A Simple Molecular Dynamics Simulation

Michael Hanrath, Institute for Theoretical Chemistry, University of Cologne
Michael.Hanrath@uni-koeln.de

I. NEWTON'S EQUATIONS: SINGLE PARTICLE

- fundamental equation of dynamics

$$\vec{F} = m\vec{a} = m\ddot{\vec{x}} = m\frac{\partial^2}{\partial t^2}\vec{x} \tag{1}$$

- conservative force field: force is governed by potential V

$$\vec{F} = -\operatorname{grad} V = -\vec{\nabla}V = -\begin{pmatrix}\frac{\partial}{\partial x}\\ \frac{\partial}{\partial y}\\ \frac{\partial}{\partial z}\end{pmatrix} V \tag{2}$$

II. DISCRETIZATION OF EQUATIONS OF MOTION

A. Straightforward Approach

- it is

$$\vec{F}(t) = m\vec{a}(t) \tag{3}$$

$$= m\frac{\partial^2}{\partial t^2}\vec{x}(t) = m\frac{\partial}{\partial t}\overbrace{\frac{\partial}{\partial t}\vec{x}(t)}^{\vec{v}} \tag{4}$$

$$= m\dot{\vec{v}}(t) = m\lim_{\delta\to 0}\frac{\vec{v}(t+\delta)-\vec{v}(t)}{\delta}$$

- introducing a discrete time step d and approximating the differential quotient (the smaller d the more accurate the result):

$$\vec{F}(t)d = m(\vec{v}(t+d) - \vec{v}(t)) \tag{5}$$

- expressing the time t in terms of d: $t \to \nu d$, $\nu \in \mathbb{N}_0$:

$$\vec{F}(\nu d)d = m(\vec{v}(\nu d + d) - \vec{v}(\nu d)) \tag{6}$$

$$\vec{v}(\nu d + d) = \vec{v}(\nu d) + \frac{d}{m}\vec{F}(\nu d) \tag{7}$$

and discritized in step variable i:

$$\vec{v}_{i+1} = \vec{v}_i + \frac{d}{m}\vec{F} \tag{8}$$

$$\vec{x}_{i+1} = \vec{x}_i + d\vec{v}_i \tag{9}$$

- for the iteration to start we require two initial conditions: $\vec{x}(t=0)$ and $\vec{v}(t=0)$

B. Velocity Verlet Algorithm

- previous straightforward algorithm lacks energy conservation
- starting from two Taylor expansions around $x(t+\delta t)$ and $x(t-\delta) \Rightarrow$ position Verlet algorithm
- further manipulations $\Rightarrow$ velocity Verlet algorithm:

$$\vec{v}(t+\delta t) = \vec{v}(t) + \frac{1}{2}\delta t[\vec{a}(t) + \vec{a}(t+\delta t)] \tag{10}$$

$$\vec{x}(t+\delta t) = \vec{x}(t) + \delta t\vec{v}(t) + \frac{1}{2}(\delta t)^2\vec{a}(t) \tag{11}$$

III. NEWTON'S EQUATIONS: MANY PARTICLES

- fundamental equation of dynamics, N particles

$$\sum_{i\neq j}^{N}\vec{F}_{ij} = m_i\vec{a}_i = m_i\ddot{\vec{x}}_i = m_i\frac{\partial^2}{\partial t^2}\vec{x}_i, \quad \forall_{i\in\{1\ldots N\}} \tag{12}$$

$\vec{F}_{ij}$: force between particles i and j, i: considered particle, j: "other" particle

- we intend to consider distance dependent force fields only (having no angular dependence) with

$$V_{ij} = V(r_{ij}), \quad r_{ij} = \|\vec{r}_i - \vec{r}_j\| \tag{13}$$

IV. ENERGY CONSERVATION

- the total energy of the system must be conserved $\rightarrow$ provides a check for our simulation
- forms of energy in our system
  - kinetic energy

$$E_{\text{kin}} = \frac{1}{2}\sum_{i=1}^{N} m_i\vec{v}_i^2 \tag{14}$$

  - potential energy

$$E_{\text{pot}} = \sum_{i<j}^{N} V_{ij} \tag{15}$$

- so

$$E_{\text{kin}} + E_{\text{pot}} \stackrel{!}{=} \text{const} \tag{16}$$

V. POTENTIALS

- three distance dependent force fields are considered
  - Harmonic

$$V_{\mathrm{H}}(r) = \frac{1}{2}c(r - r_e)^2 \tag{17}$$

  - Lennard-Jones

$$V_{\mathrm{LJ}}(r) = \frac{A}{r^{12}} - \frac{B}{r^6} \tag{18}$$

  - Morse

$$V_{\mathrm{M}}(r) = D_e(1 - e^{-a(r-r_e)})^2 \tag{19}$$

- the gradient w.r.t. x, y, z can be calculated by setting

$$r = \sqrt{x^2 + y^2 + z^2} \tag{20}$$

- since numerical differentiation is subject to error amplification analytical gradients of the potentials should be used in the implementation

VI. IMPLEMENTATION

Write a program implementing the equation of motions for the proposed potentials. Start from random configurations with a reasonable number of particles (e.g. 20). To allow for converging geometric structures also implement a simple form of damping. Try various time steps d. Check energy conservation for the straightforward and velocity Verlet algorithm. Analyze for floating point efficiency of your code with respect to the peak floating point performance of your system. What is the formal scaling of the MD in terms of number of particles N? Can one improve on this for large and potentially partially sparse systems?

Implementation is prefered in C++, Python, or Fortran.