MD Simulation (Argon gas)

Simulation practice 3

Nima yadollahi – 40111046

Introduction

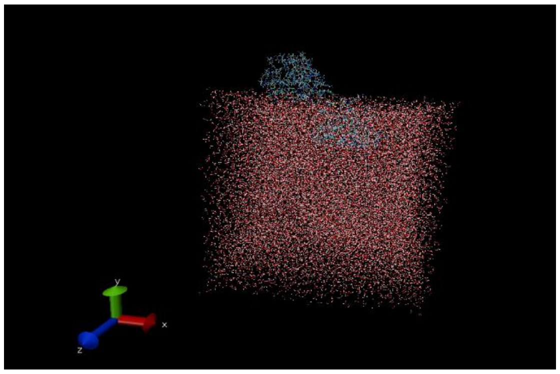
Molecular Dynamics (MD) simulations are a powerful tool used in computational chemistry, materials science, and biophysics to model the behavior of molecules over time. These simulations allow researchers to examine how atoms and molecules move and interact under various conditions, such as temperature, pressure, and external forces.

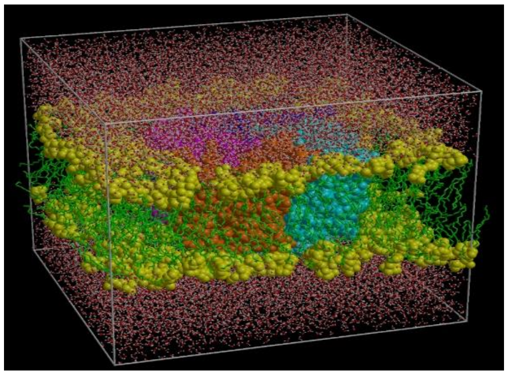
In MD simulations, atoms are treated as classical particles, interacting through potential energy functions. The primary goal of MD is to compute the time evolution of a system's atomic positions and velocities.

MD simulations have broad applications in various scientific fields, including drug discovery, protein folding, materials science, and nanotechnology.

The Verlet algorithm is a numerical method used to integrate Newton's equations of motion in molecular dynamics (MD) simulations. It was first introduced in 1791 by Jean Baptiste Delambre and has been rediscovered many times since then, most recently by Loup Verlet in the 1960s for molecular dynamics.

This algorithm works by updating the position of each particle in the system based on its previous position, current velocity, and the forces acting upon it.





Abstract & Relations

The Lennard-Jones potential is a mathematical model widely used to describe the interaction between a pair of neutral atoms or molecules. Formulated in 1924 by Sir John Lennard-Jones, the potential captures the balance between attractive and repulsive forces that govern intermolecular interactions and is expressed as:

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

r : distance between particles

ε : depth of the potential well

σ : The distance at which the particle-particle potential energy V is zero

We know that the dynamics of a classical many-body system are described by the Hamiltonian and from the Hamiltonian's principle force equals :

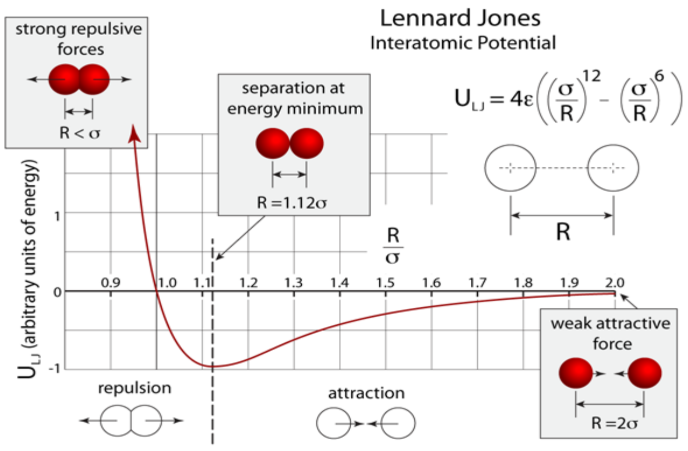
$$\dot{p}_i = -\frac{\partial H}{\partial q_i} = f_j$$

$$f_i = -\nabla U - \sum_{j\neq i} \nabla V(r_{ij})$$

In this question, we don't have external energy, so the first part of the equation is zero, and we can write the force:

$$f = \frac{24\varepsilon}{\sigma^2}\left[2\left(\frac{\sigma}{r}\right)^{14} - \left(\frac{\sigma}{r}\right)^{8}\right]$$

If r < r_{cutoff} , Lennard-Jones potential and force are assumed.



If we apply the Verlet algorithm to this terminology, we can calculate the position and velocity of the particle at the next time step.

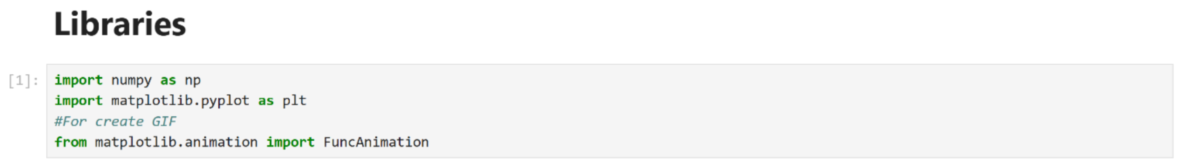
$$R_{k+1} = R_k + dt v_k + \frac{dt^2}{2} f_k$$

$$v_{k+1} = v_k + \frac{dt}{2}(f_{k+1} + f_k)$$

For the calculation of the energy of the system (E = T + V), we must find the kinetic energy of the system so:

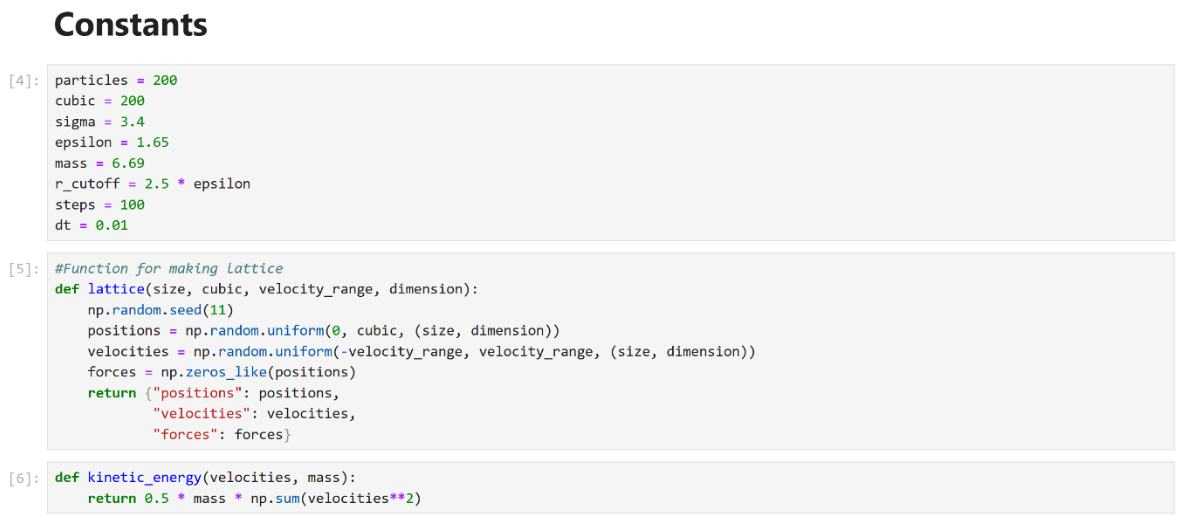
$$T = \frac{1}{2}\sum_{i=1} m_i v_i^2$$

Python code & results



Libraries

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     #For create GIF
     from matplotlib.animation import FuncAnimation
```



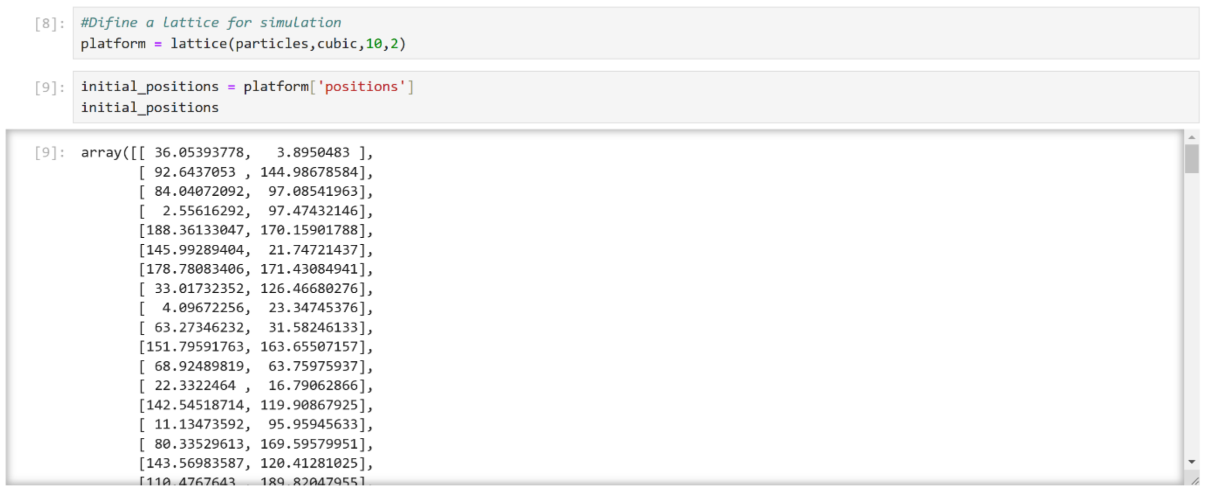
Constants

```
[4]: particles = 200
     cubic = 200
     sigma = 3.4
     epsilon = 1.65
     mass = 6.69
     r_cutoff = 2.5 * epsilon
     steps = 100
     dt = 0.01
```

```
[5]: #Function for making lattice
     def lattice(size, cubic, velocity_range, dimension):
         np.random.seed(11)
         positions = np.random.uniform(0, cubic, (size, dimension))
         velocities = np.random.uniform(-velocity_range, velocity_range, (size, dimension))
         forces = np.zeros_like(positions)
         return {"positions": positions,
                 "velocities": velocities,
                 "forces": forces}
```

```
[6]: def kinetic_energy(velocities, mass):
         return 0.5 * mass * np.sum(velocities**2)
```

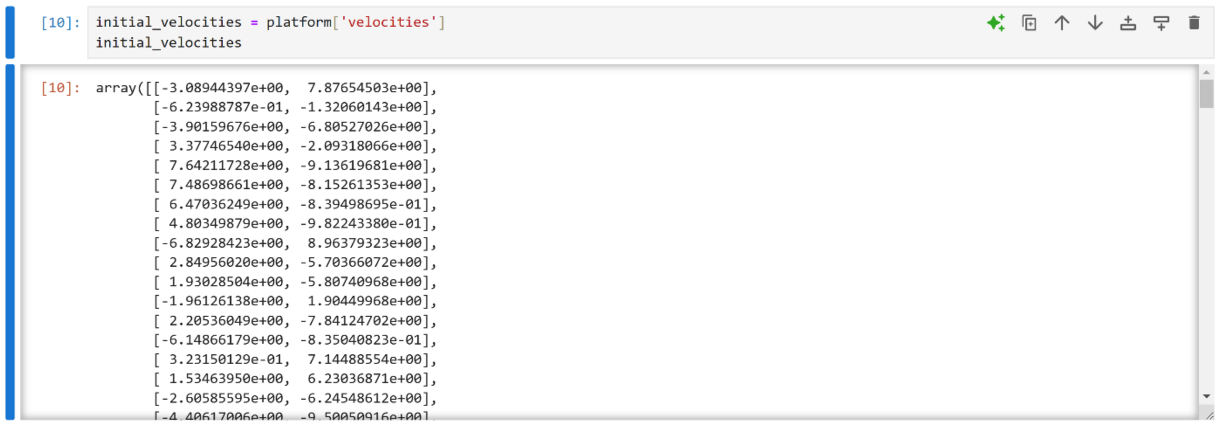
Initial 2D-Lattice



```
[8]: #Difine a lattice for simulation
     platform = lattice(particles,cubic,10,2)
```

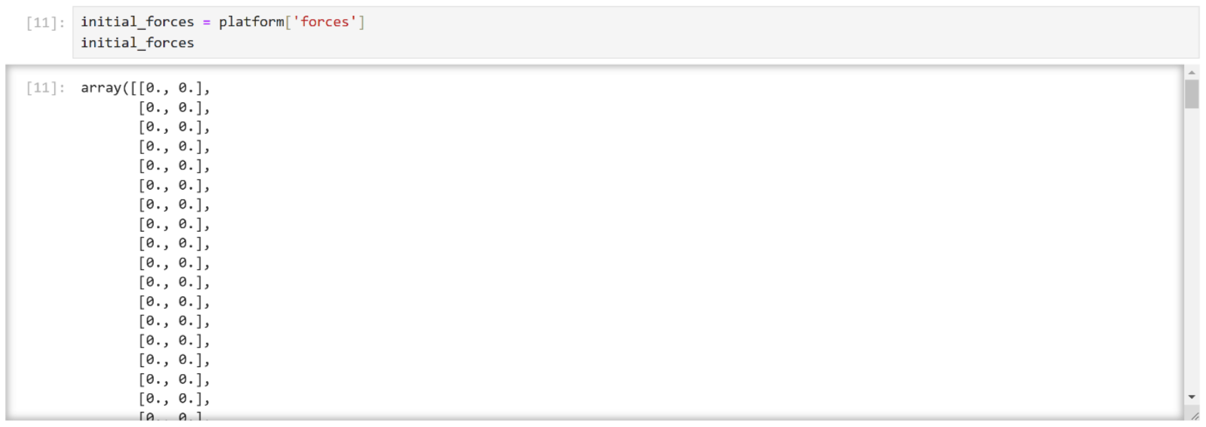
```
[9]: initial_positions = platform['positions']
     initial_positions
```

```
[9]: array([[ 36.05393778,   3.8950483 ],
            [ 92.6437053 , 144.98678584],
            [ 84.04072092,  97.08541963],
            [  2.55616292,  97.47432146],
            [188.36133047, 170.15901788],
            [145.99289404,  21.74721437],
            [178.78083406, 171.43084941],
            [ 33.01732352, 126.46680276],
            [  4.09672256,  23.34745376],
            [ 63.27346232,  31.58246133],
            [151.79591763, 163.65507157],
            [ 68.92489819,  63.75975937],
            [ 22.3322464 ,  16.79062866],
            [142.54518714, 119.90867925],
            [ 11.13473592,  95.95945633],
            [ 80.33529613, 169.59579951],
            [143.56983587, 120.41281025],
            [110.4767643 , 189.82047955],
```



```
[10]: initial_velocities = platform['velocities']
      initial_velocities
```

```
[10]: array([[-3.08944397e+00,  7.87654503e+00],
             [-6.23988787e-01, -1.32060143e+00],
             [-3.90159676e+00, -6.80527670e+00],
             [ 3.37746540e+00, -2.09318066e+00],
             [ 7.64211728e+00, -9.13619681e+00],
             [ 7.48698661e+00, -8.15261353e+00],
             [ 6.47036249e+00, -8.39498695e-01],
             [ 4.80349879e+00, -9.82243380e-01],
             [-6.82928423e+00,  8.96379323e+00],
             [ 2.84956020e+00, -5.70366072e+00],
             [ 1.93028504e+00, -5.80740968e+00],
             [-1.96126138e+00,  1.90449968e+00],
             [ 2.20536049e+00, -7.84124702e+00],
             [-6.14866179e+00, -8.35040823e-01],
             [ 3.23150129e-01,  7.14488554e+00],
             [ 1.53463950e+00,  6.23036871e+00],
             [-2.60585595e+00, -6.24548612e+00],
             [-4.40617006e+00, -9.50050916e+00],
```



```
[11]: initial_forces = platform['forces']
      initial_forces
```

```
[11]: array([[0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
             [0., 0.],
[12]: def lennard_jones(radius, epsilon, sigma):
          if radius > 1e-10:
              const = sigma / radius
              lejo_potential = 4 * epsilon * ( (const)**12 - (const)**6 )
              lejo_force = ((24 * epsilon ) / sigma**2) * (2*(const)**14 - (const)**8)
              return {"lennard_jones_potential" : lejo_potential,
                      "lennard_jones_force" : lejo_force}

          else:
              print("Warning: Zero distance encountered. Skipping particle pair.")
              return {"lennard_jones_potential" : 0,
                      "lennard_jones_force" : 0}
```

```
[13]: # Example
      lennard_jones(5,epsilon,sigma)
```

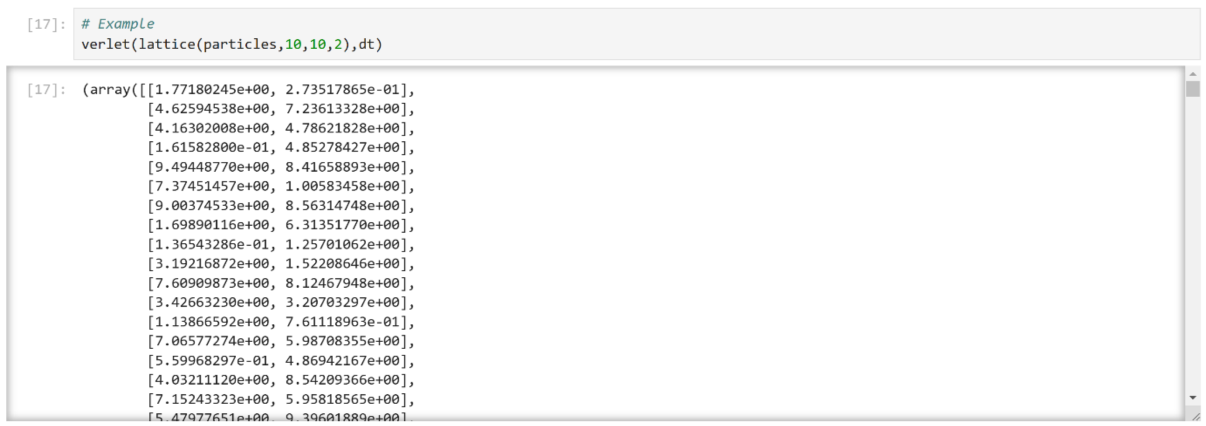
```
[13]: {'lennard_jones_potential': -0.5880118431237138,
       'lennard_jones_force': -0.12563959222296675}
```

```
[14]: def compute_force(lattice):
          forces = np.zeros_like(lattice)
          potential_energy = 0
          particles = lattice.shape[0]
          for i in range(particles):
              for j in range(i + 1, particles):
                  r_vec = lattice[j] - lattice[i]
                  r_vec -= cubic * np.round(r_vec / cubic)
                  r = np.linalg.norm(r_vec)
                  if (r < r_cutoff) and (r>1e-10) :
                      potential = lennard_jones(r, epsilon, sigma)['lennard_jones_potential']
                      f_mag = lennard_jones(r, epsilon, sigma)['lennard_jones_force']
                      potential_energy += potential
                      f_vec = f_mag * (r_vec / r)
                      forces[i] += f_vec
                      forces[j] -= f_vec
          return forces , potential_energy
```

```
[15]: # Example
      compute_force(initial_positions)
```

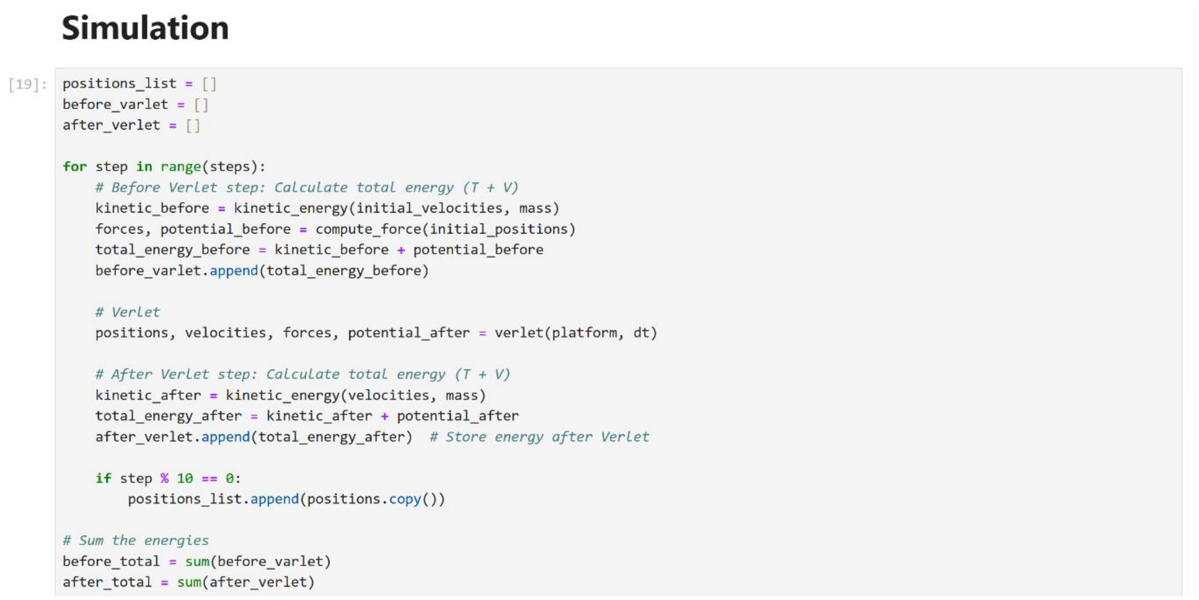
```
[15]: (array([[ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 5.21599869e+02,  1.09753480e+03],
              [ 7.32597443e+05,  6.15443184e+05],
              [ 1.86088235e+03,  1.17939336e+03],
              [ 1.15625918e+00,  1.15090919e+00],
              [ 2.64253550e+07,  1.30013733e+07],
              [ 0.00000000e+00,  0.00000000e+00],
              [ 0.00000000e+00,  0.00000000e+00],
              [-2.64253550e+07, -1.30013733e+07],
              [ 0.00000000e+00,  0.00000000e+00],
```

```
[16]: def verlet(lattice , dt):
          positions = lattice['positions']
          velocities = lattice['velocities']
          forces = lattice['forces']
          positions += velocities * dt + 0.5 * forces * dt**2
          positions %= cubic
          new_forces = compute_force(positions)[0]
          potential_energy = compute_force(positions)[1]
          velocities += 0.5 * (forces + new_forces) * dt
          return positions, velocities, new_forces , potential_energy
```



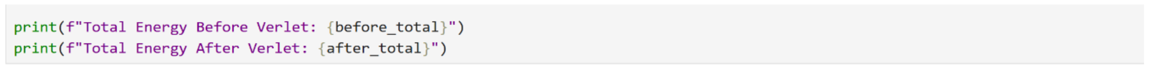
```
[17]:  # Example
       verlet(lattice(particles,10,10,2),dt)
```

```
[17]:  (array([[1.77180245e+00, 2.73517865e-01],
               [4.62594538e+00, 7.23613328e+00],
               [4.16302008e+00, 4.78621828e+00],
               [1.61582800e-01, 4.85278427e+00],
               [9.49448770e+00, 8.41658893e+00],
               [7.37451457e+00, 1.00583458e+00],
               [9.00374533e+00, 8.56314748e+00],
               [1.69890116e+00, 6.31351770e+00],
               [1.36543286e-01, 1.25701062e+00],
               [3.19216872e+00, 1.52208646e+00],
               [7.60909873e+00, 8.12467948e+00],
               [3.42663230e+00, 3.20703297e+00],
               [1.13866592e+00, 7.61118963e-01],
               [7.06577274e+00, 5.98708355e+00],
               [5.59968297e-01, 4.86942167e+00],
               [4.03211120e+00, 8.54209366e+00],
               [7.15243323e+00, 5.95818565e+00],
               [5.47977651e+00, 9.39601889e+00],
```



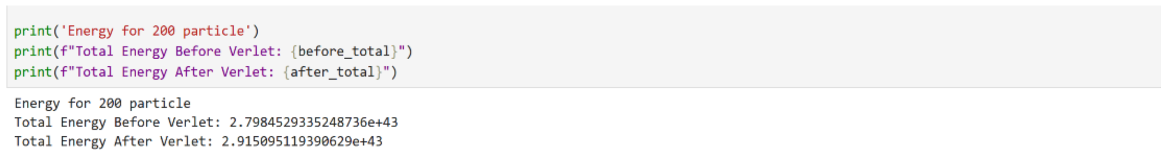
Simulation

```
[19]:  positions_list = []
       before_varlet = []
       after_verlet = []

       for step in range(steps):
           # Before Verlet step: Calculate total energy (T + V)
           kinetic_before = kinetic_energy(initial_velocities, mass)
           forces, potential_before = compute_force(initial_positions)
           total_energy_before = kinetic_before + potential_before
           before_varlet.append(total_energy_before)

           # Verlet
           positions, velocities, forces, potential_after = verlet(platform, dt)

           # After Verlet step: Calculate total energy (T + V)
           kinetic_after = kinetic_energy(velocities, mass)
           total_energy_after = kinetic_after + potential_after
           after_verlet.append(total_energy_after)  # Store energy after Verlet

           if step % 10 == 0:
               positions_list.append(positions.copy())

       # Sum the energies
       before_total = sum(before_varlet)
       after_total = sum(after_verlet)
```



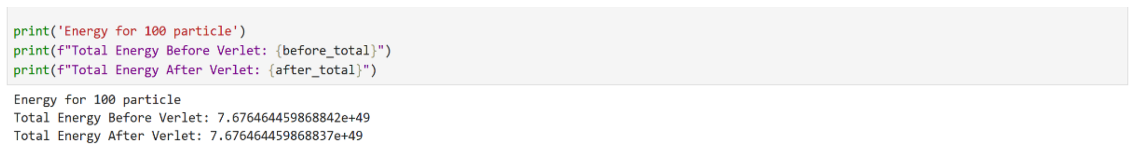
```
print(f"Total Energy Before Verlet: {before_total}")
print(f"Total Energy After Verlet: {after_total}")
```

In this section, we calculate energy to determine whether it is conserved.



```
print('Energy for 200 particle')
print(f"Total Energy Before Verlet: {before_total}")
print(f"Total Energy After Verlet: {after_total}")
```

```
Energy for 200 particle
Total Energy Before Verlet: 2.7984529335248736e+43
Total Energy After Verlet: 2.915095119390629e+43
```



```
print('Energy for 100 particle')
print(f"Total Energy Before Verlet: {before_total}")
print(f"Total Energy After Verlet: {after_total}")
```

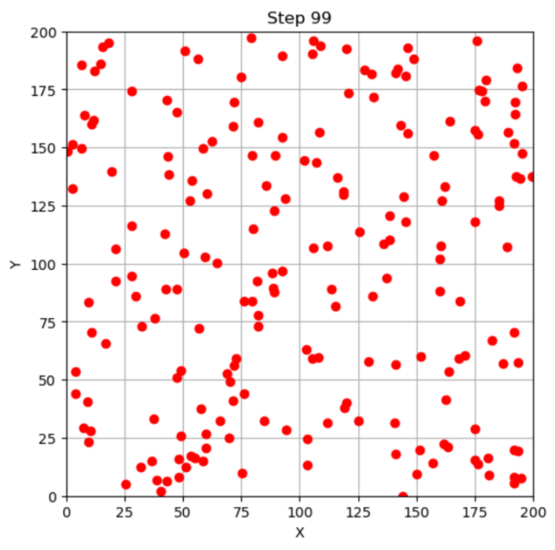
```
Energy for 100 particle
Total Energy Before Verlet: 7.676464459868842e+49
Total Energy After Verlet: 7.676464459868837e+49
```

```
print('Energy for 50 particle')
print(f"Total Energy Before Verlet: {before_total}")
print(f"Total Energy After Verlet: {after_total}")
```

```
Energy for 50 particle
Total Energy Before Verlet: 5.788832783325279e+20
Total Energy After Verlet: 5.8441082566632014e+20
```

Visualizing

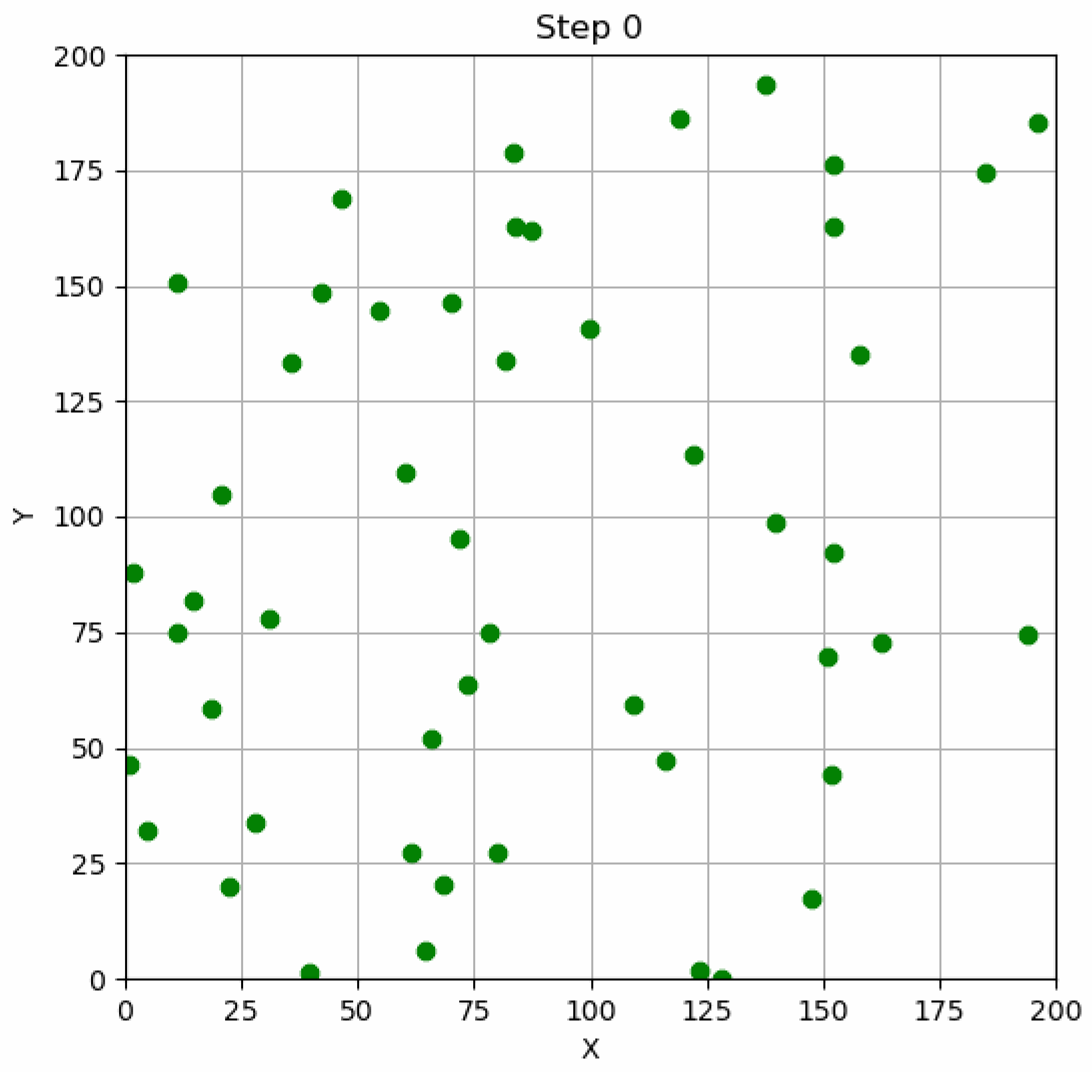
```
[20]: positions_list = []
      forces = compute_force(initial_positions)
      for step in range(steps):
          positions = verlet(platform, dt)[0]
          velocities = verlet(platform, dt)[1]
          forces = verlet(platform, dt)[2]
          positions_list.append(positions.copy())

      fig, ax = plt.subplots(figsize=(6, 6))

      ax.set_xlim(0, cubic)
      ax.set_ylim(0, cubic)
      ax.grid(True)

      ax.set_xlabel("X")
      ax.set_ylabel("Y")
      ax.set_title("Particle Simulation")

      particles, = ax.plot([], [], 'ro', markersize=6)

      def update(frame):
          positions = positions_list[frame]
          particles.set_data(positions[:, 0], positions[:, 1])
          ax.set_title(f"Step {frame}")
          return particles,

      ani = FuncAnimation(fig, update, frames=len(positions_list), interval=5000, blit=True)

      ani.save("MD simulation for 200 particle.gif", writer="pillow", fps=300)

      plt.show()
      print("GIF saved as 'MD simulation for 200 particle.gif'")
```



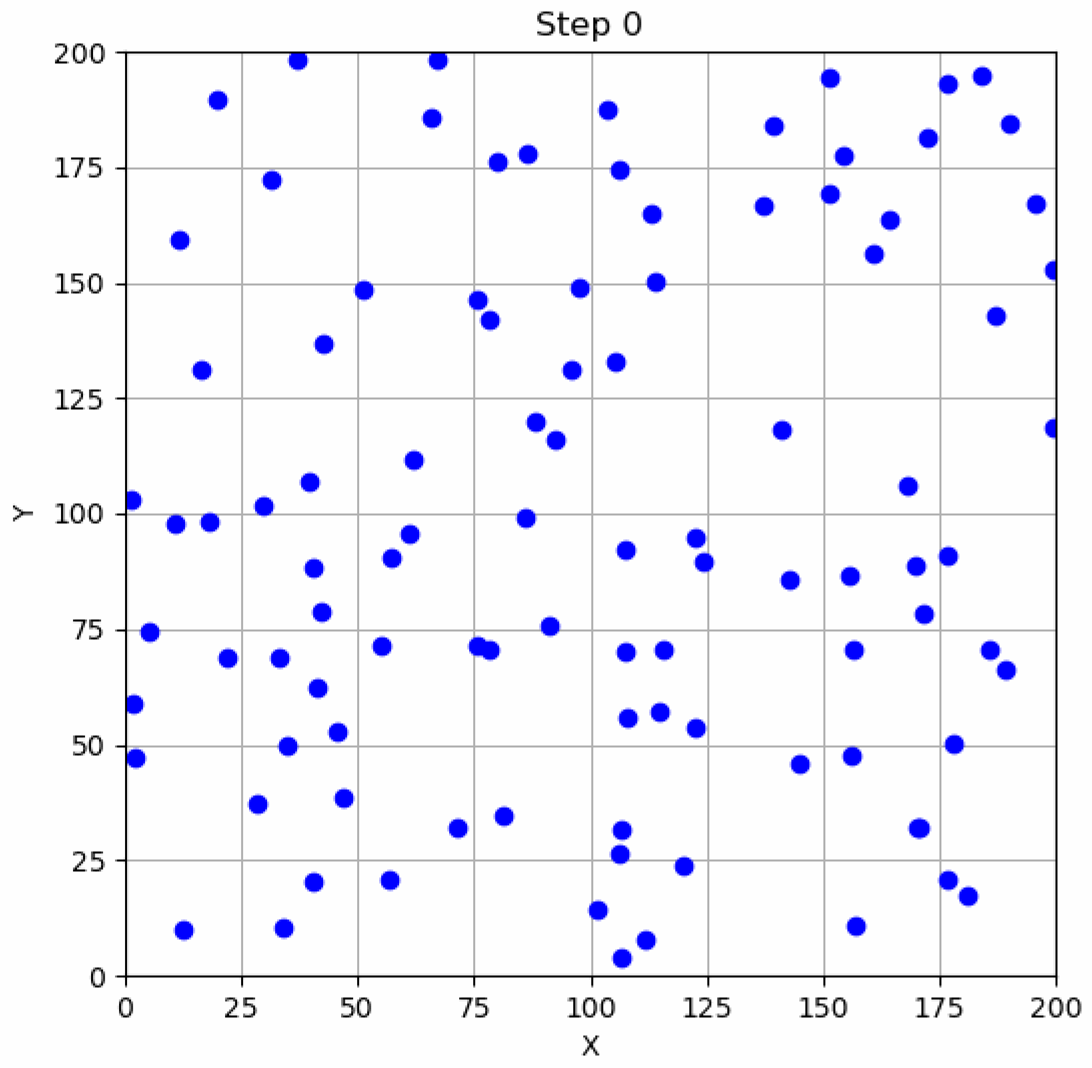
```
GIF saved as 'MD simulation for 200 particle.gif'
```

Animations for dt = 0.01

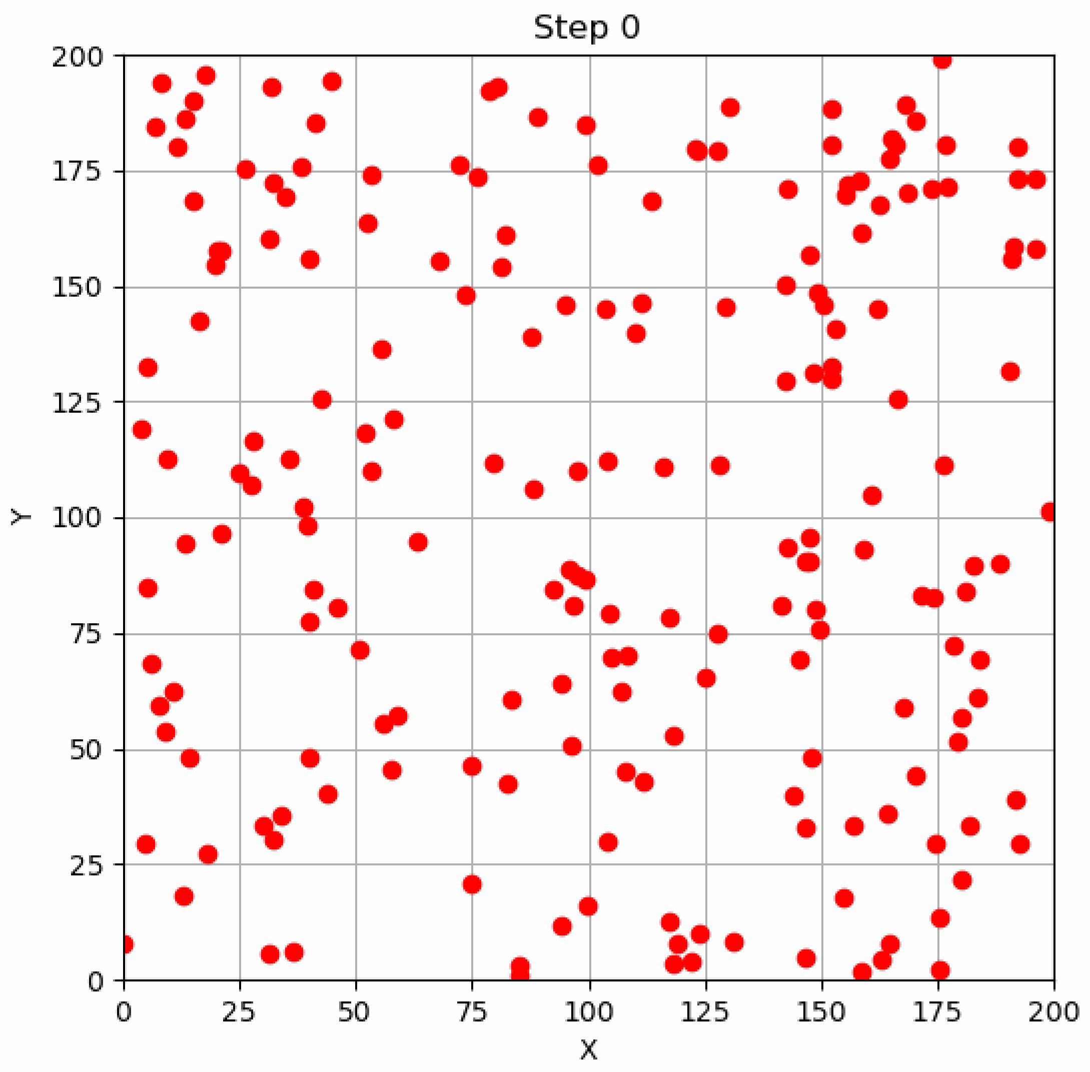
50 particle



100 particle

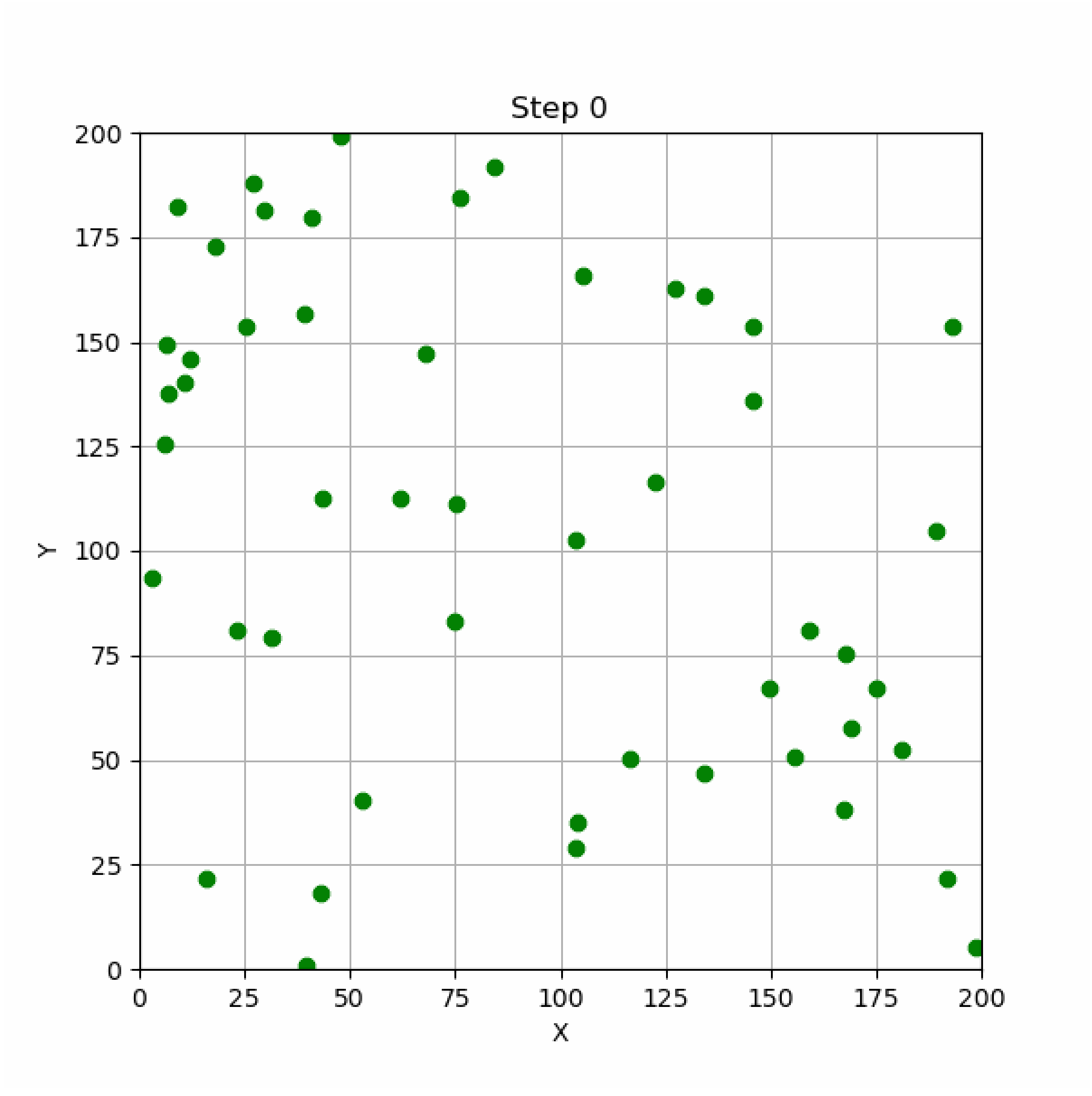


200 particle

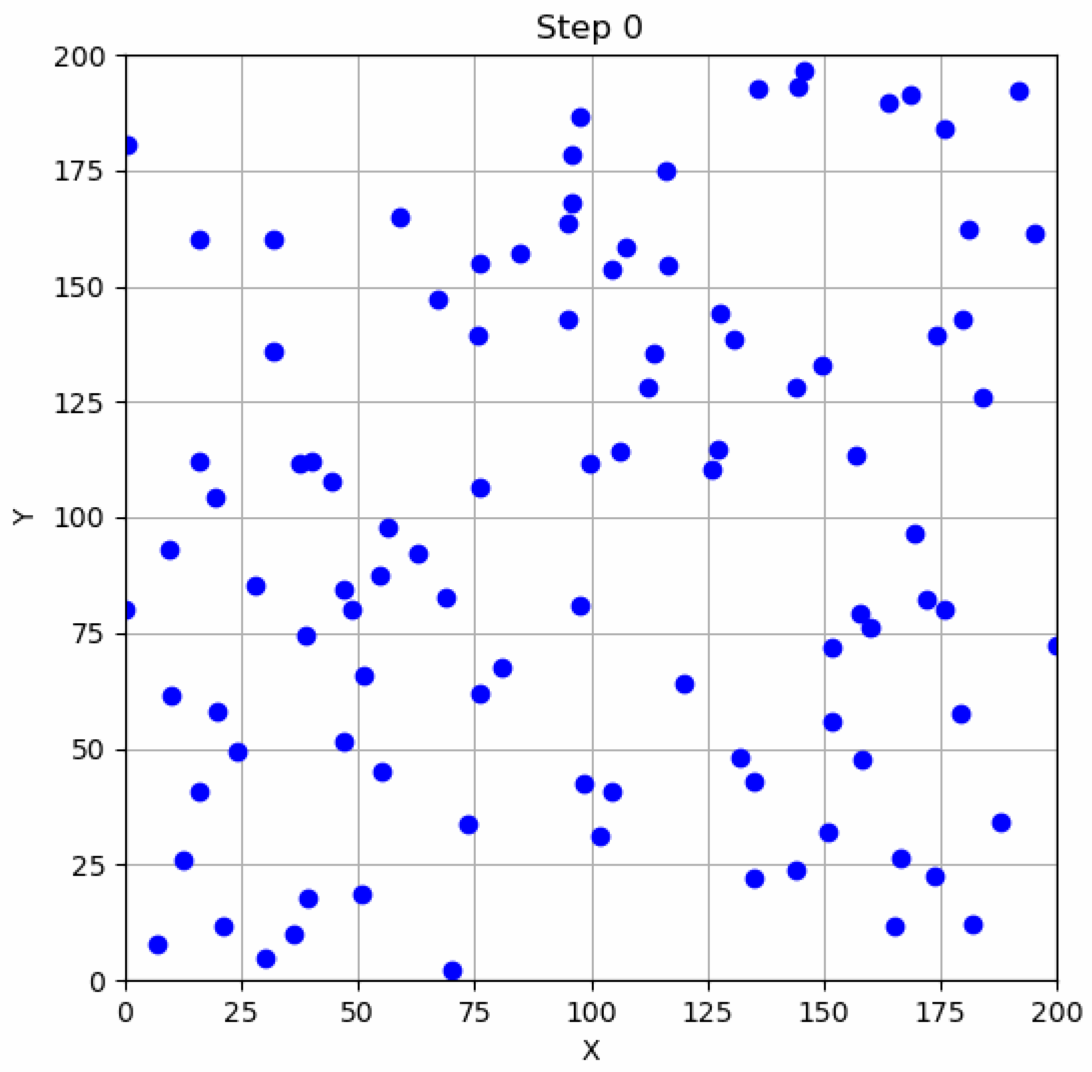


Animations for dt = 0.1

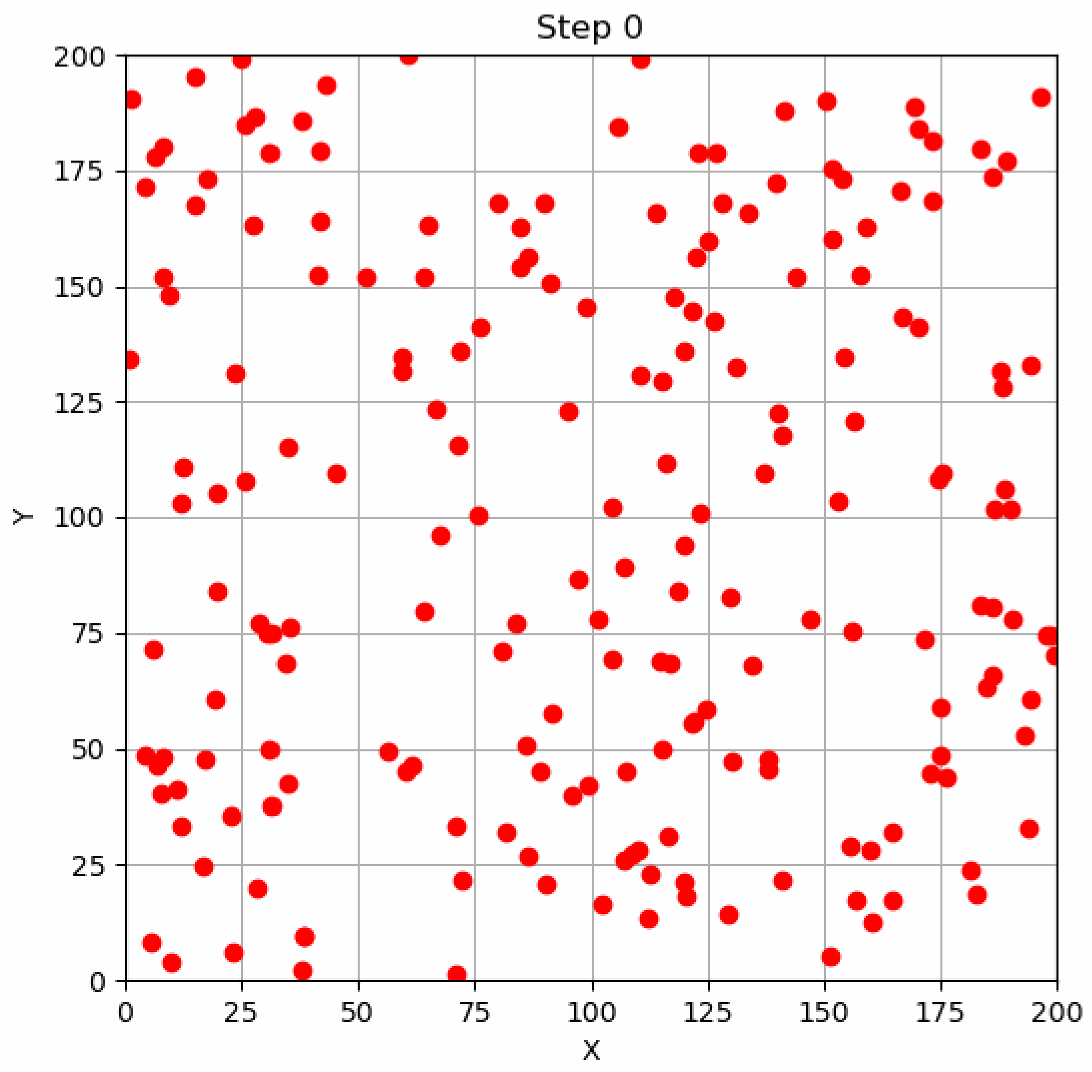
50 particle



100 particle



200 particle


Step 0
Y
X