Molecular Dynamics Onboarding

These are notes on learning molecular dynamics (MD) for the first time, for people with at least some experience coding and a good sense of vector calculus. I'm writing for the first time in 2024, any comments are welcome.

1 Energy Conservation with Velocity Verlet

In most of the simulations we work with things have harmonic potentials, mainly this is to make our lives simpler, we treat pretty much everything as a linear force law approximation so we end up with dealing with a lot of springs and 'spring like' things[1].

First set of tasks: For each system described below, (1) derive the forces on the particles by taking the gradient of the potential. (2) Then implement an energy conserving simulation using the Verlet integration (I like the version described right beneath "Eliminating the half-step velocity" on the wikipedia page). (3) check that the energy of the system is conserved and that over the course of a simulation where energy is transferred between kinetic and potential frequently:

$$\text{std}(E) \propto \delta t^2 \tag{1}$$

. Simulations will consist of one or more particles moving around and possibly bumping into the walls or one another.

1.1 Verlet Integration

A Verlet step updates forces, F, velocities, v, and positions, x, with $\text{O}(\Delta t^2)$ error in the conserved energy[2]. A step from t to $t + \Delta t$ can be written:

$$x(t + \Delta t) = x(t) + v(t)\Delta t + \frac{1}{2m}F(x(t))\Delta t^2 \tag{2}$$

$$v(t + \Delta t) = v(t) + \frac{\Delta t}{2m}(F(x(t)) + F(x(t + \Delta t))) \tag{3}$$

Or in code:

[1] 'Spring like' can be usefully because they are simple, but also because as you approach a static system we can approximate all potentials to second order with 'spring like' harmonic potentials using taylor expansions.

[2] When $\Delta t \ll 1$ we can get far more accurate simulations from an integration scheme with errors of order Δt^2 than something like Euler integration which would be something like $x(t + \Delta t) = x(t) + \Delta t v(t)$ and $v(t + \Delta t) = v(t) + \Delta t F(t)/M$.

Initialize x, v, F;
while *As Long as you want to run it* **do**
  $x(t+\Delta t) = x(t) + \Delta t * v(t) + \frac{\Delta t^2}{2*m} * F(t)$;
  Compute $F(t+\Delta t)$; `/* Usually` $\nabla U(x(t+\Delta t))$ `*/`
  $v(t+\Delta t) = v(t) + \frac{\Delta t}{2*m} * (F(t+\Delta t) + F(t))$
  Compute Energy $E(t+\Delta t)$; `/* Or any other state variables. NB order matters in these integration schemes. For example if you calculate the kinetic energy before updating the velocities you will end up with error in energy that goes as` Δt^1 `*/`
end

Algorithm 1: Velocity Verlet Psuedocode

1.2 Systems:

1. 1D Spring. A particle with mass m is connected to a 1D spring at point x_0. The potential energy is:

$$U = \frac{k}{2}(x - x_0)^2 \tag{4}$$

2. Attracted particles. N 'soft disks' in 2D (soft disks just means disks that repel with a spring like interaction) are in a central attraction. They have diameter σ and interact like purely repulsive springs. They also interact with a potential energy that pulls them towards the center the system with a spring-like force. The total potential energy is:

$$U = U_{\text{int}} + U_{\text{center}} \tag{5}$$

The interaction potential is:

$$U_{\text{int}} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{k_{\text{int}}}{2} \left(\sigma - r_{ij}\right)^2 \Theta(\sigma - r_{ij}) \tag{6}$$

where $r_{ij} = |\vec{r}_j - \vec{r}_i|$, k_{int} is the spring constant for these interactions, $\vec{r}_i$ is the center of the ith disk, $\Theta(\cdot)$ is the Heaviside step function, it is 1 when its argument is greater than zero and zero otherwise. The central attraction potential is:

$$U_{\text{center}} = \sum_{i=1}^{N} \frac{k_{\text{center}}}{2} \left(|\vec{r}_i - \vec{r}_0|\right)^2 \tag{7}$$

where k_{center} is the strength of the spring pulling everything towards the center and $\vec{r}_0$ is the center of the system.

3. Particles in a box. In the above example replace the U_{center} term with U_{walls} The wall potentials is:

$$\begin{aligned} U_{\text{walls}} = \sum_{i=1}^{N} \frac{k_{\text{walls}}}{2} \Big[& (\sigma - x_i)^2 \, \Theta(\sigma - x_i) + \\ & (x_i - (L_x - \sigma))^2 \, \Theta(x_i - (L_x - \sigma)) \\ & (\sigma - y_i)^2 \, \Theta(\sigma - x_i) + \\ & (y_i - (L_y - \sigma))^2 \, \Theta(y_i - (L_y - \sigma)) \Big] \end{aligned} \tag{8}$$

where L_x and L_y are the x and y length of the box.

4. Particles in a periodic box. A commonly used tool in MD is putting your system in a periodic box to isolate the system from boundary effects (really you're just introducing a set of different boundary effects, but it is still useful). In a periodic box disks will only interact with one another, so the only potential energy is U_{int} from above. When we say we are using a periodic box, we take that to mean that the distance between two particles is the shortest distance between particle i and any of the copies of particle j on the square periodic grid, put into math what this means is:

$$r_{ij}^{\text{Periodic}} = (\vec{r}_j - \vec{r}_i) - \begin{bmatrix} L_x \text{round}\left(\frac{\vec{r}_{j,x} - \vec{r}_{i,x}}{L_x}\right) \\ \\ L_y \text{round}\left(\frac{\vec{r}_{j,y} - \vec{r}_{i,y}}{L_y}\right) \end{bmatrix} \tag{9}$$

we sometimes call r_{ij}^{periodic}, r'_{ij}. So here the potential energy is

$$U = U_{\text{int}} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{k_{\text{int}}}{2} \left(\sigma - r'_{ij}\right)^2 \Theta(\sigma - r'_{ij}) \tag{10}$$

See Fig. 1 shows some example plots that you could try to recreate for each system. The figure shown is for particles in a non-periodic box.

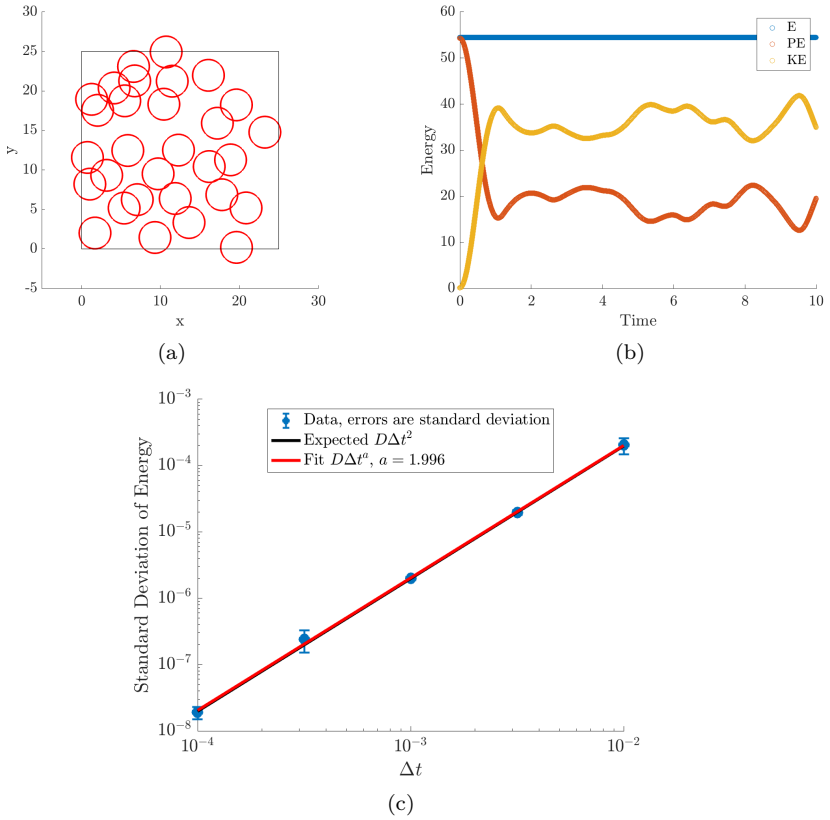

Figure 1: (a) Visualization of the system. (b) Potential and kinetic energy overtime, note there is a 'burn in' period at the start because I start the system with zero velocity and non-zero overlap, then some of the potential energy gets turned into kinetic energy. (c) Power law for $\text{std}(E) \propto \Delta t^2$. I calculate $\text{std}(E)$ using only time after the 'burn in' period. I vary Δt over several decades, but make sure the total amount of simulation time is the same, i.e. if you divide Δt by 10 you need to run for ten times as long. I also average over several different initial configurations for each data point.

1.3 Tips debugging and otherwise

1. Initializing. For a simulation to be useful it must be dynamic, to get dynamics, you need to either start a system with initial velocities, or have non-zero potential energies (i.e. overlaps) when you start. For the mass on the spring you can initialize it with $x(t = 0) \neq x_0$, for the disks in the central attractor, as long as they are not at $\vec{r}_0$ they will move. For the packings of particles in boxes you can either start the particles with some initial velocity (in different directions), or with some overlaps. If

you are initializing the positions randomly within the box you can use the following as a rule of thumb: the packing fraction of the system is the amount of space particles take up over the the total space, here,

$$\phi = \frac{N\pi(\sigma/2)^2}{L_x L_y}, \tag{11}$$

when $\phi < 0.5$ it is possible to have no overlaps at initialization, when $\phi > 0.8$ you are guaranteed to have some overlaps for disks with all the same diameter.

2. Picking parameters. To make life simple you can pick most things to be one. I suggest using $\sigma = 1$, all $k = 1$, for picking the time step it is best to have it be much smaller than the fastest harmonic. To find the periodic of the fastest harmonic you take the largest spring constant, $k_{\max}$ and the smallest mass, $m_{\min}$

$$P = 2\pi\sqrt{\frac{m_{\min}}{k_{\max}}} \tag{12}$$

and we set:

$$\Delta t = \frac{P}{100} \tag{13}$$

. This should be the largest time scale you at which you should expect stable results.

3. If you have bugs simplify the system as much as you can and try to find the line of code where the bug first appears. Start in 1D, then go to 2D, and possibly 3D.

4. You can calculate forces by hand based on the positions and compare those to your calculated forces

5. Look at what you're doing, plot the system over time and over the few time steps where an unexpected behavior begins, try plotting forces and velocities too, are they what you expect?

6. Also try plotting the energy kinetic, potential, versus time or even break it up into smaller terms to find where things are going wrong.

7. Reach out! Let me know how you're doing – I'm happy to answer questions or clarify anything!

8. A full simulation code might look something like this:

```
x = InitPositions(N) ; /* Initialize x, v, F for N particles */
v = 0 * x;
F = 0 * x ;
E(1:T) = 0;
K(1:T) = 0;
U(1:T) = 0 ;                                  /* Main Simulation loop */
for ( t = 1, t < N_steps, t = t + 1 ) {
    F_previous = F;
    x = x + Δt * v + (Δt^2)/(2*m) * F;
    [F, U(t)] =gradU(x);
    v = v + (Δt)/(2*m) * (F + F_previous);
    K(t) = (1/2) Σ m v_i^2;
    E(t) = U(t) + K(t) ; /* Save anything you want to plot here
     */
}
; /* Plotting code                                                */
plot(1:T,U(1:T);
plot(1:T,K(1:T);
plot(1:T,E(1:T);
etc;
;
; /* Function Definitions                                         */
Function gradU(x):
    ...
    return F, U;
Function InitPositions(N):
    ...
    return x;
etc;
```

Algorithm 2: Simulation Psuedocode

2 Integration Schemes

2.1 Conserving Schemes

1. NVE (done above)
2. NVT
   (a) Nose-Hoover
   (b) Langevin
3. NPH
4. NPT

2.2 Minimization Schemes

1. Damped MD (see Langevin)
2. Conjugate Gradient
3. F.I.R.E.