Molecular dynamics simulation

Bettina Keller, Annika Bande

Collection of Generic Algorithms

(Chapters 1 and 3)

Generic algorithm for integrating the Newton equations of motion

Data : initial position initialPosition, initial velocity initialVelocity, number of integration steps NT, time step deltaT, mass m

Result : position and velocity trajectories: position[], velocity[]

position[0] = initialPosition;
velocity[0] = initialVelocity;

for $t = 0$ **to** $t = NT - 1$ **do**
    force = `evaluateForce` [position[t]];
    update positions ;
    update velocities ;
end

Algorithmus 1: Generic algorithm for integrating the Newton equations of motion.

Euler method for integrating the Newton equations of motion

Data : initial position initialPosition, initial velocity initialVelocity, number of integration steps NT, time step deltaT, mass m

Result : position and velocity trajectories: position[], velocity[]

position[0] = initialPosition;
velocity[0] = initialVelocity;

for $t = 0$ **to** $t = NT - 1$ **do**
    force = `evaluateForce` [position[t]];
    position[t+1] = position[t] + velocity[t]·deltaT + (force/(2m))·deltaT·deltaT;
    velocity[t+1] = velocity[t] + (force/m)·deltaT;
end

Algorithmus 2: Euler method for integrating the Newton equations of motion

Verlet algorithm

Data : initial position initialPosition, initial velocity initialVelocity, number of integration steps NT, time step deltaT, mass m

Result : position and velocity trajectories: position[], velocity[]

position[0] = initialPosition;
velocity[0] = initialVelocity;

force = `evaluateForce` [position[0]];
postion[1] = position[0] + velocity[0]·deltaT + (force/(2m))·deltaT·deltaT;

for $t = 1$ **to** $t = NT - 1$ **do**
    force = `evaluateForce` [position[t]];
    position[t+1] = 2position[t] - position[t-1] + (force/(2m))·deltaT·deltaT;
    velocity[t] = (position[t+1] -position[t-1])/(2·deltaT);
end

Algorithmus 3: Verlet algorithm for integrating the Newton equations of motion

Leap frog algorithm

Data : initial position initialPosition, initial velocity initialVelocity, number of integration steps NT, time step deltaT, mass m

Result : position and velocity trajectories: position[], velocity[]

position[0] = initialPosition;
velocity[0] = initialVelocity;

force = `evaluateForce` [position[0]];
velocityHalfSteps[0] = initialVelocity - (force/m)·(deltaT/2);
velocityHalfSteps[1] = initialVelocity + (force/m)·(deltaT/2);

for $t = 0$ **to** $t = NT - 1$ **do**
    force = `evaluateForce` [position[t]];
    velocityHalfSteps[t+1] = velocityHalfSteps[t] + (force/m)·deltaT ;
    position[t+1] = position[t] + velocityHalfSteps[t+1] ·deltaT;
    velocity[t+1] = (velocityHalfSteps[t+1]-velocityHalfSteps[t])/deltaT;
end

Algorithmus 4: Leap frog algorithm for integrating the Newton equations of motion

Velocity Verlet algorithm

Data : initial position initialPosition, initial velocity initialVelocity, number of integration steps NT, time step deltaT, mass m

Result : position and velocity trajectories: position[], velocity[]

position[0] = initialPosition;
velocity[0] = initialVelocity;

for $t = 0$ **to** $t = NT - 1$ **do**

force01 = `evaluateForce` [position[t]];
position[t+1] = position[t] + velocity[t] ·deltaT + (force01/m)·deltaT·deltaT;
force02 = `evaluateForce` [position[t+1]];
velocity[t+1] = velocity[t] + ((force01+force02)/(2m))·deltaT;

end

Algorithmus 5: Velocity Verlet algorithm for integrating the Newton equations of motion