Supplementary Note 9.1

Leapfrog and Velocity Verlet

Derivation from Taylor Expansions

Chapter 9 of the textbook

---

1 Introduction

In Section 9.14 of the textbook, we introduced Verlet methods as useful tools for following the motion of particles. But where do their formulas come from? Why does one version use positions at two successive times, while another updates positions and velocities together? And why does leapfrog put its velocities halfway between the position times?

We can answer these questions with Taylor expansions. The mathematics is familiar, but the way we combine the expansions makes all the difference. Along the way, we will see how to organize the calculations so that we do not waste time evaluating forces—an important consideration in molecular dynamics.

To keep the notation simple, let us follow just one coordinate of a particle. Newton's equation is

$$m\ddot{x} = F(x), \qquad v = \dot{x}, \qquad a(x) = \frac{F(x)}{m}. \tag{1}$$

We will assume that the force depends on position, is sufficiently smooth, and that the time step h is constant. As usual, $t_n = t_0 + nh$; we write x_n, v_n, and a_n for the position, velocity, and acceleration at t_n. Nothing essential changes in three dimensions. For many particles, we apply the same updates to each coordinate, remembering that the acceleration of one particle generally depends on the positions of the others.

2 Basic Verlet from two Taylor expansions

How can we obtain a formula for the next position without explicitly updating the velocity? Here is a simple idea: expand the position both forward and backward about the same time t_n. Then add the results. Watch what happens to the velocity terms:

$$x(t_n + h) = x(t_n) + hv(t_n) + \frac{h^2}{2}a(t_n) + \frac{h^3}{6}x^{(3)}(t_n) + \frac{h^4}{24}x^{(4)}(t_n) + O(h^5),$$
$$x(t_n - h) = x(t_n) - hv(t_n) + \frac{h^2}{2}a(t_n) - \frac{h^3}{6}x^{(3)}(t_n) + \frac{h^4}{24}x^{(4)}(t_n) + O(h^5).$$

When we add these expressions, the terms containing v_n cancel. So do the other odd powers of h:

$$x(t_n + h) + x(t_n - h) = 2x(t_n) + h^2 a(t_n) + \frac{h^4}{12} x^{(4)}(t_n) + O(h^6).$$

We now have an update formula involving positions and acceleration only. If we drop the higher-order terms, we obtain the *basic (position) Verlet method*,

$$\boxed{x_{n+1} = 2x_n - x_{n-1} + h^2 a_n.} \tag{2}$$

Notice that the canceled terms make the local defect in this position recurrence $O(h^4)$. This does not make Verlet a fourth-order method: its global accuracy is second order. If we want a velocity, we can estimate it afterward with another centered difference,

$$v_n \approx \frac{x_{n+1} - x_{n-1}}{2h}, \tag{3}$$

which has error $O(h^2)$.

There is a small practical problem. We usually know the initial position x_0 and velocity v_0, whereas the recurrence asks for a second position. We can obtain one from a Taylor expansion, for example

$$x_{-1} = x_0 - hv_0 + \frac{h^2}{2} a_0 + O(h^3).$$

We can then start the recurrence. Still, having to keep a previous position and reconstruct velocities is sometimes inconvenient. Could we retain the useful features of Verlet but update the velocity directly?

3 Velocity Verlet from Taylor expansions

We begin with the forward Taylor expansion, this time keeping the velocity explicitly. To second order, it gives the position update

$$\boxed{x_{n+1} = x_n + hv_n + \frac{h^2}{2} a_n.} \tag{4}$$

We now have the new position, but we still need the new velocity. Using an ordinary Euler step for it would spoil the accuracy we are trying to achieve. Instead, write a second Taylor expansion, working backward from the *new* time t_{n+1}:

$$x_n = x_{n+1} - hv_{n+1} + \frac{h^2}{2} a_{n+1} + O(h^3).$$

Substitute our forward expression for x_{n+1}. The positions cancel, and we are left with

$$0 = h(v_n - v_{n+1}) + \frac{h^2}{2}(a_n + a_{n+1}) + O(h^3).$$

Solving for the new velocity gives

$$\boxed{v_{n+1} = v_n + \frac{h}{2}(a_n + a_{n+1}).} \tag{5}$$

We have obtained *velocity Verlet*. Notice the average of the old and new accelerations in the velocity formula. It is the same averaging that appears in the trapezoidal rule applied to $v_{n+1} - v_n = \int_{t_n}^{t_{n+1}} a(t)\,dt$. This also shows that the one-step velocity defect is $O(h^3)$. Both updates are globally second-order accurate.

There is an important point about the order of calculation. We cannot find a_{n+1} until we know x_{n+1}. So we first update the position, then evaluate the force at that position, and only then update the velocity. Figure 1 shows the sequence.

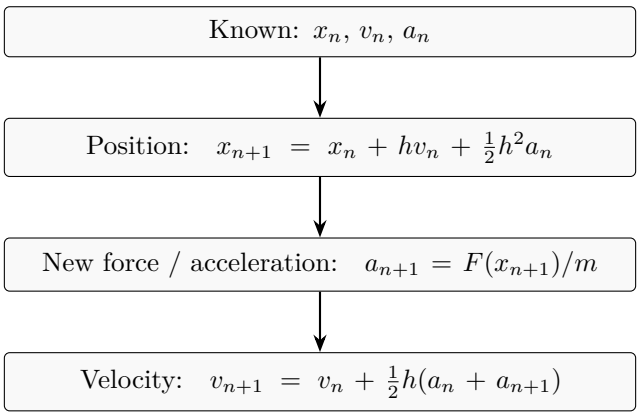


Figure 1: Computational sequence for one velocity Verlet step. The force $F(x_n)$ is reused from the preceding step; only one *new* force evaluation is needed.

> **Velocity Verlet step.** Given x_n, v_n, and a_n: (1) advance the position using Eq. (4); (2) evaluate the new acceleration $a_{n+1} = a(x_{n+1})$; (3) advance the velocity using Eq. (5). Save a_{n+1} for the next step.

4 Leapfrog: velocities at half-integer times

Velocity Verlet keeps position and velocity at the same times. But do we actually need to do that? Suppose instead that we store positions at t_n and velocities halfway between them, at $t_{n+1/2}$.

This may seem an odd choice at first, but it leads to a particularly simple pair of updates. Start with Taylor expansions of the velocity on either side of t_n:

$$v(t_n + h/2) = v(t_n) + \frac{h}{2}a(t_n) + \frac{h^2}{8}\dot{a}(t_n) + \frac{h^3}{48}\ddot{a}(t_n) + O(h^4),$$

$$v(t_n - h/2) = v(t_n) - \frac{h}{2}a(t_n) + \frac{h^2}{8}\dot{a}(t_n) - \frac{h^3}{48}\ddot{a}(t_n) + O(h^4).$$

This time we *subtract* the expansions. The even powers cancel,

$$v(t_n + h/2) - v(t_n - h/2) = ha(t_n) + O(h^3).$$

Dropping the remaining higher-order term gives the half-step velocity update

$$\boxed{v_{n+1/2} = v_{n-1/2} + ha_n.} \tag{6}$$

What about the position? Expand it about the midpoint $t_{n+1/2}$, and subtract the expressions at the two ends of the interval. The result is

$$x(t_{n+1}) - x(t_n) = hv(t_{n+1/2}) + O(h^3),$$

which leads to

$$\boxed{x_{n+1} = x_n + hv_{n+1/2}.} \tag{7}$$

These two equations define the *leapfrog method*. Look at Figure 2: the positions and velocities occupy alternating points on the time axis. That is where the name comes from. Both centered updates have local defects $O(h^3)$ and provide global second-order accuracy.

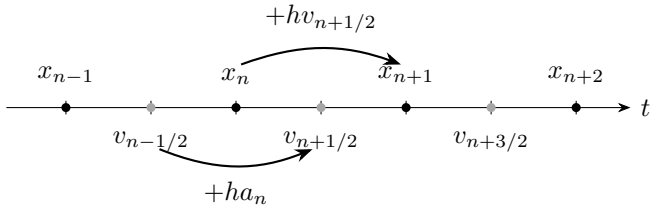


Figure 2: Leapfrog time grid: positions lie at t_n, velocities at $t_{n+1/2}$. A velocity kick is followed by a position drift.

4.1 Starting the leapfrog calculation

There is one detail to take care of before the first step. The initial conditions normally give us x_0 and v_0, not $v_{1/2}$. No problem: evaluate $a_0 = a(x_0)$ and take a half-step in velocity,

$$\boxed{v_{1/2} = v_0 + \frac{h}{2}a_0.} \tag{8}$$

Now we can calculate $x_1 = x_0 + hv_{1/2}$ and continue with full leapfrog steps. If we later need a velocity at an integer time, we can recover it to second-order accuracy by averaging the neighboring half-step velocities:

$$v_n \approx \frac{v_{n-1/2} + v_{n+1/2}}{2}. \tag{9}$$

This distinction matters in molecular dynamics. For example, when we calculate kinetic energy or temperature, we must pay attention to the time at which the velocity is defined.

5 Why leapfrog and velocity Verlet are equivalent

At first sight, leapfrog and velocity Verlet look like different algorithms: one uses half-step velocities, the other full-step velocities. Are they really different methods?

Let us rewrite a velocity Verlet step. First define an intermediate velocity,

$$v_{n+1/2} = v_n + \frac{h}{2}a_n.$$

With this definition, the velocity Verlet position update becomes

$$x_{n+1} = x_n + hv_{n+1/2},$$

That is exactly the leapfrog position update. Once the new position is known, calculate a_{n+1} and take another half-step in velocity:

$$v_{n+1} = v_{n+1/2} + \frac{h}{2}a_{n+1}.$$

Add the two half-steps, and we get

$$v_{n+1} = v_n + \frac{h}{2}(a_n + a_{n+1}),$$

This is precisely the velocity Verlet velocity update. So the apparently new half-step variable has not changed the underlying calculation.

We can also work in the opposite direction. Since $v_{n-1/2} = v_n - \frac{h}{2}a_n$, subtracting the neighboring half-step expressions gives

$$v_{n+1/2} = v_{n-1/2} + ha_n.$$

This is the central result of the note: with consistent initialization, leapfrog and velocity Verlet produce the same positions in exact arithmetic. What changes is *where we store the velocity in time*. Figure 3 makes this connection especially clear.

6 Practical comparison

Which form should we use? That depends mostly on what quantities we want readily available. Velocity Verlet gives positions and velocities at the same times, which can make the analysis of a trajectory easier. Leapfrog has a particularly simple staggered update. Basic position Verlet avoids storing velocities during the update, but requires a previous position.

For molecular interactions, evaluating forces is often the most expensive part of the calculation. If we retain the previous acceleration, velocity Verlet needs only one *new* force calculation per step. Leapfrog has the same advantage after initialization. Both methods are useful for long trajectories of conservative, position-dependent forces. They tend to have good long-time energy behavior, although neither conserves the exact energy at every time step.

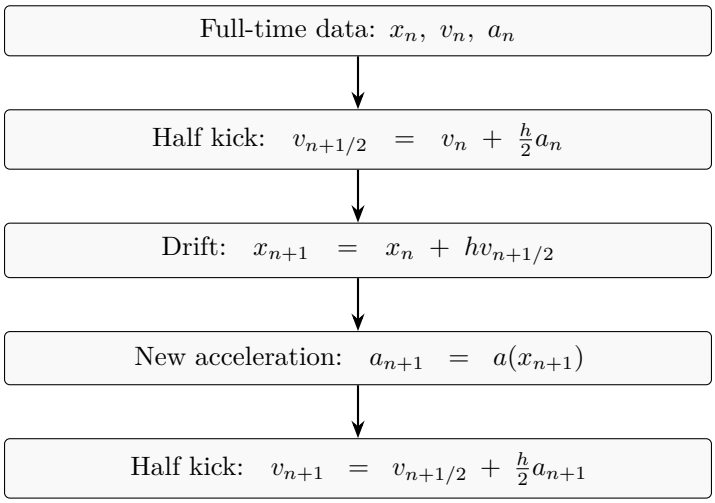


Figure 3: Kick–drift–kick representation of velocity Verlet. The first half kick constructs the leapfrog half-time velocity; the final half kick returns to the full-time velocity.

Table 1: Three forms of the Verlet family for position-dependent forces.

Method	Primary stored data	Practical consideration
Position Verlet	$x_n,\ x_{n-1}$	Velocity must be reconstructed; a previous position must be initialized.
Velocity Verlet	$x_n,\ v_n,\ a_n$	Position and velocity are available at the same time level.
Leapfrog	$x_n,\ v_{n-1/2}$	Compact staggered update; velocities are at half-integer times.

Main result. Position Verlet, leapfrog, and velocity Verlet come from centered Taylor expansions. Leapfrog and velocity Verlet are algebraically equivalent when velocities are related by a half-step kick. The choice between them is mainly a choice of the variables and time levels most convenient for the simulation.

7 A short numerical experiment

We can check the equivalence ourselves with a simple numerical experiment. Take a harmonic oscillator, $\ddot{x} = -\omega^2 x$, with $x_0 = 1$, $v_0 = 0$, and $\omega = 1$. Implement leapfrog and velocity Verlet using the same time step h, and initialize leapfrog with Eq. (8).

Plot the difference between the two calculated positions as a function of time. How small is it? Now compare v_n from velocity Verlet with $v_{n+1/2}$ from leapfrog. Should those velocities agree directly? Use Eq. (9) to compare velocities at the same times. What do you find?