Lecture 8
CH-4114
Molecular Simulation

"Everything that living things do can be understood in terms of the jigglings and wigglings of atoms."

- Richard P. Feynman

By
Dr. Susmita Roy
IISER-Kolkata

Verlet central difference method

or

Verlet Algorithm

$$r_i(t_0 + \Delta t) = 2r_i(t_0) - r_i(t_0 - \Delta t) + f_i(t_0)/m\Delta t^2 + \ldots$$

Positions at t_0

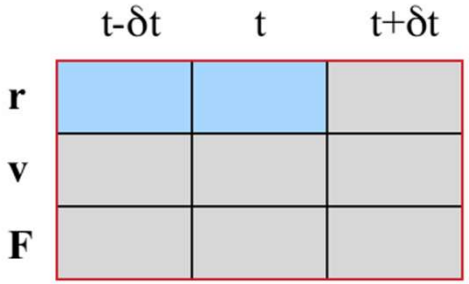
Positions at t_0-Δt

Forces at t_0

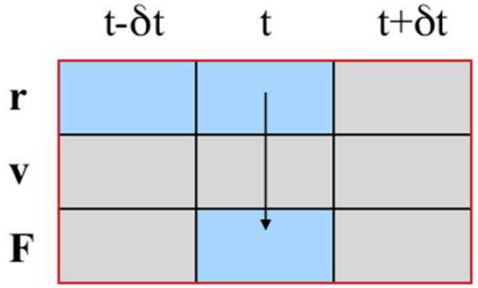
Verlet Algorithm 2. Flow Diagram

$$r_i(t_0 + \Delta t) = 2r_i(t_0) - r_i(t_0 - \Delta t) + f_i(t_0)/m\Delta t^2 + \ldots$$

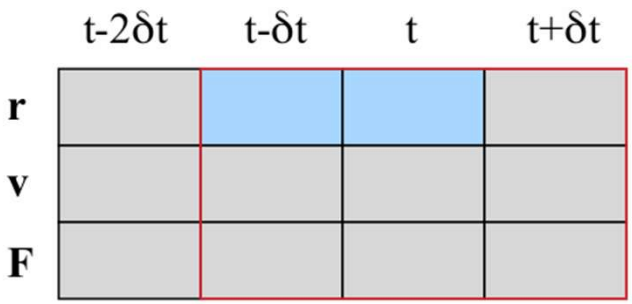
Positions at t_0 | Positions at t_0-Δt | Forces at t_0



Given current position and position at end of previous time step

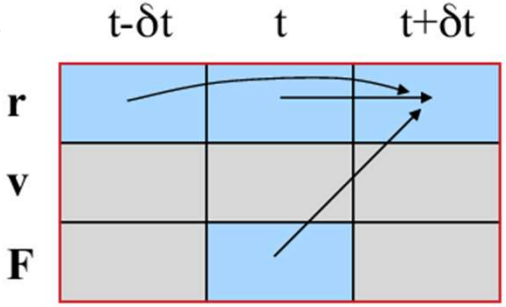


Compute the force at the current position



Current time becomes past time

Advance to next time step, repeat



Compute new position from present and previous positions, and present force

Schematic from Allen & Tildesley, Computer Simulation of Liquids

Verlet Algorithm: Loose Ends

- Initialization
  - how to get position at "previous time step" when starting out?
  - simple approximation

$$\mathbf{r}(t_0 - \delta t) = \mathbf{r}(t_0) - \mathbf{v}(t_0)\delta t$$

- Obtaining the velocities
  - not evaluated during normal course of algorithm
  - needed to compute some properties, e.g.
    - temperature
    - diffusion constant

Verlet Algorithm Performance Issues

- Time reversible
  - forward time step

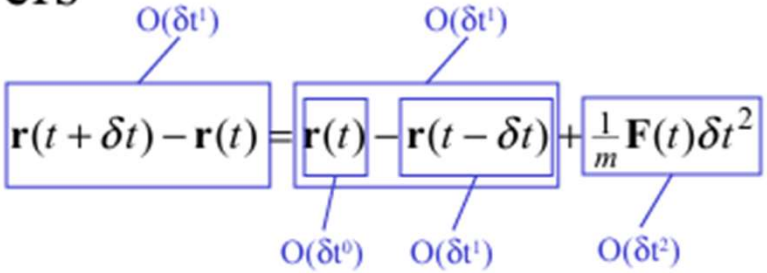
$$\mathbf{r}(t+\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\delta t) + \frac{1}{m}\mathbf{F}(t)\delta t^2$$

  - replace δt with –δt

$$\mathbf{r}(t+(-\delta t)) = 2\mathbf{r}(t) - \mathbf{r}(t-(-\delta t)) + \frac{1}{m}\mathbf{F}(t)(-\delta t)^2$$

$$\mathbf{r}(t-\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t+\delta t) + \frac{1}{m}\mathbf{F}(t)\delta t^2$$

  - same algorithm, with same positions and forces, moves system backward in time
- Numerical imprecision of adding large/small numbers



The core idea behind Leapfrog Algorithm:

The leapfrog algorithm improves numerical stability by staggering position and velocity updates in time (half-step apart), avoiding error-prone subtraction of small terms from large ones, and conserving energy over long simulations.

A naive method (like **forward Euler**) computes:

A native method (like forward Euler) computes:

$$x(t+\delta t) = x(t) + v(t)\delta t$$

$$v(t+\delta t) = v(t) + \frac{F(t)}{m}\delta t$$

- **Positions** are evaluated at integer timesteps: $x(t), x(t+\delta t), x(t+2\delta t), \ldots$
- **Velocities** are evaluated at half-integer timesteps: $v(t+\frac{1}{2}\delta t), v(t+\frac{3}{2}\delta t), \ldots$

So they "leapfrog" over each other in time.

1. **Half-step velocity update:**

$$v\left(t+\tfrac{1}{2}\delta t\right) = v\left(t-\tfrac{1}{2}\delta t\right) + \frac{F(t)}{m}\delta t$$

2. **Position update:**

$$x(t+\delta t) = x(t) + v\left(t+\tfrac{1}{2}\delta t\right)\delta t$$