# Contents

# 1 Langevin Equation

The Langevin Equation for the i-th particle is:

$$m_i\ddot{\vec{x}}_i = \vec{F}(\vec{x}_i(t)) - \gamma_i m_i \dot{\vec{x}}_i + \vec{R}_i \tag{1}$$

where $\vec{F}(\vec{x}_i(t)) = \vec{F}_i$ is the interaction force, $\gamma_i$ is the collision frequency and $\vec{R}_i$ represents a random force which is related to the mass $m_i$, the Temperature $T$ and the collision frequency $\gamma_i$ by:

$$\langle \vec{R}_i(t)\vec{R}_j(t')\rangle = 2m_l k_B T \gamma_l \delta_{i,j}\delta(t-t') \tag{2}$$

and

$$\langle \vec{R}_i(t)\rangle = 0 \tag{3}$$

This force can be calculated using a Gaussian distribution

$$\rho(\vec{R}_i) = \frac{1}{\sqrt{4\pi\gamma_i m_i k_B T}} \cdot e^{\frac{-\vec{R}_i^2}{4\gamma_i m_i k_B T}} \tag{4}$$

or using

$$\vec{R}_i(t) = \sqrt{2k_B T \gamma_i m_i}\eta(t) \tag{5}$$

where $\eta(t) = \dot{W}(t)$ is a white-noise ($W(t)$ being a Wiener process). In order to solve the differential equation one needs to integrate the equation of motion. A second-order integrator for Langevin equations was found by Eric Vanden-Eijnden and Giovanni Ciccotti [1] and is a generalizaton of the BBK integrator. The algorithm reduces to the velocity-Verlet algorithm when $\gamma_i = 0$.

# 2 Integrator

The Integrator found is

$$\begin{cases} \vec{v}^{(n+1/2)} = \vec{v}^{(n)} + \frac{1}{2}\Delta t\vec{f}(\vec{x}^{(n)}) - \frac{1}{2}\Delta t\gamma\vec{v}^{(n)} + \frac{1}{2}\sqrt{\Delta t}\sigma\vec{\xi}^{(n)} \\ \qquad\qquad - \frac{1}{8}\Delta t^2\gamma(\vec{f}(\vec{x}^{(n)}) - \gamma\vec{v}^{(n)}) - \frac{1}{4}\Delta t^{3/2}\gamma\sigma\left(\frac{1}{2}\vec{\xi}^{(n)} + \frac{1}{\sqrt{3}}\vec{\eta}^{(n)}\right) \\ \vec{x}^{(n+1)} = \vec{x}^{(n)} + \Delta t\vec{v}^{(n+1/2)} + \Delta t^{3/2}\sigma\frac{1}{\sqrt{12}}\vec{\eta}^{(n)} \\ \vec{v}^{(n+1)} = \vec{v}^{(n+1/2)} + \frac{1}{2}\Delta t\vec{f}(\vec{x}^{(n+1)}) - \frac{1}{2}\Delta t\gamma\vec{v}^{(n+1/2)} + \frac{1}{2}\sqrt{\Delta t}\sigma\vec{\xi}^{(n)} \\ \qquad\qquad - \frac{1}{8}\Delta t^2\gamma(\vec{f}(\vec{x}^{(n+1)}) - \gamma\vec{v}^{(n+1/2)}) - \frac{1}{4}\Delta t^{3/2}\gamma\sigma\left(\frac{1}{2}\vec{\xi}^{(n)} + \frac{1}{\sqrt{3}}\vec{\eta}^{(n)}\right) \end{cases} \tag{6}$$

where $(\vec{\xi}^{(n)}, \vec{\eta}^{(n)})$ are independent Gaussian variables with mean zero and covariance

$$\langle \xi_i^{(n)}\xi_j^{(n)}\rangle = \langle \eta_i^{(n)}\eta_j^{(n)}\rangle = \delta_{i,j} \qquad \langle \xi_i^{(n)}\eta_j^{(n)}\rangle = 0 \tag{7}$$

# 3 Reduced Units

As molecular dynamics actions take place on a small time- and spacescale it's convenient to change the unit system to the system of reduced units (MD units) [3]. Following changes are made

$$\tilde{r} \to r\sigma \tag{8}$$
$$\tilde{E} \to E\epsilon \tag{9}$$
$$\tilde{m} \to m m_a \tag{10}$$

where $\sigma$ and $\epsilon$ are paramter of the Lennard-Jones potential ($\epsilon$ governs the strength of the interaction and $\sigma$ defines a length scale, both just numbers) and $m_a$ is the atomic mass. In order to derive the factor for the conversion factor of time one can use the fact that for example the formula for the kinetic energy shouldn't change under unit system transformations. This leads to

$$\tilde{t} \to t\sqrt{\frac{m_a\sigma^2}{\epsilon}} \tag{11}$$

By setting $k_B = 1$ the MD unit of temperature is now also defined. The following table contains the most used quantities, as well as values for argon:

| Physical quantity | Unit | Value for Ar |
|---|---|---|
| Length | $\sigma$ | $3.4 \cdot 10^{-10}\ m$ |
| Energy | $\epsilon$ | $1.65 \cdot 10^{-25}\ J$ |
| Mass | $m$ | $6.69 \cdot 10^{-26}\ kg$ |
| Time | $\sigma(m/\epsilon)^{1/2}$ | $2.17 \cdot 10^{-12}\ s$ |
| Velocity | $(\epsilon/m)^{1/2}$ | $1.57 \cdot 10^{2}\ m/s$ |
| Force | $\epsilon/\sigma$ | $4.85 \cdot 10^{-12}\ N$ |
| Pressure | $\epsilon/\sigma^3$ | $4.20 \cdot 10^{7}\ Nm^{-2}$ |
| Temperature | $\epsilon/k_B$ | $120\ K$ |

Table 1: Conversation factors for MD units and specific values for argon

For the mass of argon the reduced timesunit corresponds to $2.161 \cdot 10^{-12}\ s$. Thus a simulation timestep of $\Delta t = 0.005$ would correspond to approx. $10^{-14}\ s$. For a liquid density of $0.942\ g/cm^3$ the reduced lenght of a box sized simulation region is $L = 1.218N^{1/3}$.

# 4 Lennard-Jones Potential

The Lennard-Jones Potential approximates the interaction between neutral atoms or molecules. A common expression is

$$U = 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) \tag{12}$$

where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ and $r_{ij} = |\vec{r}_{ij}|$ is the distance between the particle $i$ and $j$, $\sigma$ is the finite distance at which the inter-particle potential is zero and $\epsilon$ is the depth of the potential well.
For simulations it is necessary to cut off the potential at a given radius $r_c$ in order to reduce the simulationtime, and the potential becomes

$$U_{LJ} = \begin{cases} 4\epsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \tag{13}$$

To avoid a discontinuity at the cutoff it is commom to use a truncated and shifted potential. The following potential satifies this condition

$$U = \begin{cases} U_{LJ} - U_{LJ}(r_c) & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \tag{14}$$

The force is

$$-\vec{\nabla}U = \vec{F}_{ij} = \begin{cases} \frac{48\epsilon}{\sigma^2}\left(\left(\frac{\sigma}{r_{ij}}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r_{ij}}\right)^{8}\right)\vec{r}_{ij} & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \tag{15}$$

When reduced units are used, the force reduces to the following form

$$-\vec{\nabla}U = \vec{F}_{ij} = \begin{cases} 48\left(r_{ij}^{-14} - \frac{1}{2}r_{ij}^{-8}\right)\vec{r}_{ij} & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \tag{16}$$

A plot of the Lennard-Jones-Potential can be seen in figure 1.

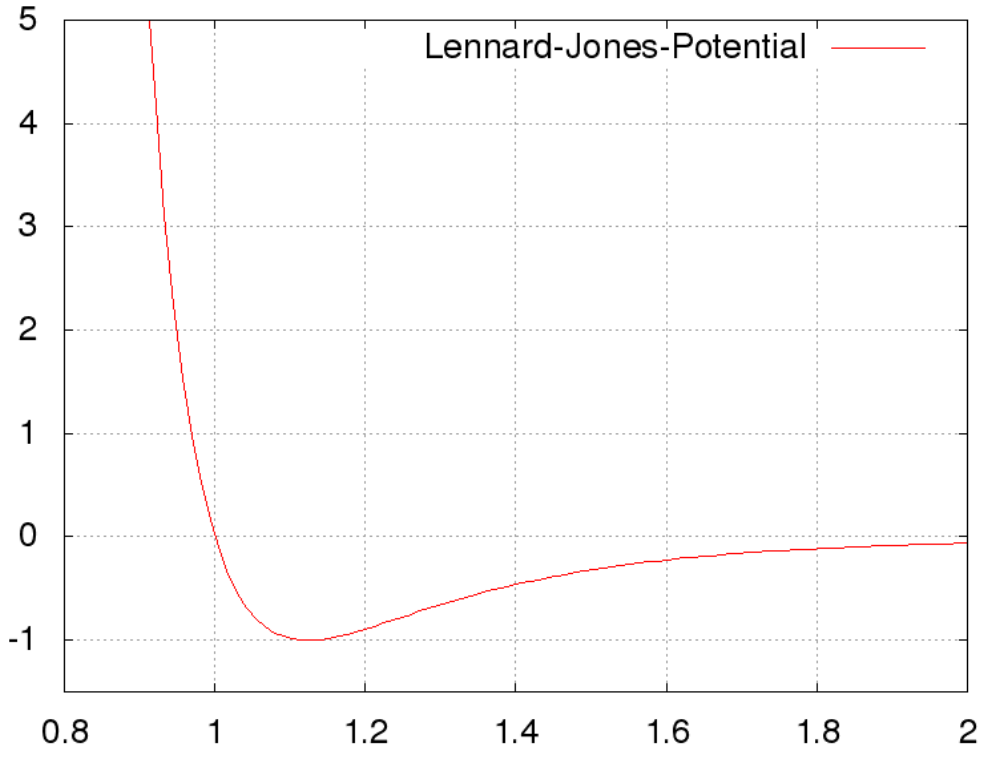


Figure 1: Lennard-Jones-Potential in reduced units of Argon

# 5 Implementation details

## 5.1 General algorithm

In the case where the force depends on all other particles (e.g. Lennard-Jones-Potential) one is forced to split the algorithm in two different parts. One for the calculation of the half-step for the velocity and the new position. And the other part for the calculation of the new velocity. A pseudocode for the implementation could be like Algorithm 1 and a expiration chart of this algorithm may look like Figure 2.

**Data**: Initial positions and velocities of $m$ particles $(x_1^1, x_2^1, ..., x_m^1)$, $(v_1^1, v_2^1, ..., v_m^1)$
**Result**: Final positions $(x_1^n, x_2^n, ..., x_m^n)$ and velocities $(v_1^n, v_2^n, ..., v_m^n)$
initialization;
**for** $k \leftarrow 1$ **to** $n$ **do**
  **for** $i \leftarrow 1$ **to** $m$ **do**
    calculate force for all $m$ particles;
    update positions of all particles;
    calculate velocity half step;
    calculate force for new positions;
    update velocity;
  **end**
**end**

**Algorithm 1:** Main algorithm

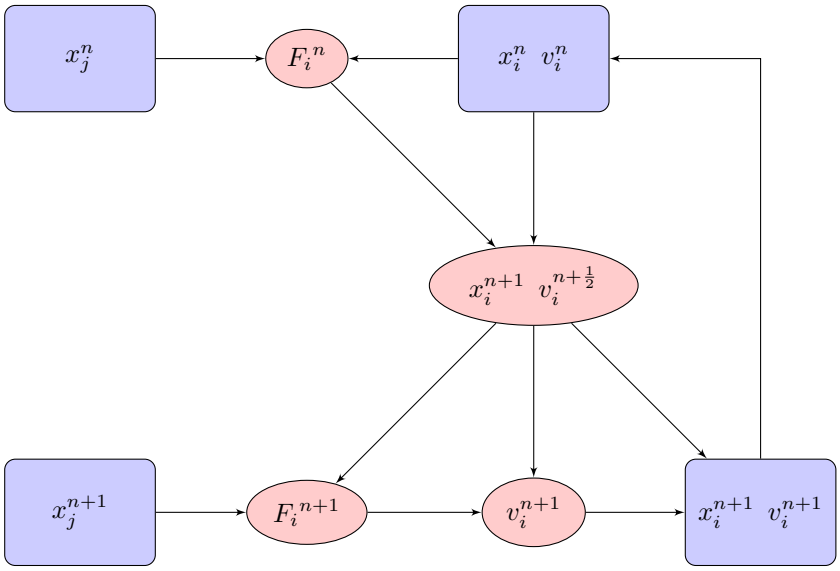


Figure 2: Expiration chart for the implementation. Red: calculation, Blue: storage

## 5.2 Periodic Boundary Conditions

When simulating liquids the problem is the huge amount of particles. To reduce this problem one can introduce periodic boundary conditions, where the simulation takes place in a container of some kind and considering an infinte, space-filling array of idential copies of the simulation region. This means, the particles near the boundaries effect the particles on the other side of the simulation space. This leads to a wraparound effect. If a particle moves outside the region, it has to be set on the other side. The following pseudocode takes care of this behavoir

**Data**: New position $x_i$ of particle, Simulationspace size $L$
**Result**: If particle leaves simulation region, update position based on periodic boundary conditions
initialization;
**if** $x_i > L$ **then**
  $x_i = x_i - L$;
**end**
**if** $x_i < 0$ **then**
  $x_i = x_i + L$;
**end**

**Algorithm 2:** Particle wraparound

## 5.3 Force calculation

For a system of $N$ particles, most of the simulation time is spend in calculating the force between different particles. A way to reduce this calculation time is to cut off after a specific radius $r_c$ and only include particle within this radius. For the Lennard-Jones potential typical values are $r_c = 2.5\sigma - 3.5\sigma$. A way to improve this technique was introduced by Loup Verlet [4], the so-called Neighbor list or Verlet list. The idea is to use a second radius $r_v > r_c$ and calculate the force for all particles within this radius, but only at every n-th step. In order to estimate $r_v$ one can use

$$r_v = r_c + \Delta r \tag{17}$$

with

$$\Delta r \leq n\tilde{v}\Delta t \tag{18}$$

where $\Delta t$ is the time step, $\tilde{v}$ the root-mean-square. See 5.3 for a graphic representation of this method. This means that within the following $n-1$ steps no particles, other the ones in the list, will move into the cut-off radius. A typical number for $n$ is 10 - 20. The computation time is reduced by a factor of 10.

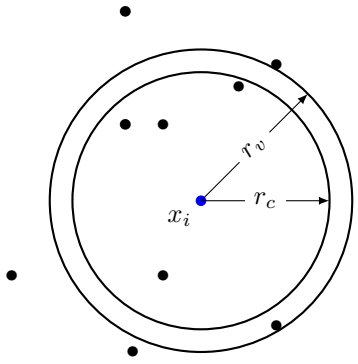


Figure 3: Neighbor list

For periodic boundary conditions the distance between to particles depends on the simulation region, as two particles near the boundaries can effect each other. A pseudocode for the calculation of the force $F_i$ for the i-th particle is listed below (see algorithm 3).

**Data**: Position of all $N$ particles at timestep $m$ $(x_1^m, x_2^m, ..., x_N^m)$, boolean periodic
**Result**: Force $F_i^m$ for i-th particle
initialization;
**if** $m \mod n == 0$ **then**
  **for** $j \leftarrow 1$ **to** $N$ **do**
    calculate distance $d$ from particle $i$ to other particles;
    **if** $d \leq r_v$ **then**
      add particle position to neighbor list;
    **end**
  **end**
**end**
calculate force on i-th particle based on neighbor list;

**Algorithm 3:** Neighbor list with boundary conditions

## 5.4 OpenMP

OpenMP [2] is application programming interface (API) which supports parallel programming in C/C++ and Fortran. The easiest way to use OpenMP is by splitting up indpendent loops in a given amount of threads, so that each thread can calculate parallel. To use OpenMP one needs to import the header omp.h. The following function call specifies the amount of threads which want to be used in the following code segment

```
omp_set_num_threads(NUM_THREADS);
```

It is not guaranteed that the specified amount of threads are created. A parallel region is introduced by

```
#pragma omp parallel for
```

and closed by

```
#pragma omp barrier
```

The last statement ensures that the computation continues only if every thread has finished it's task. It is possible to keep variables private inside of threads. The following expamle splits the loop over $i$ in a specified amount of threads, but keeps $j$ and $k$ private for each thread to prevent unexpected changes of these variables. Another way of keeping variables private is to declare them inside the loops.

```
omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for private(j,k)
  for (i=0; i<amount; i++){
    int l;
    for (j=0; j<amount; j++){
      for (k=0; k<3; k++){
        // Computations
      }
    }
  }
#pragma omp barrier
```

Listing 1: Example: OpenMP

If two or more threads want to change a shared value, it causes problems if this happens at the same time. There are two ways to prevent this. First by using the *reduction* directive. This statement creates a copy of the argument for each thread and uses the unity element for the specified operation to combine them after the parallel region.

```
#pragma omp parallel for reduction(+:variable)
```

This method doesn't work for class members. In this case the *atomic* directive can be used.

```
#pragma omp atomic
```

The statement after this directive is protected and can only be written if no other thread is currently writing. If a compiler doesn't support OpenMP, the directives cause errors on this systems. To avoid this, the following statement is used

```
#ifdef _OPENMP
  #include <omp.h>
#endif
```

Listing 2: To avoid error messages where OpenMp is not installed

# 6 Measurements

The kinetic and potential energies are

$$E_K = \frac{1}{2}\sum_{i=1}^{N} \vec{v}_i^2 \tag{19}$$

$$E_U = 4 \sum_{i \leq i < j \leq N} (r_{ij}^{-12} - r_{ij}^{-6}) \tag{20}$$

The temperature of the system is

$$T = \frac{1}{3N}\sum_{i=1}^{N} \vec{v}_i^2 \tag{21}$$

The pressure is defined in terms of the virial expression

$$PV = NT + \frac{1}{3}\Big\langle \sum_{i=1}^{N} \vec{x}_i \cdot \vec{F}_i \Big\rangle \tag{22}$$

For the Lennard-Jones-Potential this becomes

$$PV = \frac{1}{3}\Big\langle \sum_{i}^{N} \vec{v}_i^2 + 48 \sum_{i<j} (r_{ij}^{-12} - r_{ij}^{-6}) \Big\rangle \tag{23}$$

# 7 Results for the Lennard-Jones-Fluid

If the friction constant $\gamma$ is set to zero, the integrator reduces to the velocity verlet integrator and therefore energy should be conserved. To qualify this, the shift with respect to the initial value is calculated. This is done in the following way

$$\Delta E = \frac{E(t) - E(0)}{E(t)} \tag{24}$$

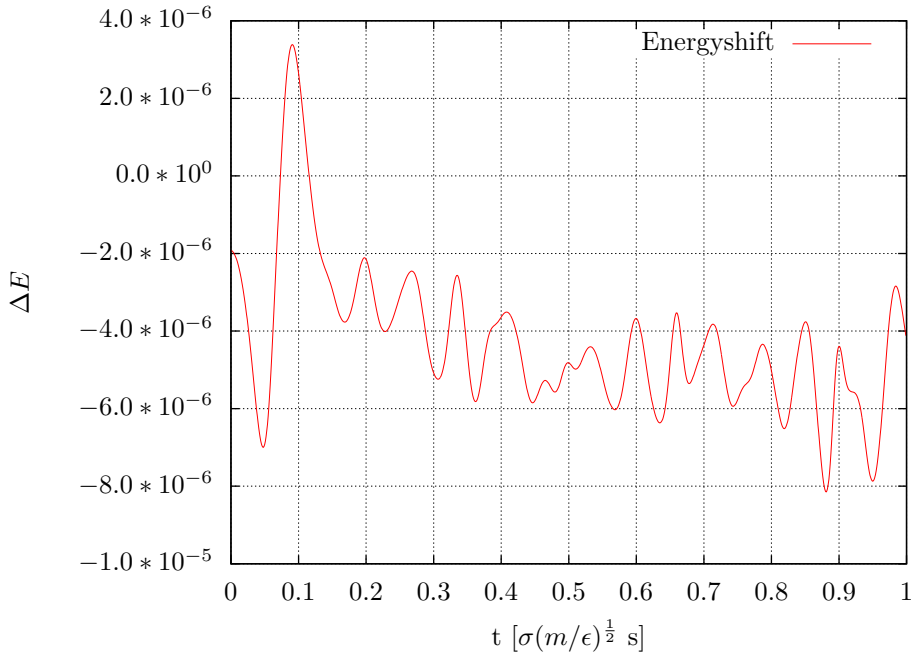


Figure 4: Energyshift ($\gamma = 0$, $N = 512$, $\rho = 0.6$, $\Delta t = 0.001$, $T = 1$)

The Langevin-Integrator works as a thermostat, i.e. the system is coupled to a heat bath. Like the kinetic energy and the pressure, the temperature fluctuates. The difference between a system with and without a heat bath can be seen in figure

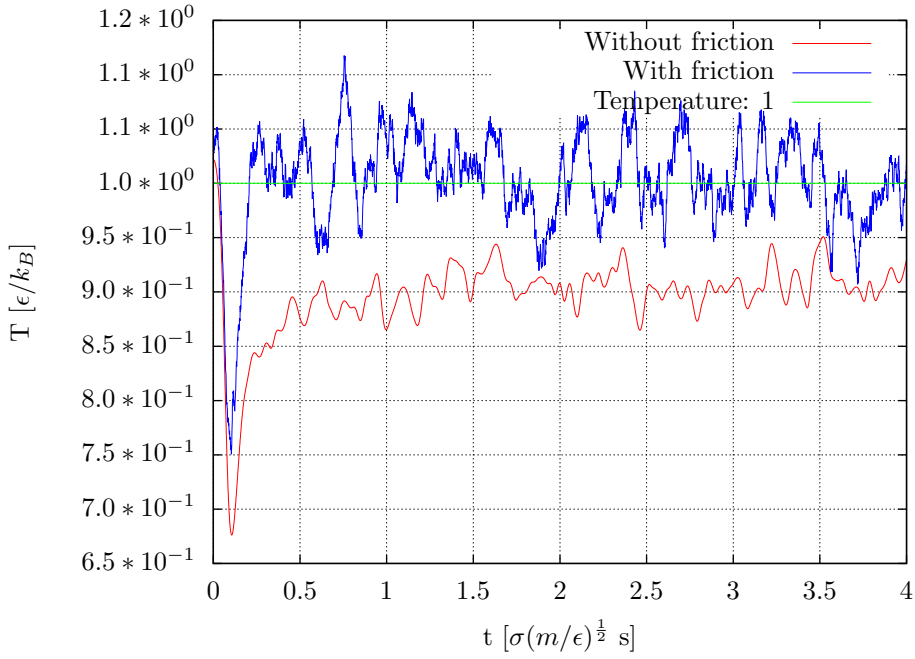


Figure 5: Temperature ($N = 512$, $\rho = 0.6$, $T = 1$) Red: $\gamma = 0$, Blue: $\gamma = 5$, Green: Initial temperature 1

# References


[1] Giovanni Ciccotti Eric Vanden-Eijnden. Second-order integrators for langevin equations with holonomic constraints. *Chemical Physics Letters*, 429, 2006.

[2] OpenMP API for parallel programming, version 3.1.

[3] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge, 2004.

[4] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, 159, 1967.