2

Basics of molecular dynamics

2.1 Equations of motion for MD simulations

The classical MD simulations boil down to numerically integrating Newton’s equations of motion for the particles (atoms, in the simplest case) which build up the investigated system:

$$m \frac{d^2 \mathbf{r}_i}{dt^2} = \mathbf{F}_i(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N), \quad i = 1, 2, \ldots, N. \quad (2.1)$$

Here $\mathbf{r}_i$ are the position vectors and $\mathbf{F}_i$ are the forces acting upon the $N$ particles in the system.

Quite often forces derive from potential functions, $U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$, representing the potential energy of the system for the specific geometric arrangement of the particles:

$$\mathbf{F}_i(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = -\nabla_{\mathbf{r}_i} U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N).$$

This form implies the conservation of the total energy $E = E_{\text{kin}} + U$, where $E_{\text{kin}}$ is the instantaneous kinetic energy.

In the absence of external forces, the potential can be represented in the simplest case as a sum of pairwise interactions:

$$U = \sum_{i=1}^{N} \sum_{j>i}^{N} u(\mathbf{r}_{ij}). \quad (2.2)$$
where $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$, $r_{ij} \equiv |\mathbf{r}_{ij}|$, and the condition $j > i$ prevents the double counting of the particle pairs. The forces acting on the particles are composed in such a case of the individual interactions with the rest of the particles:

$$
\mathbf{F}_i = \sum_{j \neq i}^N \mathbf{f}_{ij}, \quad \mathbf{f}_{ij} = -\frac{du(r_{ij})}{dr_{ij}} \cdot \frac{\mathbf{r}_{ij}}{r_{ij}}. \tag{2.3}
$$

According to Newton’s third law, $\mathbf{f}_{ji} = -\mathbf{f}_{ij}$. The computational effort to solve the set of equations of motion (2.1) is proportional to $N^2$ and is mainly associated with the evaluation of forces. Therefore, for tractable computations the forces should be expressed analytically. To further reduce the computational effort the potential can be cut off at some limiting separation (for $r_{ij} > r_{\text{cut}}$) beyond which the potential becomes negligible.

### 2.2 The Lennard-Jones potential

One of the most famous pair potentials for van der Waals systems is the **Lennard-Jones potential**:

$$
u_{\text{LJ}}(r_{ij}) = 4\varepsilon \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \left( \frac{\sigma}{r_{ij}} \right)^6 \right] \tag{2.4}$$

and it was initially proposed for liquid argon. The parameter $\varepsilon$ governs the strength of the interaction and $\sigma$ defines a length scale. This potential is strongly repulsive at short distances and it passes through 0 at $r_{ij} = \sigma$, $\nu_{\text{LJ}}(\sigma) = 0$. It reaches its minimum $\nu_{\text{LJ}}(r_m) = \varepsilon$ at $r_m = 2^{1/6} \sigma \simeq 1.225\sigma$ and it exhibits an attractive tail at large $r_{ij}$. The parameters $\varepsilon$ and $\sigma$ are chosen to fit certain physical properties of the system.

The term proportional to $r_{ij}^{-12}$, dominating at short distances, models the repulsion due to the nonbonded overlap of the electronic orbitals. It has a rather arbitrary form and other exponents or even other functional forms are sometimes preferred. As a matter of fact, on physical grounds, an exponential term would be more appropriate, but would imply one more parameter and a slightly larger computational effort.

The attractive term proportional to $r_{ij}^{-6}$ dominates at large distances and it models the van der Waals dispersion forces caused by the dipole-dipole interactions due to fluctuating dipoles. These weak forces provide the bonding character of closed-shell systems (rare gases, such as Ar or Kr).
2.2 The Lennard-Jones potential

The Lennard-Jones potential is totally inadequate for open shell systems, in which strong localized bonds are formed.

The inter-particle forces arising from the Lennard-Jones potential (2.4) have the form:

$$f_{ij} = \frac{48\varepsilon}{r_{ij}^2} \left[ \left( \frac{\sigma}{r_{ij}} \right)^{12} - \frac{1}{2} \left( \frac{\sigma}{r_{ij}} \right)^6 \right] r_{ij},$$

(2.5)

Here is a typical routine for computing the interactions:

```fortran
!============================================================================
subroutine Forces(x,y,z,fx,fy,fz,natm,Epot)
!----------------------------------------------------------------------------
! Energies in eV, forces in eV/Angstroem, distances in Angstroem
implicit real(8) (a-h,o-z)
parameter (epsilon = 1.d0, sigma = 1.25d0, sigma2 = sigma * sigma)
parameter (Rcut = 2.d0, Rcut2 = Rcut * Rcut)
real(8) x(natm), y(natm), z(natm)
real(8) fx(natm), fy(natm), fz(natm)
do iatm = 1,natm
  fx(iatm) = 0.d0; fy(iatm) = 0.d0; fz(iatm) = 0.d0
end do
Epot = 0.d0
do iatm = 1,natm
  do jatm = iatm+1,natm
    dx = x(iatm) - x(jatm)
dy = y(iatm) - y(jatm)
dz = z(iatm) - z(jatm)
r2 = dx*dx + dy*dy + dz*dz
    if (r2 < Rcut2) then
      fr2 = sigma2 / r2
      fr6 = fr2 * fr2 * fr2
      fpr = 48.d0 * epsilon * fr6 * (fr6 - 0.5d0) / r2
      fxi = fpr * dx
      fyi = fpr * dy
      fzi = fpr * dz
      fx(iatm) = fx(iatm) + fxi; fx(jatm) = fx(jatm) - fxi
      fy(iatm) = fy(iatm) + fyi; fy(jatm) = fy(jatm) - fyi
      fz(iatm) = fz(iatm) + fzi; fz(jatm) = fz(jatm) - fzi
    end if
  end do
Epot = Epot + 4.d0 * epsilon * fr6 * (fr6 - 1.d0)
end do
!============================================================================
```}

Especially when modelling fluids, it is convenient to work in dimensionless or reduced units, also called MD units. Among the advantages of using such units the following are worth mentioning:
the possibility to work with numerical values of the order of unity, instead of the typically very small values associated with the atomic scale;

• the simplification of the equations of motion, due to the absorption of the parameters defining the model into the units;

• the possibility of scaling the results for a whole class of systems described by the same model.

When using Lennard-Jones potentials in simulations, the most appropriate system of units adopts $\sigma$, $m$ and $\varepsilon$ as units of length, mass and energy, respectively, and implies making the replacements:

$$
\begin{aligned}
    r & \rightarrow r\sigma \\
    E & \rightarrow E\varepsilon \\
    t & \rightarrow t\sigma(m/\varepsilon)^{1/2}.
\end{aligned}
$$

The equations of motion in MD units are:

$$
\ddot{\mathbf{r}}_i = \sum_{j \neq i}^N f_{ij}, \quad i = 1, 2, \ldots, N \quad (2.6)
$$

and the interaction forces take the simplified form:

$$
f_{ij} = 48 \left( \frac{r_{ij}^{-14}}{r_{ij}^{-6}} - \frac{1}{2} r_{ij}^{-8} \right) \mathbf{r}_{ij}. \quad (2.7)
$$

The kinetic and the potential energies are given by:

$$
\begin{aligned}
    E_{\text{kin}} &= \frac{1}{2} \sum_{i=1}^{N} \mathbf{v}_{i}^2, \quad (2.8) \\
    E_{\text{pot}} &= 4 \sum_{i<j}^{N} \left( \frac{r_{ij}^{-12}}{r_{ij}^{-6}} \right). \quad (2.9)
\end{aligned}
$$

Since each translational degree of freedom contributes to the total kinetic energy by $k_B T/2$, the temperature of the system may be defined by:

$$
T = \frac{2k_B E_{\text{kin}}}{3N}. \quad (2.10)
$$
2.3 Thermodynamic properties

In MD units $k_B = 1$ is considered. The denominator should be, strictly speaking, $3N - 3$, accounting for the three global degrees of freedom, which have to be eliminated to ensure momentum conservation.

In Table 2.1 are listed, besides the definitions of the MD units for various physical quantities of interest in simulations, the corresponding numerical values for liquid argon.

<table>
<thead>
<tr>
<th>Physical quantity</th>
<th>Unit</th>
<th>Value for Ar</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>$\sigma$</td>
<td>$3.4 \times 10^{-10}$ m</td>
</tr>
<tr>
<td>energy</td>
<td>$\varepsilon$</td>
<td>$1.65 \times 10^{-21}$ J</td>
</tr>
<tr>
<td>mass</td>
<td>$m$</td>
<td>$6.69 \times 10^{-26}$ kg</td>
</tr>
<tr>
<td>time</td>
<td>$\sigma(m/\varepsilon)^{1/2}$</td>
<td>$2.17 \times 10^{-12}$ s</td>
</tr>
<tr>
<td>velocity</td>
<td>$(\varepsilon/m)^{1/2}$</td>
<td>$1.57 \times 10^{2}$ m/s</td>
</tr>
<tr>
<td>force</td>
<td>$\varepsilon/\sigma$</td>
<td>$4.85 \times 10^{-12}$ N</td>
</tr>
<tr>
<td>pressure</td>
<td>$\varepsilon/\sigma^3$</td>
<td>$4.20 \times 10^{7}$ N·m$^{-2}$</td>
</tr>
<tr>
<td>temperature</td>
<td>$\varepsilon/k_B$</td>
<td>120 K</td>
</tr>
</tbody>
</table>

2.3 Thermodynamic properties

Considering the temperature $T$ and the density $\rho$ as independent variables, the energy $E$ and the pressure $P$ can be readily expressed. These quantities provide the link between the microscopic and the macroscopic level and their measurement in an MD simulation is straightforward. It should be noted, however, that in an MD simulation the energy is constant, rather than the temperature, and as such the average temperature $\langle T \rangle$ should be used, rather than $T$ itself.

Pressure is given by the virial formula:

$$PV = Nk_B T + \frac{1}{3} \left\langle \sum_{i=1}^{N} \mathbf{r}_i \cdot \mathbf{F}_i \right\rangle. \quad (2.11)$$

In the case of a pair potential, this formula changes to:

$$PV = Nk_B T + \frac{1}{3} \left\langle \sum_{i<j}^{N} \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \right\rangle. \quad (2.12)$$
2. Basics of molecular dynamics

Having in view relation (2.10) between the temperature $T$ and the kinetic energy of the atoms $E_{\text{kin}}$, we can express the pressure as:

$$P = \frac{\rho}{3N} \left(2E_{\text{kin}} + \sum_{i<j} N \mathbf{r}_{ij} \cdot \mathbf{f}_{ij} \right)$$

(2.13)

where solely quantities specific to the simulation appear.

As opposed to the total energy, $E_{\text{tot}} = E_{\text{kin}} + E_{\text{pot}}$, which should be conserved during a simulation (apart from numerical integration errors), the temperature and the pressure fluctuate and, therefore, they should be averaged over a certain number of timesteps.

2.4 Time integration

The most time-consuming component of an MD calculation — evaluation of forces.

Any method requiring more than one force evaluation / time step is inefficient.

Classes of MD integrators:

- low-order methods — leapfrog, Verlet, velocity Verlet — easy implementation, stability
- predictor-corrector methods — high accuracy for large time-steps.

"Velocity" Verlet method

For the motion of a particle:

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \mathbf{a}(t)\Delta t + O((\Delta t)^2)$$

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)(\Delta t)^2 + O((\Delta t)^3)$$

Expressing the speed at mid-interval $\mathbf{v}(t + \Delta t/2)$:

$$\begin{align*}
\mathbf{v}(t + \Delta t/2) &= \mathbf{v}(t) + \mathbf{a}(t)\Delta t/2 \\
\mathbf{r}(t + \Delta t) &= \mathbf{r}(t) + \mathbf{v}(t + \Delta t/2)\Delta t \\
\mathbf{a}(t + \Delta t) &= \mathbf{F}(\mathbf{r}(t + \Delta t))/m \\
\mathbf{v}(t + \Delta t) &= \mathbf{v}(t + \Delta t/2) + \mathbf{a}(t + \Delta t)\Delta t/2
\end{align*}$$

Algebraically equivalent to the leapfrog method.

Advantages:
Predictor-corrector methods

- conservation of energy,
- higher stability than for predictor-corrector methods,
- lower memory requirements.

Subroutine:

```fortran
!============================================================================
subroutine Verlet(amass,x,y,z,vx,vy,vz,ax,ay,az,natm,dt,Ekin,Epot)
!----------------------------------------------------------------------------
implicit real(8) (a-h,o-z)
real(8) x(natm), y(natm), z(natm) ! x[Angs]
real(8) vx(natm), vy(natm), vz(natm) ! v[Angs/psec]
real(8) ax(natm), ay(natm), az(natm) ! a[Angs/psec*psec]
d t2 = 0.5d0 * dt
do iatm = 1,natm
   vx(iatm) = vx(iatm) + ax(iatm) * dt2 ! v(t+dt/2)
   vy(iatm) = vy(iatm) + ay(iatm) * dt2
   vz(iatm) = vz(iatm) + az(iatm) * dt2
   x(iatm) = x(iatm) + vx(iatm) * dt ! r(t+dt)
   y(iatm) = y(iatm) + vy(iatm) * dt
   z(iatm) = z(iatm) + vz(iatm) * dt
end do

! forces

!kinetic energy

Ekin = 0.d0
do iatm = 1,natm
   ax(iatm) = ax(iatm) / amass ! a(t+dt)
   ay(iatm) = ay(iatm) / amass
   az(iatm) = az(iatm) / amass
   vx(iatm) = vx(iatm) + ax(iatm) * dt2 ! v(t+dt)
   vy(iatm) = vy(iatm) + ay(iatm) * dt2
   vz(iatm) = vz(iatm) + az(iatm) * dt2
   Ekin = Ekin + vx(iatm)**2 + vy(iatm)**2 + vz(iatm)**2
end do
Ekin = 0.5d0 * amass * Ekin
end
```

2.5 Predictor-corrector methods

Multiple-value methods — use information from one or several earlier time steps.

- **multistep methods** — use the acceleration at a series of previous time steps — the Adams approach

- **higher derivatives methods** — use higher accelerations at the current time step — the Nordsieck method.
Algebraically equivalent for methods of the same order in \( h \).

Higher order than the Verlet method — require extra computations and storage. We consider multistep methods — disadvantage: stepsize \( h \) cannot be changed easily.

The method implies two steps:

- a **predictor** step providing an initial approximation to the propagated solution
- a **corrector** step yielding a refined approximation.

Consider the second-order differential equation:

\[
\frac{d^2x}{dt^2} = f(t, x, \frac{dx}{dt})
\]

**Predictor step** — an extrapolation to time \( t + \Delta t \) of values from earlier time steps:

\[
P(x) : \quad x(t + \Delta t) = x(t) + \Delta t \left. \frac{dx}{dt} \right|_t + (\Delta t)^2 \sum_{i=1}^{k-1} \alpha_i f[t + (1 - i) \Delta t]
\]

\[
P\left(\left. \frac{dx}{dt} \right|_t \right) : \quad \left. \frac{dx}{dt} \right|_{t+\Delta t} = \frac{x(t + \Delta t) - x(t)}{\Delta t} + \Delta t \sum_{i=1}^{k-1} \alpha_i f[t + (1 - i) \Delta t]
\]

**Adams-Bashforth method** \( O((\Delta t)^{k+1}) \).
Provides exact results for \( x(t) = t^q \) with \( q \leq k \) if the (rational) coefficients \( \alpha_i \) satisfy the \( k - 1 \) equations:

\[
\sum_{i=1}^{k-1} (1 - i)^q \alpha_i = \frac{1}{(q + 1)(q + 2)}.
\]

\[
\sum_{i=1}^{k-1} (1 - i)^q \alpha'_i = \frac{1}{q + 2}, \quad q = 0, 1, \ldots, k - 2.
\]
2.5 Predictor-corrector methods

Corrector step — uses the predicted values of $x$ and $dx/dt$:

$$C(x) : \quad x(t + \Delta t) = x(t) + \Delta t \left( \frac{dx}{dt} \right)_{t+\Delta t} + (\Delta t)^2 \sum_{i=1}^{k-1} \beta_i f[t + (2 - i) \Delta t]$$

$$C\left( \frac{dx}{dt} \right) : \quad \left. \frac{dx}{dt} \right|_{t+\Delta t} = \frac{x(t + \Delta t) - x(t)}{\Delta t} + \Delta t \sum_{i=1}^{k-1} \beta'_i f[t + (1 - i) \Delta t]$$

Adams-Moulton method

Coefficients obtained from:

$$\sum_{i=1}^{k-1} (2 - i)^q \beta_i = \frac{1}{(q + 1)(q + 2)},$$

$$\sum_{i=1}^{k-1} (1 - i)^q \beta'_i = \frac{1}{q + 2}, \quad q = 0, 1, \ldots, k - 2.$$

The predicted values do not appear explicitly — only their involvement in evaluating $f$.

Implementation ($k = 4$):

$$P(x) : \quad x(t + \Delta t) = x(t) + v(t) \Delta t + [19a(t) - 10a(t - \Delta t) + 3a(t - 2\Delta t)] \frac{(\Delta t)^2}{24}$$

$$P(v) : \quad v(t + \Delta t) = \frac{x(t + \Delta t) - x(t)}{\Delta t} + [27a(t) - 22a(t - \Delta t) + 7a(t - 2\Delta t)] \frac{\Delta t}{24}$$

$$a(t + \Delta t) = f(t + \Delta t, x(t + \Delta t), v(t + \Delta t))/m$$

$$C(x) : \quad x(t + \Delta t) = x(t) + v(t) \Delta t + [3a(t + \Delta t) + 10a(t) - a(t - \Delta t)] \frac{(\Delta t)^2}{24}$$

$$C(v) : \quad v(t + \Delta t) = \frac{x(t + \Delta t) - x(t)}{\Delta t} + [7a(t + \Delta t) + 6a(t) - a(t - \Delta t)] \frac{\Delta t}{24}$$
2. Basics of molecular dynamics

```fortran
real(8) vx(natm), vy(natm), vz(natm) ! v[Angs/psec]
real(8) ax(natm), ay(natm), az(natm) ! a[Angs/psec*psec]
real(8) ax1(natm), ay1(natm), az1(natm), ax2(natm), ay2(natm), az2(natm)
real(8), allocatable :: x0(:,), y0(:,), z0(:,), vx0(:,), vy0(:,), vz0(:,)
real(8), parameter :: pr0 = 19.d0/24.d0, pr1 =-10.d0/24.d0, pr2 = 3.d0/24.d0
real(8), parameter :: pv0 = 27.d0/24.d0, pv1 =-22.d0/24.d0, pv2 = 7.d0/24.d0
real(8), parameter :: cr0 = 3.d0/24.d0, cr1 = 10.d0/24.d0, cr2 =-1.d0/24.d0
real(8), parameter :: cv0 = 7.d0/24.d0, cv1 = 6.d0/24.d0, cv2 =-1.d0/24.d0
allocate(x0(natm), y0(natm), z0(natm), vx0(natm), vy0(natm), vz0(natm))
dt2 = dt * dt

! PREDICTOR

do iatm = 1,natm
  x0(iatm) = x(iatm); vx0(iatm) = vx(iatm)
  y0(iatm) = y(iatm); vy0(iatm) = vy(iatm)
  z0(iatm) = z(iatm); vz0(iatm) = vz(iatm)
  x(iatm) = x0(iatm) + vx0(iatm) * dt & + (pr0*ax(iatm) + pr1*ax1(iatm) + pr2*ax2(iatm)) * dt2
  y(iatm) = y0(iatm) + vy0(iatm) * dt & + (pr0*ay(iatm) + pr1*ay1(iatm) + pr2*ay2(iatm)) * dt2
  z(iatm) = z0(iatm) + vz0(iatm) * dt & + (pr0*az(iatm) + pr1*az1(iatm) + pr2*az2(iatm)) * dt2
  vx(iatm) = (x(iatm) - x0(iatm)) / dt & + (pv0*ax(iatm) + pv1*ax1(iatm) + pv2*ax2(iatm)) * dt
  vy(iatm) = (y(iatm) - y0(iatm)) / dt & + (pv0*ay(iatm) + pv1*ay1(iatm) + pv2*ay2(iatm)) * dt
  vz(iatm) = (z(iatm) - z0(iatm)) / dt & + (pv0*az(iatm) + pv1*az1(iatm) + pv2*az2(iatm)) * dt
  ax2(iatm) = ax1(iatm); ax1(iatm) = ax(iatm)
  ay2(iatm) = ay1(iatm); ay1(iatm) = ay(iatm)
  az2(iatm) = az1(iatm); az1(iatm) = az(iatm)
end do

! CORRECTOR

call Forces(x,y,z,ax,ay,az,natm,Epot) ! f(t+dt)
Ekin = 0.d0

do iatm = 1,natm
  ax(iatm) = ax(iatm) / amass
  ay(iatm) = ay(iatm) / amass
  az(iatm) = az(iatm) / amass
  x(iatm) = x0(iatm) + vx0(iatm) * dt & + (cr0*ax(iatm) + cr1*ax1(iatm) + cr2*ax2(iatm)) * dt2
  y(iatm) = y0(iatm) + vy0(iatm) * dt & + (cr0*ay(iatm) + cr1*ay1(iatm) + cr2*ay2(iatm)) * dt2
  z(iatm) = z0(iatm) + vz0(iatm) * dt & + (cr0*az(iatm) + cr1*az1(iatm) + cr2*az2(iatm)) * dt2
  vx(iatm) = (x(iatm) - x0(iatm)) / dt & + (cv0*ax(iatm) + cv1*ax1(iatm) + cv2*ax2(iatm)) * dt
  vy(iatm) = (y(iatm) - y0(iatm)) / dt & + (cv0*ay(iatm) + cv1*ay1(iatm) + cv2*ay2(iatm)) * dt
end do
```

The behavior of finite systems is very different from that of infinite (massive) systems. Unless we want to simulate clusters of atoms or molecules (having a well-defined number of constituents), the number of particles used to simulate bulk properties of macroscopic systems plays an essential role. Irrespective of how large the simulated system can be considered (still keeping the simulation tractable), the number of particles \( N \) would be negligible as compared to the number of particles contained in a macroscopic system (of the order of \( 10^{21} \div 10^{23} \)).

In macroscopic systems, only a small fraction of the atoms are located in the vicinity of the boundaries (walls of the container in which the system resides). Considering, for example, a typical liquid characterized by \( N = 10^{21} \), the number of atoms in the vicinity of the walls would be of the order \( N^{2/3} = 10^{14} \), that means 1 in \( 10^7 \) is a surface atom. Thus, in massive systems, the fraction of particles near the walls is negligible. The typical number of particles which can be handled in MD simulation nowadays is of the order \( 10^6 \).

For such a system, the fraction of the surface atoms is much more significant and the behavior would be dominated by surface effects.

The most convenient solution to overcome both the problem of the "finite" size of the system and to minimize the surface effects is to use periodic boundary conditions. Using periodic boundary conditions implies that particles are enclosed in a box, which is mentally replicated to infinity by rigid translation in all the three Cartesian directions, completely filling the space. All the “image” particles move solidary with their “original” particle from the simulated box and, in fact, only one of them is effectively simulated. When a particle enters or leaves the simulation region, an image particle leaves or enters this region, such that the number of particles from the simulation region is always conserved. One can easily see that the surface effects are thus virtually eliminated and the position of the box boundaries plays no role.
Calculating the interactions. As a result of using periodic boundary conditions, each particle \( i \) in the simulation box appears to be interacting not only with the other particles in the box, but also with their images. Apparently, the number of interacting pairs increases enormously. Nevertheless, this inconvenience can be overcome if one uses a potential with a finite range, since the interaction of two particles separated by a distance exceeding a cutoff distance \( R_{\text{cut}} \) can be ignored. Supposing that the box size is larger than \( 2R_{\text{cut}} \) along each Cartesian direction, it is obvious that at most one among the pairs formed by a particle \( i \) in the box and all the periodic images of another particle \( j \) will lie within the cut off distance \( R_{\text{cut}} \) and, thus, will interact. This is the essence of the minimum image criterion: among all images of a particle, consider only the closest and neglect the rest.

```fortran
!----------------------------------------------------------------------------
! subroutine Forces(x,y,z,fx, fy, fz, natm, dcell, Epot, virial)
!----------------------------------------------------------------------------
!
! Energies in eV, forces in eV/Angstroem, distances in Angstroem
implicit real(8) (a-h, o-z)
!
parameter (epsilon = 1.0d0, sigma = 1.0d0, sigma2 = sigma * sigma)
parameter (Rcut = 3.0d0, Rcut2 = Rcut * Rcut)
real(8) x(natm), y(natm), z(natm)
real(8) fx(natm), fy(natm), fz(natm)
!
do iatm = 1, natm
    fx(iatm) = 0.0d0; fy(iatm) = 0.0d0; fz(iatm) = 0.0d0
end do
!
Epot = 0.0d0
virial = 0.0d0
!
do iatm = 1, natm-1
    do jatm = iatm+1, natm
        dx = x(iatm) - x(jatm)
dy = y(iatm) - y(jatm)
dz = z(iatm) - z(jatm)
!
        if (abs(dx) > 0.5d0*dcell) dx = dx - sign(dcell,dx)
        if (abs(dy) > 0.5d0*dcell) dy = dy - sign(dcell,dy)
        if (abs(dz) > 0.5d0*dcell) dz = dz - sign(dcell,dz)
!
        r2 = dx*dx + dy*dy + dz*dz
        if (r2 < Rcut2) then
            fr2 = sigma2 / r2
            fr6 = fr2 * fr2 * fr2
            fpr = 48.0d0 * epsilon * fr6 * (fr6 - 0.5d0) / r2  ! f/r
            fxi = fpr * dx
            fyi = fpr * dy
            fzi = fpr * dz
!
            fx(iatm) = fx(iatm) + fxi; fx(jatm) = fx(jatm) - fxi
            fy(iatm) = fy(iatm) + fyi; fy(jatm) = fy(jatm) - fyi
        end if
    end do
end do
```


2.6 Periodic boundary conditions

\[ fz(iatm) = fz(iatm) + fzi; fz(jatm) = fz(jatm) - fzi \]

\[ Epot = Epot + 4.d0 * \epsilon * fr6 * (fr6 - 1.d0) \]

\[ \text{virial} = \text{virial} + \text{fpr} * r2 \]

end if
end do
end do

Epot = Epot / natm
virial = virial / natm
end

Correcting particle coordinates. After each integration step the coordinates of the particles must be examined. If a particle is found to have left the simulation region, its coordinates must be readjusted to bring it back inside, which is equivalent to bring in an image particle through the opposite boundary. Supposing that the simulation region is a rectangular box, this is done by adding to or subtracting from the affected particle coordinate the size \( L \) of the box along the corresponding direction. Thus, for instance, assuming that the \( x \)-coordinates are defined to lie between 0 and \( L \), if \( x < 0 \) (the particle leaves the box in the negative \( O x \) direction), then the coordinate must be corrected by adding the box size \( L \). If \( \dot{x} > L \) (the particle exits in the positive \( O x \) direction), \( L \) must be subtracted from the particle coordinate:

\[
\begin{align*}
\text{if } x < 0 & \text{ then } x \rightarrow x + L \\
\text{if } \dot{x} > L & \text{ then } x \rightarrow x - L.
\end{align*}
\]

A similar analysis must be carried out for each coordinate of the particle.

The readjustment of the coordinates in the context of using periodic boundary conditions has been implemented in the following routine.

!============================================================================
!subroutine BoundCond(x,y,z,natm,dcell)
!----------------------------------------------------------------------------
! Adjusts the atomic coordinates for periodic boundary conditions
! according to the minimum image criterion
!----------------------------------------------------------------------------
implicit real(8) (a-h,o-z)
real(8) x(natm), y(natm), z(natm)
do iatm = 1,natm
if (x(iatm) < 0.d0) then
  x(iatm) = x(iatm) + dcell
else if (x(iatm) > dcell) then
  x(iatm) = x(iatm) - dcell
end if

if (y(iatm) < 0.d0) then
  y(iatm) = y(iatm) + dcell
else if (y(iatm) > dcell) then
  y(iatm) = y(iatm) - dcell
end if
end do
The routine `BoundCond` is normally called from the `Verlet` propagator:

```fortran
!============================================================================
subroutine Verlet(x,y,z,vx,vy,vz,ax,ay,az,natm,dt,dens,dcell,Ekin,Epot,pres)
!----------------------------------------------------------------------------
implicit real(8) (a-h,o-z)
real(8) x(natm), y(natm), z(natm)
real(8) vx(natm), vy(natm), vz(natm)
real(8) ax(natm), ay(natm), az(natm)
dt2 = 0.5d0 * dt
do iatm = 1,natm
  vx(iatm) = vx(iatm) + ax(iatm) * dt2 ! v(t+dt/2)
  vy(iatm) = vy(iatm) + ay(iatm) * dt2
  vz(iatm) = vz(iatm) + az(iatm) * dt2
  x(iatm) = x(iatm) + vx(iatm) * dt ! r(t+dt)
  y(iatm) = y(iatm) + vy(iatm) * dt
  z(iatm) = z(iatm) + vz(iatm) * dt
end do

! call BoundCond(x,y,z,natm,dcell)
! call Forces(x,y,z,ax,ay,az,natm,dcell,Epot,virial) ! f(t+dt)
Ekin = 0.d0
do iatm = 1,natm
  vx(iatm) = vx(iatm) + ax(iatm) * dt2 ! v(t+dt)
  vy(iatm) = vy(iatm) + ay(iatm) * dt2
  vz(iatm) = vz(iatm) + az(iatm) * dt2
  Ekin = Ekin + vx(iatm)**2 + vy(iatm)**2 + vz(iatm)**2
end do
Ekin = 0.5d0 * Ekin / natm

pres = dens * (2.d0*Ekin + virial) / 3.d0
end
```

Bibliografie

