

Molecular Dynamics Simulation of Argon

Molecular Dynamics (MD) is widely used to simulate many-particle systems ranging from solids, liquids, gases, and biomolecules on Earth, to the motion of stars and galaxies in the Universe.

Newton's equations of motion for the system are integrated numerically. If the system is in equilibrium, static properties such as temperature and pressure are measured as averages over time. Dynamical properties such as heat transport, or relaxation of systems far from equilibrium, can also be studied.

The fundamental work on this problem was done by A. Rahman, Phys. Rev. **136**, A405 (1964). It was extended in many important ways by L. Verlet, Phys. Rev. **159**, 98 (1967), who introduced the Verlet algorithm and the use of a neighbor list to speed up the calculation.

Simple model of interacting Argon atoms

Consider N atoms of argon each with mass $m = 6.69 \times 10^{-26}$ kg. Argon is an *inert gas*: argon's atoms behave approximately like hard spheres which attract one another with weak van der Waals forces. The forces between two argon atoms can be approximated quite well by a Lennard-Jones potential energy function:

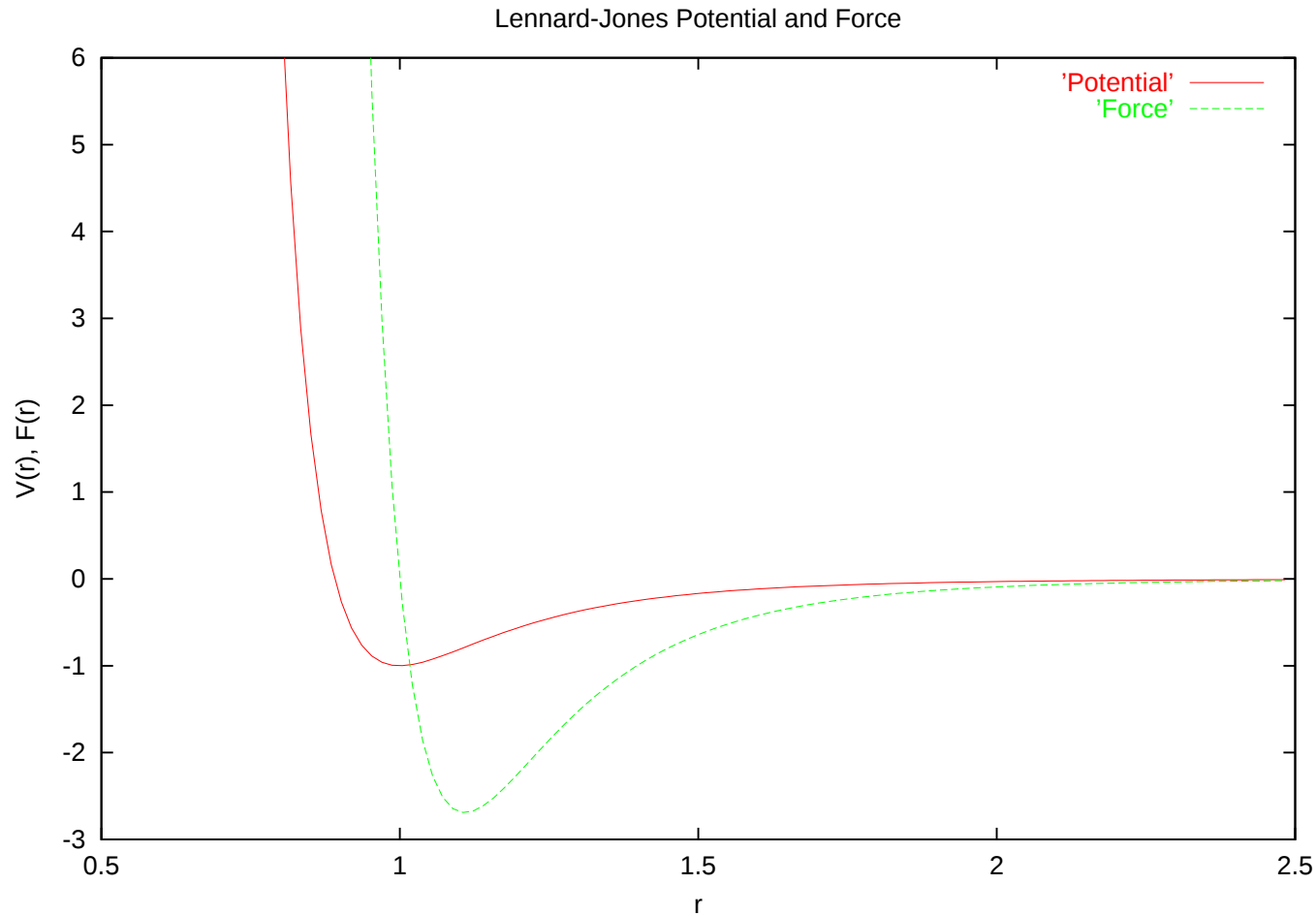
$$U(r) = \varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^6 \right],$$

where r is the distance between the centers of the two atoms, $\varepsilon/k_B = 119.8$ K is the strength of the potential energy, and $\sigma = 3.822 \times 10^{-10}$ m is the value of r at which the energy is zero. The potential has its minimum $U(\sigma) = -\varepsilon$ at $r = \sigma$.

The shape of the potential and the strength of the Lennard-Jones force

$$F(r) = -\frac{dU(r)}{dr} = \frac{12\varepsilon}{\sigma} \left[\left(\frac{\sigma}{r} \right)^{13} - \left(\frac{\sigma}{r} \right)^7 \right],$$

are shown in the following figure:



We will choose units of mass, length and energy so that $m = 1$, $\sigma = 1$, and $\varepsilon = 1$. The unit of time in this system is given by

$$\tau = \sqrt{\frac{m\sigma^2}{\varepsilon}} = 2.43 \times 10^{-12} \text{ s} ,$$

which shows that the natural time scale for the dynamics of this system is a few picoseconds!

A simple MD program

First include some standard headers

```
#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
```

Define data structures to describe the kinematics of the system

```
const int N = 64;           // number of particles
double r[N][3];             // positions
double v[N][3];             // velocities
double a[N][3];             // accelerations
```

We next need to set the initial positions and velocities of the particles. This is actually a complicated problem! Because the system can be simulated only for a few nanoseconds, the starting configuration must be very close to equilibrium to get good results. For a dense system, the atoms are usually placed at the vertices of a face-centered cubic lattice, which tends to minimize the potential energy. The atoms are also given random velocities to approximate the desired temperature.

In this preliminary program we will put the system in a cubical volume of side L and place the particles at the vertices of a simple cubic lattice:

```
double L = 10;              // linear size of cubical volume
double vMax = 0.1;          // maximum initial velocity component
```

```

void initialize() {

    // initialize positions
    int n = int(ceil(pow(N, 1.0/3))); // number of atoms in each direction
    double a = L / n;                // lattice spacing
    int p = 0;                        // particles placed so far
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            for (int z = 0; z < n; z++) {
                if (p < N) {
                    r[p][0] = (x + 0.5) * a;
                    r[p][1] = (y + 0.5) * a;
                    r[p][2] = (z + 0.5) * a;
                }
                ++p;
            }

    // initialize velocities
    for (int p = 0; p < N; p++)
        for (int i = 0; i < 3; i++)
            v[p][i] = vMax * (2 * rand() / double(RAND_MAX) - 1);
}

```

Newton's Equations of Motion

The vector forces between atoms with positions $\mathbf{r}_i$ and $\mathbf{r}_j$ are given by

$$\mathbf{F}_{\text{on } i \text{ by } j} = -\mathbf{F}_{\text{on } j \text{ by } i} = 12(\mathbf{r}_i - \mathbf{r}_j) \left[\left(\frac{1}{r} \right)^{-14} - \left(\frac{1}{r} \right)^{-8} \right],$$

where $r = |\mathbf{r}_i - \mathbf{r}_j|$.

The net force on atom i due to all of the other $N - 1$ atoms is given by

$$\mathbf{F}_i = \sum_{\substack{j=1 \\ j \neq i}}^N \mathbf{F}_{\text{on } i \text{ by } j} .$$

The equation of motion for atom i is

$$\mathbf{a}_i(t) \equiv \frac{d\mathbf{v}_i(t)}{dt} = \frac{d^2\mathbf{r}_i(t)}{dt^2} = \frac{\mathbf{F}_i}{m} ,$$

where $\mathbf{v}_i$ and $\mathbf{a}_i$ are the velocity and acceleration of atom i . These $3N$ second order ordinary differential equations (ODE's) have a unique solution as function of time t if *initial conditions*, that is, the values of positions $\mathbf{r}_i(t_0)$ and velocities $\mathbf{v}_i(t_0)$ of the particles are specified at some initial time t_0 . The equations can be integrated numerically by choosing a small time step h and a discrete approximation to the equations to advance the solution by one step at a time.

The following function computes the accelerations of the particles from their current positions:

```
void computeAccelerations() {

    for (int i = 0; i < N; i++)                // set all accelerations to zero
        for (int k = 0; k < 3; k++)
            a[i][k] = 0;

    for (int i = 0; i < N-1; i++)                // loop over all distinct pairs i,j
        for (int j = i+1; j < N; j++) {
            double rij[3];                      // position of i relative to j
            double rSqd = 0;
            for (int k = 0; k < 3; k++) {
```

```

        rij[k] = r[i][k] - r[j][k];
        rSq = rij[k] * rij[k];
    }
    double f = 12 * (pow(rSq, -7) - pow(rSq, -4));
    for (int k = 0; k < 3; k++) {
        a[i][k] += rij[k] * f;
        a[j][k] -= rij[k] * f;
    }
}
}

```

Velocity Verlet Integration Algorithm

There are many algorithms which can be used to solve ODE's. Verlet has developed several algorithms which are very widely used in MD simulations. One of them is the *velocity Verlet algorithm*

$$\mathbf{r}_i(t + dt) = \mathbf{r}_i(t) + \mathbf{v}_i(t)dt + \frac{1}{2}\mathbf{a}_i(t)dt^2$$

$$\mathbf{v}_i(t + dt) = \mathbf{v}_i(t) + \frac{1}{2}[\mathbf{a}_i(t + dt) + \mathbf{a}_i(t)]dt$$

It can be shown that the errors in this algorithm are of $\mathcal{O}(dt^4)$, and that it is very stable in MD applications and in particular conserves energy very well.

The following function advances the positions and velocities of the particles by one time step:

```

void velocityVerlet(double dt) {
    computeAccelerations();
    for (int i = 0; i < N; i++)

```

```
    for (int k = 0; k < 3; k++) {  
        r[i][k] += v[i][k] * dt + 0.5 * a[i][k] * dt * dt;  
        v[i][k] += 0.5 * a[i][k] * dt;  
    }  
    computeAccelerations();  
    for (int i = 0; i < N; i++)  
        for (int k = 0; k < 3; k++)  
            v[i][k] += 0.5 * a[i][k] * dt;  
}
```

The instantaneous temperature

This is a simulation in which the number of particles N and the volume L^3 of the system are fixed. Because the Lennard-Jones force is *conservative*, the total energy of the system is also constant.

If the system is in thermal equilibrium, then Boltzmann's *Equipartition Theorem* relates the absolute temperature T to the kinetic energy:

$$3(N - 1) \times \frac{1}{2} k_B T = \left\langle \frac{m}{2} \sum_{i=1}^N v_i^2 \right\rangle.$$

Here the angle brackets $\langle \dots \rangle$ represent a thermal ensemble average. The factor $3(N - 1)$ is the number of *internal* translational degrees of freedom which contribute to thermal motion: the motion of the center of mass of the system does not represent thermal energy!

```
double instantaneousTemperature() {  
    double sum = 0;  
    for (int i = 0; i < N; i++)  
        for (int k = 0; k < 3; k++)  
            sum += v[i][k] * v[i][k];  
}
```

```
    return sum / (3 * (N - 1));  
}
```

Finally, here is the main function which steers the simulation:

```
int main() {  
    initialize();  
    double dt = 0.01;  
    ofstream file("T.data");  
    for (int i = 0; i < 1000; i++) {  
        velocityVerlet(dt);  
        file << instantaneousTemperature() << '\n';  
    }  
    file.close();  
}
```

There are several ways in which this simple program needs to be improved:

- The volume is not really constant because the particles can move out of it! We need to impose suitable boundary conditions, for example periodic boundary conditions.
- The initial positions and velocities need to be chosen more carefully. We will place the particles on a face-centered cubic lattice, and use a Maxwell-Boltzmann distribution for the velocities.
- The system needs to be allowed to come to thermal equilibrium at the desired temperature.
- Thermal averages of various quantities need to be measured.

The next version of the program will include these improvements.