \left[\vec{v}_i(t) + \vec{v}_i(t-\Delta)\right]\Delta + \frac{\vec{a}_i(t) + \vec{a}_i(t-\Delta)}{2}\Delta^2 \end{aligned} \quad (5)$$

   By multiplying Δ to Eq. (4) with the time shifted by -Δ,

$$0 = \left[\vec{v}_i(t) - \vec{v}_i(t-\Delta)\right]\Delta - \frac{\vec{a}_i(t) + \vec{a}_i(t-\Delta)}{2}\Delta^2 \quad (6)$$

   By adding Eqs. (5) and (6),

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

   and this is equivalent to the second equation, Eq. (2), of the Verlet scheme. In summary, by using only the two equations in the velocity-Verlet scheme, Eqs. (3) and (4), we have derived both equations, Eqs. (1) and (2), of the Verlet scheme. Thus the two schemes are equivalent, given atomic positions at the first two time steps.//

VELOCITY-VERLET ALGORITHM

For notational convenience, let's define

$$\vec{v}_i(t+\frac{\Delta}{2}) \equiv \vec{v}_i(t) + \frac{\Delta}{2}\vec{a}_i(t). \tag{7}$$

Eqs. (3) and (4) are then rewritten as

$$\begin{cases} \vec{r}_i(t+\Delta) = \vec{r}_i(t) + \vec{v}_i(t+\frac{\Delta}{2})\Delta & (8) \\ \vec{v}_i(t+\Delta) = \vec{v}_i(t+\frac{\Delta}{2}) + \frac{\Delta}{2}\vec{a}_i(t+\Delta) & (9) \end{cases}$$

The velocity-Verlet algorithm is just a re-statement of Eqs. (7)–(9).

(Velocity-Verlet algorithm)

Given $\left(\vec{r}_i(t), \vec{v}_i(t)\right)$,

1. Compute $\vec{a}_i(t)$ as a function of $\{\vec{r}_i(t)\}$,
2. $\vec{v}_i(t+\frac{\Delta}{2}) \leftarrow \vec{v}_i(t) + \frac{\Delta}{2}\vec{a}_i(t)$,
3. $\vec{r}_i(t+\Delta) \leftarrow \vec{r}_i(t) + \vec{v}_i(t+\frac{\Delta}{2})\Delta$,
4. Compute $\vec{a}_i(t+\Delta)$ as a function of $\{\vec{r}_i(t+\Delta)\}$,
5. $\vec{v}_i(t+\Delta) \leftarrow \vec{v}_i(t+\frac{\Delta}{2}) + \frac{\Delta}{2}\vec{a}_i(t+\Delta)$

Computing $\vec{a}_i(t)$ as a function of $\{\vec{r}_i(t)\}$ is done in function `computeAccel()`. In the program, we perform an MD simulation for `StepLimit` steps, and `computeAccel()` is called `StepLimit`+1 times.

(Velocity-Verlet algorithm for `StepLimit` steps)

Initialize $\left(\vec{r}_i, \vec{v}_i\right)$ for all i
Compute $\vec{a}_i$ as a function of $\{\vec{r}_i\}$ for all i—function `computeAccel()`
for `stepCount` = 1 to `StepLimit`
  do the following—function `singleStep()`
    $\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i\Delta/2$ for all i
    $\vec{r}_i \leftarrow \vec{r}_i + \vec{v}_i\Delta$ for all i
    Compute $\vec{a}_i$ as a function of $\{\vec{r}_i\}$ for all i—function `computeAccel()`
    $\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i\Delta/2$ for all i
endfor

Now we have learned the skeleton of `md.c` program. The next lecture will deal with some details of the program. These include:

- Periodic boundary conditions
- Shifted potential
- Energy conservation and temperature
- Initialization: fcc lattice, random velocity

* The velocity-Verlet algorithm is more elegantly derived using a split-operator technique in the Ph.D. thesis by M. Tuckerman at Columbia University. See "Reversible multiple time scale molecular dynamics," M. Tuckerman, B. J. Berne, and G. J. Martyna, *J. Chem. Phys.* **97**, 1990 (1992).

Molecular Dynamics II: Implementation Details

Periodic Boundary Conditions

(There are only two places you see this in the program)

This technique allows us to simulate bulk solid and liquid properties with a small number of atoms ($N < 1{,}000$) by eliminating surface effects.

Note that for 1,000 atoms arranged in a 10 × 10 × 10 cube,

$$10^3 - 8^3 = 488 \text{ atoms}$$

(nearly half the atoms) are at the surface. Since these surface atoms are in very different environment from the bulk atoms, the average properties computed for this system are very different from bulk properties.

Periodic boundary conditions are implemented by replicating a simulation box of size $L_x \times L_y \times L_z$ to form an infinite lattice. Note that we assume atoms to be contained in a rectangular simulation box.

`double Region[3]`: Three-element array containing the box lengths, L_x, L_y, and L_z

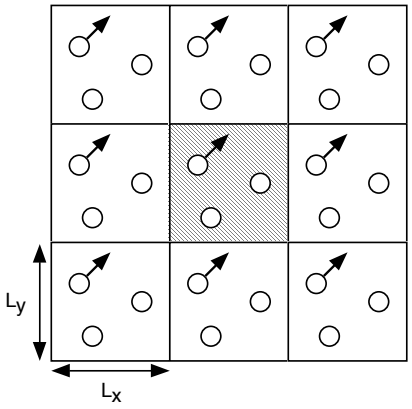


Figure: Periodically repeated images of an original simulation box (shaded) in 2 dimensions.

As an atom moves in the original simulation box, all the images move in a concerted manner by the same amount.

Since all the images are just shifted copies of an original atom, we need to keep only the coordinates of the original (central) image as a representative of all images. When an atom leaves the central box by crossing a boundary, attention is switched to the image just entering the central box.

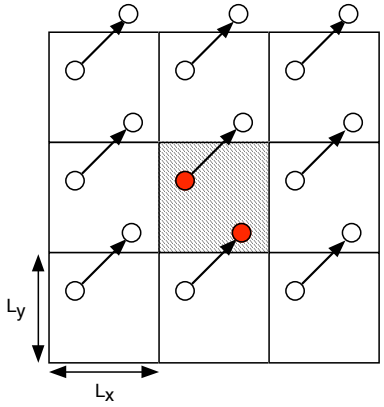


Figure: Change of the representative atom due to boundary crossing of the atom.

At every MD step, we enforce that an atomic coordinate satisfies

$$0 \le x_i < L_x,\ 0 \le y_i < L_y,\ 0 \le z_i < L_z.$$

If $\vec{r}_i(t)$ is in the central box and time increment Δ is small, $\vec{r}_i(t+\Delta)$ is at most in the neighbor image (if this assumption becomes invalid, then exceptions may occur during program execution): $-L_x \le x_i(t+\Delta) < 2L_x$, etc. In this case, an atom can be "pulled back" to the central box by

$$\begin{cases} x_i \leftarrow x_i - \mathrm{SignR}\left(\frac{L_x}{2}, x_i\right) - \mathrm{SignR}\left(\frac{L_x}{2}, x_i - L_x\right) \\ y_i \leftarrow y_i - \mathrm{SignR}\left(\frac{L_y}{2}, y_i\right) - \mathrm{SignR}\left(\frac{L_y}{2}, y_i - L_y\right) \\ z_i \leftarrow z_i - \mathrm{SignR}\left(\frac{L_z}{2}, z_i\right) - \mathrm{SignR}\left(\frac{L_z}{2}, z_i - L_z\right) \end{cases}$$

where

$$\mathrm{SignR}\left(\frac{L_x}{2}, x_i\right) = \begin{cases} \frac{L_x}{2} & x_i > 0 \\ -\frac{L_x}{2} & x_i \le 0 \end{cases}$$

and this means

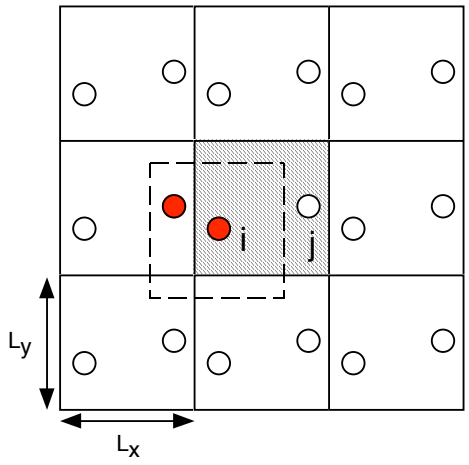
$$\begin{cases} L_x < x_i & x_i - \frac{L_x}{2} - \frac{L_x}{2} = x_i - L_x \\ 0 < x_i \le L_x & x_i - \frac{L_x}{2} + \frac{L_x}{2} = x_i \\ x_i \le 0 & x_i + \frac{L_x}{2} + \frac{L_x}{2} = x_i + L_x \end{cases}$$

`double RegionH[3]`: An array ($L_x/2$, $L_y/2$, $L_z/2$)

Minimum Image Convention

In principle, an atom interacts with all the images of another atom (and even with its own images except for itself).

In practice, we make the simulation box larger enough so that $u(r > L_x/2)$ etc. can be neglected. Then an atom interacts with only the nearest image of another atom.



Or we can imagine a box of size $L_x \times L_y \times L_z$ centered at atom i, and this atom interacts only with other atoms in this imaginary box. Therefore,

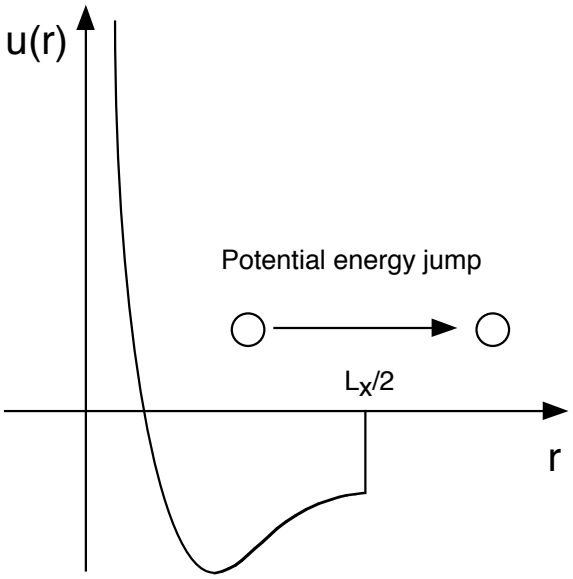
$$\begin{cases} -\frac{L_x}{2} \le x_{ij} < \frac{L_x}{2} \\ -\frac{L_y}{2} \le y_{ij} < \frac{L_y}{2} \\ -\frac{L_z}{2} \le z_{ij} < \frac{L_z}{2} \end{cases}$$

during the computation of forces. In the program, this is achieved by

$$\begin{cases} x_{ij} \leftarrow x_{ij} - \mathrm{SignR}\left(\frac{L_x}{2}, x_{ij} + \frac{L_x}{2}\right) - \mathrm{SignR}\left(\frac{L_x}{2}, x_i - \frac{L_x}{2}\right) \\ y_{ij} \leftarrow y_{ij} - \mathrm{SignR}\left(\frac{L_y}{2}, y_{ij} + \frac{L_y}{2}\right) - \mathrm{SignR}\left(\frac{L_x}{2}, y_{ij} - \frac{L_x}{2}\right) \\ z_{ij} \leftarrow z_{ij} - \mathrm{SignR}\left(\frac{L_z}{2}, z_{ij} + \frac{L_z}{2}\right) - \mathrm{SignR}\left(\frac{L_z}{2}, z_{ij} - \frac{L_z}{2}\right) \end{cases}$$

Shifted Potential

With the minimal image convention, interatomic interaction is cut-off at half the box length. If the box is not large enough, this truncation causes a significant discontinuity in the potential function and our derivation of forces breaks down (we cannot differentiate the potential).



To remedy this, we introduce a cut-off length $r_c < L_x/2$ etc. and modify the potential such that both $u(r)$ and du/dr are continuous at r_c.

$$u'(r) = \begin{cases} u(r) - u(r_c) - (r - r_c)\left.\frac{du}{dr}\right|_{r=r_c} & r < r_c \\ 0 & r \ge r_c \end{cases}$$

For the Lennard-Jones potential, $r_c = 2.5\sigma$ is conventionally used. This causes some shift in the potential minimum but little shift in the minimum position.

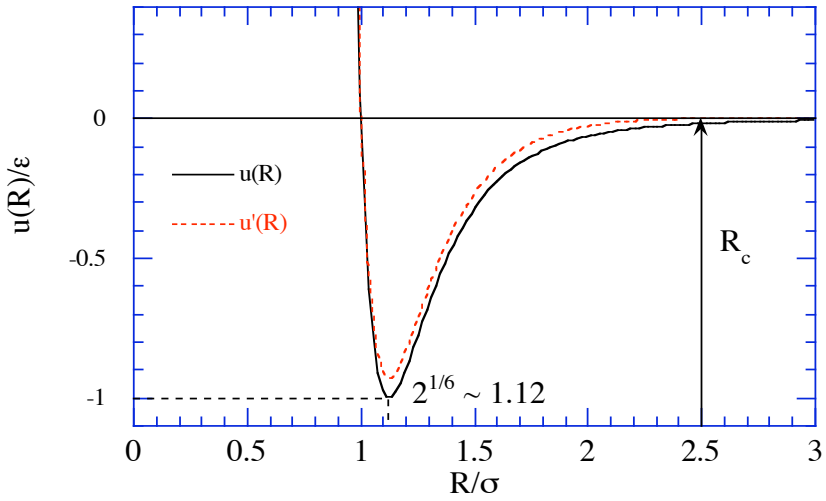


Figure: Truncated Lennard-Jones potential.

Energy

In addition to the potential energy, V, introduced before, we introduce the kinetic energy, K, and the total energy, E, as a sum of both.

$$\begin{aligned} E &= K + V \\ &= \sum_{i=0}^{N-1} \frac{m}{2} \left|\dot{\vec{r}}_i\right|^2 + \sum_{i<j} u(r_{ij}) \end{aligned}$$

where

$$\left|\dot{\vec{r}}_i\right|^2 = \dot{x}_i^2 + \dot{y}_i^2 + \dot{z}_i^2 .$$

ENERGY CONSERVATION

We can prove that the total energy, E, does not change in time, if Newton's equation is exactly integrated.

In order to prove the energy conservation, let us consider

$$\begin{aligned} \dot{E} &= \dot{K} + \dot{V} \\ &= \sum_{i=0}^{N-1} \frac{m}{2} \frac{d}{dt}\left(\dot{x}_i^2 + \dot{y}_i^2 + \dot{z}_i^2\right) + \sum_{i=0}^{N-1} \left(\frac{dx_i}{dt}\frac{\partial V}{\partial x_i} + \frac{dy_i}{dt}\frac{\partial V}{\partial y_i} + \frac{dz_i}{dt}\frac{\partial V}{\partial z_i} \right) \\ &= \sum_{i=0}^{N-1} \frac{m}{2} 2\left(\dot{x}_i\ddot{x}_i + \dot{y}_i\ddot{y}_i + \dot{z}_i\ddot{z}_i\right) + \sum_{i=0}^{N-1} \left(\dot{x}_i\frac{\partial V}{\partial x_i} + \dot{y}_i\frac{\partial V}{\partial y_i} + \dot{z}_i\frac{\partial V}{\partial z_i} \right) \\ &= \sum_{i=0}^{N-1} m\dot{\vec{r}}_i \bullet \ddot{\vec{r}}_i + \sum_{i=0}^{N-1} \dot{\vec{r}}_i \bullet \frac{\partial V}{\partial \vec{r}_i} \\ &= \sum_{i=0}^{N-1} \dot{\vec{r}}_i \bullet \left(m\ddot{\vec{r}}_i - \vec{F}_i \right) \end{aligned}$$

From Newton's equation, the last expression is zero.

In the derivation, vector inner product is defined as

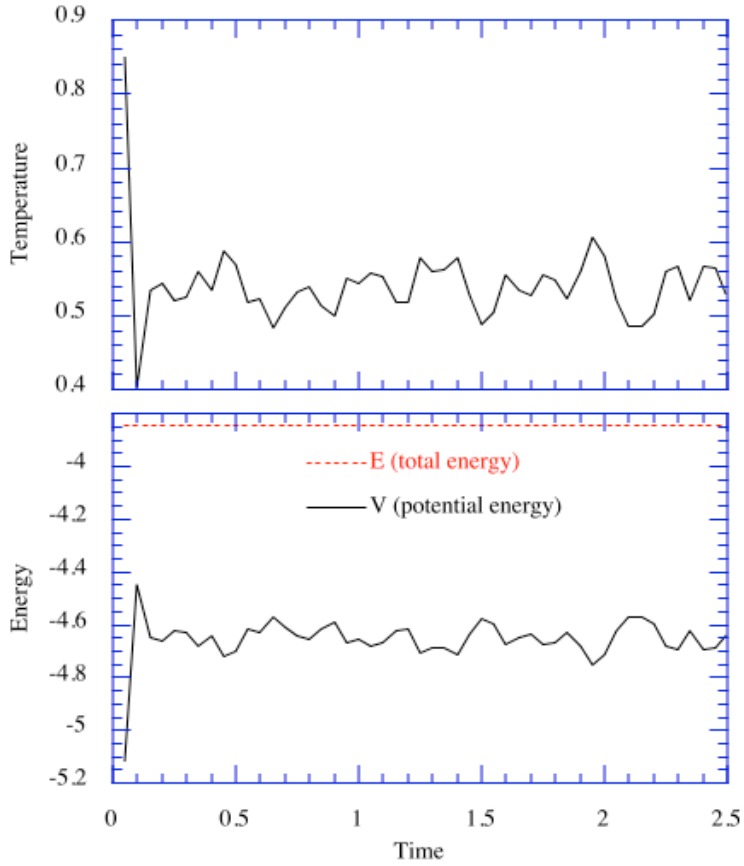
$$\vec{a} \bullet \vec{b} = (a_x, a_y, a_z) \bullet (b_x, b_y, b_z) = a_x b_x + a_y b_y + a_z b_z$$

and note that

$$\frac{d}{dt}(fg) = \dot{f}g + f\dot{g}.$$

With discretization approximation, the total energy is not rigorously conserved anymore. How well the total energy is conserved is a good measure for how small Δ we should use.

Δ = 0.005 in the normalized unit is typically used for the Lennard-Jones potential (see the figure below).



Temperature

Temperature, *T*, is related to the kinetic energy as follows:

$$\frac{3Nk_BT}{2} = \sum_{i=0}^{N-1} \frac{m}{2} \left| \dot{\vec{r}}_i \right|^2,$$

where

k_B = 1.38062 × 10^{-16} erg/Kelvin—Boltzmann constant

By using the normalized coordinates,

$$\frac{3Nk_BT}{2} = m\sigma^2 \bullet \frac{\varepsilon}{\sigma^2 m} \sum_{i=0}^{N-1} \frac{m}{2} \left| \dot{\vec{r}}_i' \right|^2$$

$$\therefore \frac{T}{\varepsilon / k_B} = \frac{1}{3N} \sum_{i=0}^{N-1} \left| \vec{r}_i' \right|^2$$

where

ε/k_B = 120 Kelvin

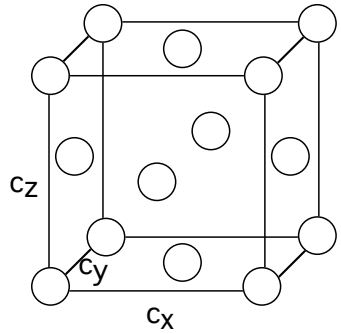
is the Lennard-Jones temperature unit. So the average temperature 0.5 means 60 K << 273 K = 0°C.

Initialization

The program starts to initialize the coordinates and velocities of all atoms.

FACE CENTERED CUBIC (FCC) LATTICE

Atoms are initially arranged to form a regular lattice. Namely they occupy all the corners and face centers of a cube called a unit cell.



`double gap[3]`: (c_x, c_y, c_z), the edge lengths of the unit cell

Since our unit cell is cubic, $c_x = c_y = c_z = c$.

Each unit cell contains

$$\frac{1}{8} \times 8(\text{corners}) + \frac{1}{2} \times 6(\text{faces}) = 4\,\text{atoms},$$

and the number density of atoms is given by $\rho = 4/c^3$, or

$$c = (4/\rho)^{1/3}.$$

The four coordinates are given (in unit of c) as: (0, 0, 0), (0, 1/2, 1/2), (1/2, 0, 1/2), (1/2, 1/2, 0). These four coordinates are stored in

`double origAtom[4][3]`

The simulated system is constructed by repeating the unit cells.

`int InitUcell[3]`: Stores the number of unit cells in the x, y, and z directions

The total number of atoms is therefore given by

$$N = 4 \times \texttt{InitUcell[0]} \times \texttt{InitUcell[1]} \times \texttt{InitUcell[2]}$$

RANDOM VELOCITY

We generate random velocities of magnitude

$$\frac{v_0^2}{3} = T_{init} \Rightarrow v_0 = \sqrt{3T_{init}}$$

`double InitTemp`: Initial temperature

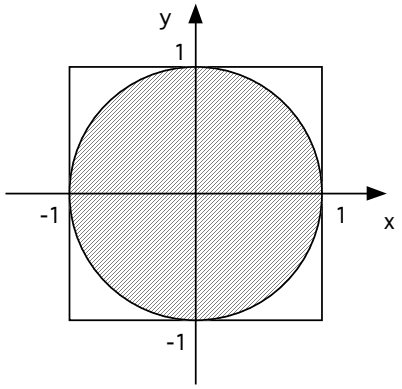
For each atom, the velocity vector is then given by

$$\vec{v}_i = v_0(\xi_0, \xi_1, \xi_2) = v_0\vec{\xi}$$

where $\vec{\xi}$ is a rondomly oriented vector of unit length.

RANDOM POINT IN A UNIT CIRCLE

The question is how to randomly generate a point on a unit sphere. We first start from a simpler problem, random point in a unit circle.

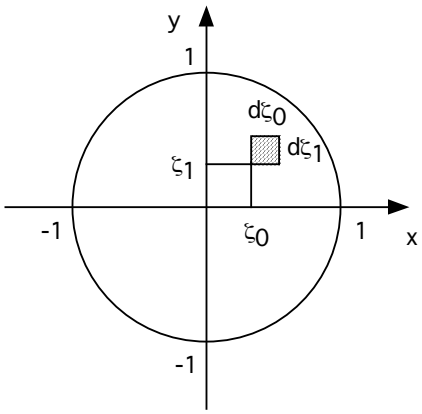


(Algorithm)

Let `rand()` be a uniform random-number generator in the range [0, 1]

1. $\zeta_i = 2*$`rand()` $- 1$ (i = 0, 1) so that $-1 \le \zeta_i < 1$
2. $s^2 = \zeta_0^2 + \zeta_1^2$
3. if $s^2 < 1$, accept $\vec{\xi} = (\xi_0, \xi_1)$
4. else reject and go to 1

(Probability distribution)



Probability to find a point in the shaded area

= (# of points in the shaded area)/(total # of generated points)

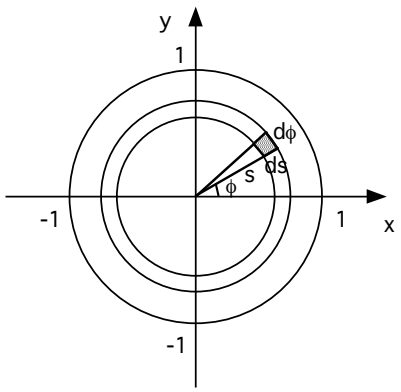
= $P(\zeta_0, \zeta_1)d\zeta_0 d\zeta_1$

= ($d\zeta_0 d\zeta_1$ ~ shaded area)/($\pi \bullet 1^2$ ~ area of unit circle), if there is no bias

$$\therefore P(\xi_0, \xi_1) = 1/\pi$$

(Polar coordinate system)

$$(\xi_0, \xi_1) = (s\cos\varphi, s\sin\varphi) \quad 0 \le s < 1; 0 \le \varphi < 2\pi$$



Again the probability density is given by the ratio of the shaded area divided by the total circular area.

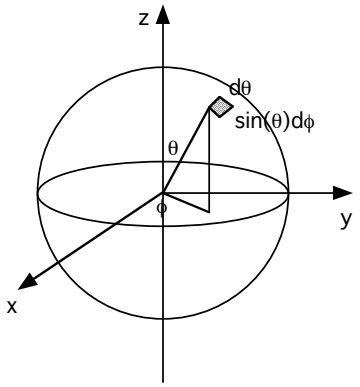
$$P(s,\varphi)dsd\varphi = \frac{ds \bullet sd\varphi}{\pi \bullet 1^2} \text{ (if uniform)}$$

$$\therefore P(s,\varphi) = \frac{s}{\pi} \tag{1}$$

RANDOM POINT ON A UNIT SPHERE

The above distribution is closely related to the uniform distribution on a unit sphere.

(Spherical coordinate)



$$(\xi_0,\xi_1,\xi_2) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta) \quad 0 \le \theta < \pi, 0 \le \varphi < 2\pi$$

(Probability distribution)

$$P(\theta,\varphi)d\theta d\varphi = \frac{d\theta \bullet \sin\theta d\varphi}{4\pi \bullet 1^2}$$

$$\therefore P(\theta,\varphi) = \frac{\sin\theta}{4\pi} \tag{2}$$

If we identify $s \equiv \sin\frac{\theta}{2}$, Eqs. (1) and (2) are equivalent.

$$\because 0 \le \theta < \pi \Leftrightarrow 0 \le \frac{\theta}{2} < \frac{\pi}{2} \Leftrightarrow 0 \le s \equiv \sin\frac{\theta}{2} < 1$$

$$\frac{sdsd\varphi}{\pi} = \frac{\sin\frac{\theta}{2}\frac{ds}{d\theta}d\theta d\varphi}{\pi} = \frac{\sin\frac{\theta}{2}\left(\frac{1}{2}\cos\frac{\theta}{2}\right)d\theta d\varphi}{\pi} = \frac{2\sin\frac{\theta}{2}\cos\frac{\theta}{2}d\theta d\varphi}{4\pi} = \frac{\sin\theta d\theta d\varphi}{4\pi} //$$

Note that

$$\begin{cases} \xi_0 = \sin\theta\cos\varphi = 2\sin\dfrac{\theta}{2}\cos\dfrac{\theta}{2}\cos\varphi = 2s\sqrt{1-s^2}\,\dfrac{\zeta_0}{s} = 2\sqrt{1-s^2}\,\zeta_0 \\ \xi_1 = \sin\theta\sin\varphi = 2\sin\dfrac{\theta}{2}\cos\dfrac{\theta}{2}\sin\varphi = 2s\sqrt{1-s^2}\,\dfrac{\zeta_1}{s} = 2\sqrt{1-s^2}\,\zeta_1 \\ \xi_2 = \cos\theta = 1-2\sin^2\dfrac{\theta}{2} = 1-2s^2 \end{cases}$$

(Algorithm—Generating random points on a unit sphere)

Let `rand()` be a uniform random-number generator in the range [0, 1]

1. $\zeta_i = 2*$`rand()` $- 1$ (i = 0, 1) so that $-1 \le \zeta_i < 1$
2. $s^2 = \zeta_0^2 + \zeta_1^2$
3. if $s^2 < 1$, accept $(\xi_0,\xi_1,\xi_2) = (2\sqrt{1-s^2}\,\zeta_0, 2\sqrt{1-s^2}\,\zeta_1, 1-2s^2)$
4. else reject and go to 1

LINEAR CONGRUENTIAL RANDOM NUMBER GENERATOR

Sequence of seemingly random integers between 1 and m-1 is obtained by starting from some nonzero I_1 and

$$I_{j+1} = aI_j \pmod{m}$$

We choose

$m = 2^{31}-1 = 2147483647$

$a = 16807$

Uniform random number in the range [0, 1] is obtained as

$$r_j = I_j / m .$$

Molecular Dynamics III: $O(N)$ Linked-List Cell Algorithm

This chapter explains the linked-list cell MD algorithm, the computational time of which scales as $O(N)$ for N atoms. We will replace function `computeAccel()` in the $O(N^2)$ program `md.c` by this algorithm.

Recall that the naive double-loop implementation to compute pair interactions scales as $O(N^2)$. In fact, with a finite cut-off length, r_c (see `#define RCUT 2.5` in `md.c`), an atom interacts with only a limited number of atoms $\sim (4\pi/3)r_c^3(N/V)$, where V is the volume of the simulation box. The linked-list cell algorithm explained below computes the entire interaction with $O(N)$ operations.

CELLS

First divide the simulation box into small cells of equal size. The edge lengths of each cell, (r_{cx}, r_{cy}, r_{cz}) (`double rc[3]`), must be at least r_c; we use $r_{c\alpha} = L_\alpha/L_{c\alpha}$, where $L_{c\alpha} = \lfloor L_\alpha/r_c \rfloor$ (α = x, y, z) and L_α is the simulation box length in the α direction (`double Region[3]`). Here $\lfloor x \rfloor$ is the floor function, i.e., the largest integer that is less than or equal to x. An atom in a cell interacts with only the other atoms in the same cell and its 26 neighbor cells. The number of cells to accommodate all these atoms is $L_{cx}L_{cy}L_{cz}$, where $L_{c\alpha} = L_\alpha/r_{c\alpha}$ (α = x, y, z) (`int lc[3]`). We identify a cell with a vector cell index, $\vec{c} = (c_x, c_y, c_z)$ ($0 \le c_x \le L_{cx}-1$; $0 \le c_y \le L_{cy}-1$; $0 \le c_z \le L_{cz}-1$), and a serial cell index (see the figure below),

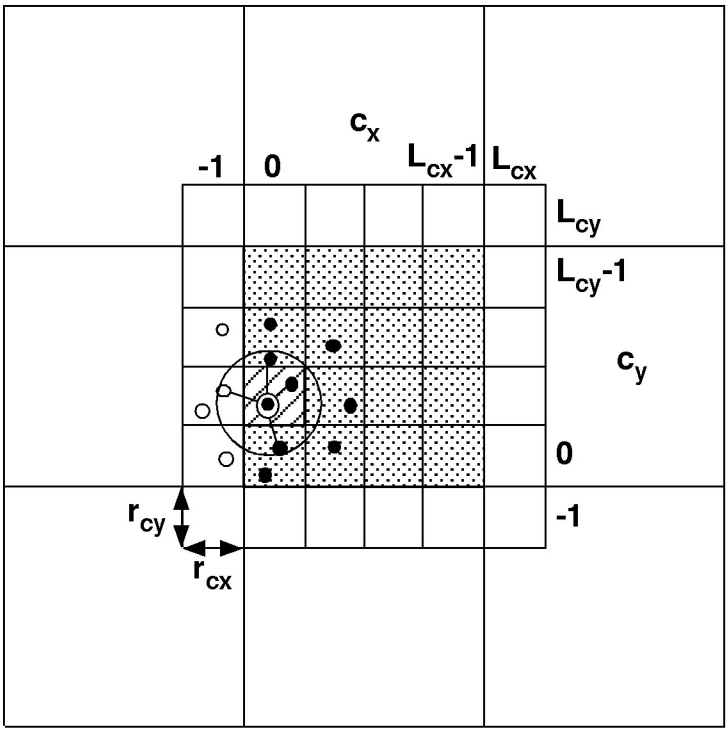
$$c = c_x L_{cy} L_{cz} + c_y L_{cz} + c_z$$

or

$$c_x = c/(L_{cy}L_{cz})$$
$$c_y = (c/L_{cz}) \bmod L_{cy}$$
$$c_z = c \bmod L_{cz}.$$

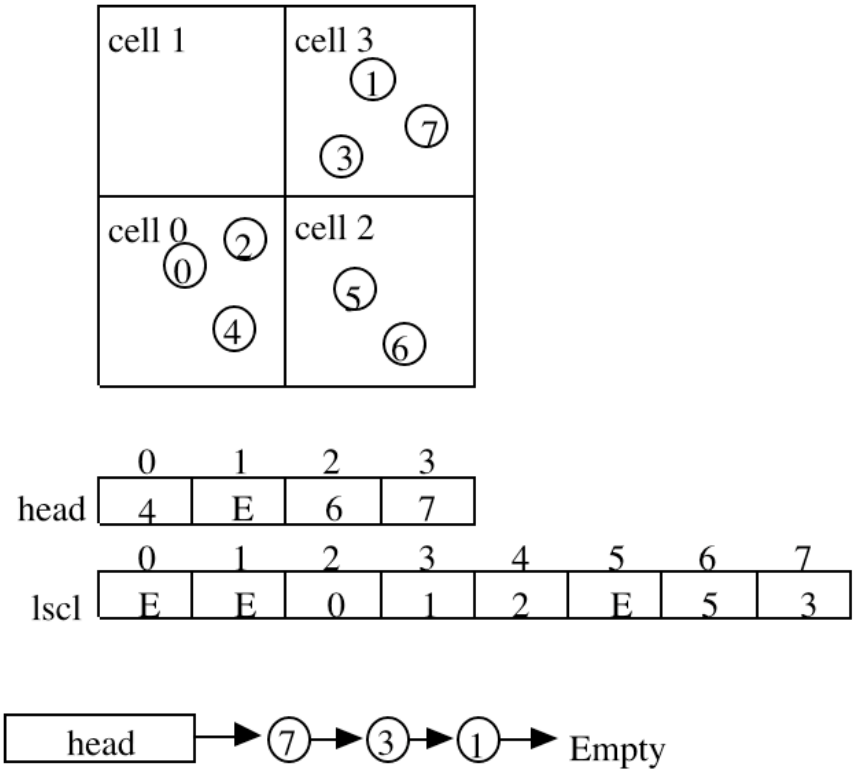
An atom with coordinate $\vec{r}$ belongs to a cell with the vector cell index,

$$c_\alpha = \lfloor r_\alpha/r_{c\alpha} \rfloor \ (\alpha = \text{x, y, z}).$$



LISTS

The atoms belonging to a cell is organized using linked lists. The following describes the data structures and algorithms to construct the linked lists and compute the interatomic interaction using them.



DATA STRUCTURES

`lscl[NMAX]`: An array implementation of the linked lists. `lscl[i]` holds the atom index to which the *i*-th atom points.

`head[NCLMAX]`: `head[c]` holds the index of the first atom in the *c*-th cell, or `head[c]` = `EMPTY` (= –1) if there is no atom in the cell.

ALGORITHM 1: LIST CONSTRUCTOR

```
/* Reset the headers, head */
for (c=0; c<lcxyz; c++) head[c] = EMPTY;
/* Scan atoms to construct headers, head, & linked lists, lscl */
for (i=0; i<nAtom; i++) {
  /* Vector cell index to which this atom belongs */
  for (a=0; a<3; a++) mc[a] = r[i][a]/rc[a];
  /* Translate the vector cell index, mc, to a scalar cell index */
  c = mc[0]*lcyz+mc[1]*lc[2]+mc[2];
  /* Link to the previous occupant (or EMPTY if you're the 1st) */
  lscl[i] = head[c];
  /* The last one goes to the header */
  head[c] = i;
}
```

where `lcyz = lc[1]*lc[2]; lcxyz = lcyz*lc[0]`.

ALGORITHM 2: INTERACTION COMPUTATION

```
/* Scan inner cells */
for (mc[0]=0; mc[0]<lc[0]; (mc[0])++)
for (mc[1]=0; mc[1]<lc[1]; (mc[1])++)
for (mc[2]=0; mc[2]<lc[2]; (mc[2])++) {
  /* Calculate a scalar cell index */
  c = mc[0]*lcyz+mc[1]*lc[2]+mc[2];
  /* Scan the neighbor cells (including itself) of cell c */
  for (mc1[0]=mc[0]-1; mc1[0]<=mc[0]+1; (mc1[0])++)
  for (mc1[1]=mc[1]-1; mc1[1]<=mc[1]+1; (mc1[1])++)
  for (mc1[2]=mc[2]-1; mc1[2]<=mc[2]+1; (mc1[2])++) {
    /* Periodic boundary condition by shifting coordinates */
    for (a=0; a<3; a++) {
      if (mc1[a] < 0)
        rshift[a] = -Region[a];
      else if (mc1[a]>=lc[a])
        rshift[a] = Region[a];
      else
        rshift[a] = 0.0;
    }
    /* Calculate the scalar cell index of the neighbor cell */
    c1 = ((mc1[0]+lc[0])%lc[0])*lcyz
        +((mc1[1]+lc[1])%lc[1])*lc[2]
        +((mc1[2]+lc[2])%lc[2]);
    /* Scan atom i in cell c */
    i = head[c];
    while (i != EMPTY) {
      /* Scan atom j in cell c1 */
      j = head[c1];
      while (j != EMPTY) {
        if (i < j) { /* Avoid double counting of pair (i, j) */
          r_ij = r_i-(r_j+r_shift); /* Image-corrected relative pair position */
          if (r_ij < r_c^2)
            Compute forces on pair (I, j)
          ...
        }
        j = lscl[j];
      }
      i = lscl[i];
    }
  }
}
```

A neighbor cell can be outside of the central simulation box, i.e., the cell-index vector component c_α can be -1 or L_α (α = x, y, z). To find the atoms that belong to such an outside cell, the modulo operator (%) pulls back the cell index into the range $[0, L_\alpha-1]$ (α = x, y, z):

```
c1 = ((mc1[0]+lc[0])%lc[0])*lcyz
   + ((mc1[1]+lc[1])%lc[1])*lc[2]
   + ((mc1[2]+lc[2])%lc[2]);
```

The atomic positions in an outside cell, on the other hand, must be treated as they are and must not be pulled back into the central simulation box. The shift vector $\vec{r}_{\text{shift}}$ (`double rshift[3]`) to shift the central image of such an atom to the appropriate image position by a simulation box length, if necessary:

```
if (mc1[a] < 0) rshift[a] = -Region[a]; else if (mc1[a]>=lc[a]) rshift[a] =
Region[a]; else rshift[a] = 0.0;
```

Molecular Dynamics IV: Data Locality

The linked-list cell algorithm is one of the many algorithms that expose data locality inherent in molecular dynamics (MD) simulations. Data locality is either spatial or temporal:

- **Spatial locality**: Atoms closer to each other interact more tightly. For example, interatomic potentials with a finite cut-off radius, the force acting on each atom depends only on the positions of its finite near-neighbor atoms. Even long-range interatomic interactions without cut-off (e.g., the electrostatic or Coulombic interaction between charged particles) can often be reformulated using a hierarchy of length scales such that, at each length scale, interaction is finite ranged.[4]
- **Temporal locality**: Computations performed in consecutive MD time steps are similar. For example, the list of the near-neighbor atoms for a given atom may not change for several time steps. Also, certain forces, especially those due to farther atoms, tend to change slowly in time, so that we can reuse previously computed forces for several time steps.[5]

Data locality plays a central role in developing efficient simulation algorithms. It not only allows us to reduce the amount of computation but also causes extensive processing of data elements that reside close to each other in space. With proper layout of data within memory and that of computation, this results in better utilization of hardware memory hierarchy (e.g., reduction of cache misses). In parallel computing with proper data/computation decomposition, this also minimizes the communication overhead and the enhancement of the parallel efficiency.

The hardest computation in MD simulations is the evaluation of long-range electrostatic forces, which requires $O(N^2)$ operations. For example, the multiresolution molecular dynamics (MRMD) algorithm[6,7] has been developed to reduce this $O(N^2)$ complexity to $O(N)$, by making use of spatial localities based on the fast multipole method (FMM) by Greengard and Rokhlin.[4]

The first essential idea of the FMM is clustering, i.e., instead of computing interactions between all atomic pairs, atoms are clustered and cluster-cluster interactions are computed (see the Figure below). At the source of interaction, cluster information is encapsulated in terms of the multipoles of the charge distribution with a well-defined error bound. At the destination, the electrostatic potential is expanded in terms of local terms, which is similar to the Taylor expansion. The second essential idea is to use larger clusters for longer distances, in order to reduce the computation and keep the error constant. This is achieved by recursively subdividing the simulation box into smaller cells to form an octree data structure. The $O(N)$ algorithm traverses this tree twice. In the upward pass, multipoles are computed for all cells at all levels. First the multipoles of the leaf cells are computed using atomic charges and coordinates, and the multipoles of these children cells are shifted and combined to obtain the multipoles of the parent cells. This procedure is repeated until the root of the octree is reached. In the downward pass, these multipoles are translated to local terms for all cells at all levels, starting from the root. For a given cell at each level, only the multipoles of a constant number of interactive cells contribute to the local terms. Contributions from farther cells have already been computed at the previous coarse level, and they are inherited from the parent cell. On the other hand, the contributions from the nearest-neighbor cells will be computed at the next fine level. This procedure is repeated until we reach the leaf level. Finally, the nearest-neighbor-cell contributions at the leaf level are computed by direct summation over atoms. Since constant computation is performed at each of the $O(N)$ octree nodes, the complexity of the FMM algorithm is $O(N)$. A scalable and portable parallel FMM algorithm, which has the unique capability to compute stress tensors with a novel complex charge method, is freely available from an open source public library.[8]

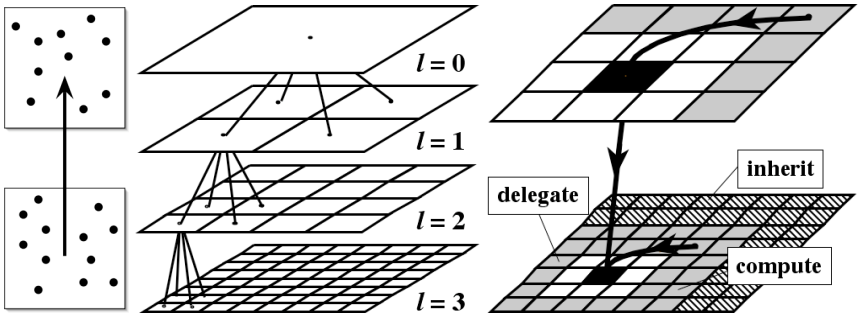


Figure. Schematic of the FMM. (Left) Atoms (dots) are clustered (squares), and cluster-cluster interactions are computed. (Center) Two-dimensional example of hierarchical clustering, in which the entire system at level $l = 0$ is recursively divided into subsystems. (Right) For a given cell at an octree level (shown in black in the bottom panel), only the multipoles from a constant number of neighbor cells (shown in gray in the bottom panel) are translated to the local terms. Contributions from farther cells (hatched) have already been computed at the previous coarse level, and they are inherited from the parent cell (black in the top panel). On the other hand, translation of the multipoles from the nearest-neighbor cells (white in the bottom panel) will be delegated to the children cells at the next fine level.

The MRMD algorithm[6,7] also uses multiple time stepping (MTS)[5] to take advantage of temporal localities. The MTS uses different force-update schedules for different force components, i.e., forces from the nearest-neighbor atoms are computed at every MD step, and forces from farther atoms are computed with less frequency. This not only reduces the computational cost but also enhances the data locality, and accordingly the parallel efficiency is increased. These different force components are combined using a reversible symplectic integrator,[5] and the resulting algorithm consists of nested loops to use forces from different spatial regions. It has been proven that the phase-space volume occupied by atoms is a simulation-loop invariant in this algorithm, and this loop invariant results in excellent long-time stability of the solutions.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford U. Press, Oxford, 1987).
2. D.C. Rapaport, *The Art of Molecular Dynamics Simulation* (Cambridge Univ. Press, Cambridge, 1995).
3. D. Frenkel and B. Smit, *Understanding Molecular Simulation, 2nd Ed.* (Academic Press, New York, 2001).
4. L. Greengard and V. Rokhlin, "a fast algorithm for particle simulations," *J. Comput. Phys.* **73**, 325 (1987).
5. M. Tuckerman, B. J. Berne, and G. J. Martyna, "reversible multiscale molecular dynamics," *J. Chem. Phys.* **97**, 1990 (1992).
6. A. Nakano, R. K. Kalia, and P. Vashishta, "multiresolution molecular dynamics algorithm for realistic materials modeling on parallel computers," *Comput. Phys. Commun.* **83**, 197 (1994).
7. A. Nakano, R. K. Kalia, P. Vashishta, T. J. Campbell, S. Ogata, F. Shimojo, and S. Saini, "scalable atomistic simulation algorithms for materials research," *Scientific Programming* **10**, 263 (2002).
8. S. Ogata, T. J. Campbell, R. K. Kalia, A. Nakano, P. Vashishta, and S. Vemparala, "scalable and portable implementation of the fast multipole method on parallel computers," *Comput. Phys. Commun.* **153**, 445 (2003).