Molecular Dynamics Simulation of a Van der Waals Gas

Generated by Doxygen 1.10.0

Chapter 1

Project-III: Simulation of a Van der Waals gas using Molecular Dynamics.

1.1 Brief description of the project

This project involves the development of a Molecular Dynamics code to simulate a Van der Waals gas, performing calculations in both serial and parallel modes.

Additionally, this project requires collaborative work to ensure the proper functioning of the code.

1.2 Team Members and Responsibilities

Each team member has assigned tasks, which are indicated below:

1. **Anna Monclús (@anna-mr98)**: Initialize the configuration and define boundary conditions. Also coordinates the GitHub repository.

1. **Aina Gaya (@ainagaya)**: Integration Newton's equations.
2. **Albert Plazas (@Alplalo)**: Compute the forces for a Van der Waals interaction.
3. **Manel Serrano (@gluoon8)**: Post-processing of data, statistics, and visualization.

1.3 Prerequisites

To execute the program, there are some pre-requisites:

- Make: to execute the program (`https://www.gnu.org/software/make/#download`)
- Gfortran: to run the MD simulation. (`https://gcc.gnu.org/wiki/GFortran`)
- Python 3.x : to generate the plots after simulation
  - Numpy (`https://numpy.org/install/`)
  - Matplotlib (`https://matplotlib.org/stable/users/installing/index.html`)

1.4 How to

> [!IMPORTANT] Current features are only available for serial code, parallel code is WIP!

1. Clone repository to your local host
2. Choose *serial* or *parallel (WIP)* folder with `cd serial` or `cd parallel`
3. Use `make` or `make help` to see the available commands.
4. Before starting a simulation, change your parameters in *namMD.nml* file

5. To carry out the simulation, use `make run` and the program will be compiled and run.
6. Data is generated in ∗.dat∗ files. If you want to generate figures, use `make plot`

1.4.1 Quick guide

To carry out a simulation after choosing parameters in *namMD.nml* file you can use:

```
make run
make plot
```

And files will appear in your main directory!

> [!TIP] There are some ways to clean generated files, have a look at `make clean`, `make cleandata` and `make cleanplot`.

1.5 Help

- Commands:
  - `make run`: Compiles needed files and also runs the program.
  - `make plot`: Plots the output data:
    - Epot, Ekin, Etot vs time
    - Momentum vs time
    - T vs time
    - Pressure vs time
  - `make all`: Compiles the program and creates executable MD.exe
  - `make clean`: Removes the modules, objects and executable
  - `make cleandata`: Removes data files
  - `make cleanplot`: Removes plot files

1.6 Contributors

Anna Monclús	Aina Gaya	Albert Plazas	Manel Serrano
anna-mr98	ainagaya	Alplalo	gluoon8

Work developed in the Advanced Informatic Tools subject from Master of Atomistic and Multiscale Computational Modelling in Physics, Chemistry and Biochemistry.

Universitat de Barcelona	Universitat Politècnica de Catalunya

Chapter 2

Modules Index

2.1 Modules List

Here is a list of all documented modules with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 forces Module Reference

Module containing the subroutine `find_force_LJ` which calculates the forces and potential energy between particles using the Lennard-Jones potential.

Functions/Subroutines

- subroutine, public find_force_lj (r, n, l, cutoff, f, pot, ppot, nprocs, rank, counts_recv, displs_recv, imin, imax)

  Calculates the forces and potential energy between particles using the Lennard-Jones potential (parallel implementation).
- subroutine, public find_force_lj (r, n, l, cutoff, f, pot, ppot)

  Calculates the forces and potential energy between particles using the Lennard-Jones potential (serial implementation).

4.1.1 Detailed Description

Module containing the subroutine `find_force_LJ` which calculates the forces and potential energy between particles using the Lennard-Jones potential.

This module contains the subroutine `find_force_LJ` which calculates the forces and potential energy between particles using the Lennard-Jones potential.

4.1.2 Function/Subroutine Documentation

4.1.2.1 find_force_lj() `[1/2]`

```
subroutine, public forces::find_force_lj (
            real(8), dimension(n, 3), intent(in) r,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), dimension(n, 3), intent(out) f,
            real(8), intent(out) pot,
            real(8), intent(out) ppot )
```

Calculates the forces and potential energy between particles using the Lennard-Jones potential (serial implementation).

This subroutine calculates the forces and potential energy between particles in a system using the Lennard-Jones potential. The Lennard-Jones potential is a pairwise potential that models the interaction between neutral atoms or molecules. It is commonly used to simulate the behavior of noble gases and other non-reactive systems.

Notes:

- The positions of the particles are given as a 3D array, where each row represents the position of a particle in Cartesian coordinates.
- The forces acting on each particle are calculated and stored in the `F` array.
- The potential energy of the system is calculated and stored in the `pot` variable.
- The subroutine uses periodic boundary conditions to handle particles that are close to the edges of the simulation box.
- The subroutine assumes that the `pbc` subroutine is defined elsewhere in the code, which handles the periodic boundary conditions.
- The subroutine assumes that the `isnan` function is available to check for NaN values.

4.1.2.2 find_force_lj() [2/2]

```
subroutine, public forces::find_force_lj (
            real(8), dimension(n, 3), intent(in) r,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), dimension(n, 3), intent(out) f,
            real(8), intent(out) pot,
            real(8), intent(out) ppot,
            integer, intent(in) nprocs,
            integer, intent(in) rank,
            integer, dimension(0:nprocs) counts_recv,
            integer, dimension(0:nprocs-1) displs_recv,
            integer imin,
            integer imax )
```

Calculates the forces and potential energy between particles using the Lennard-Jones potential (parallel implementation).

This subroutine calculates the forces and potential energy between particles in a system using the Lennard-Jones potential. The Lennard-Jones potential is a pairwise potential that models the interaction between neutral atoms or molecules. It is commonly used to simulate the behavior of noble gases and other non-reactive systems.

Parameters: r: real(8), dimension(N, 3), intent(in)

- The positions of the particles in the system. N: integer, intent(in)
- The number of particles in the system. L: real(8), intent(in)
- The length of the simulation box. cutoff: real(8), intent(in)

- The cutoff distance for the Lennard-Jones potential. F: real(8), dimension(N, 3), intent(out)
- The forces acting on each particle. pot: real(8), intent(out)
- The potential energy of the system.

Notes:

- The positions of the particles are given as a 3D array, where each row represents the position of a particle in Cartesian coordinates.
- The forces acting on each particle are calculated and stored in the `F` array.
- The potential energy of the system is calculated and stored in the `pot` variable.
- The subroutine uses periodic boundary conditions to handle particles that are close to the edges of the simulation box.
- The subroutine assumes that the `pbc` subroutine is defined elsewhere in the code, which handles the periodic boundary conditions.
- The subroutine assumes that the `isnan` function is available to check for NaN values.

4.2 initialization Module Reference

Module containing initialization rutines for molecular dynamics simulations.

Functions/Subroutines

- subroutine initialize_positions (n, rho, r)

  *Initializes the positions of N particles in a cubic box with density rho. The box is centered at the origin. The positions are stored in the array r. The length of the box is calculated as L = (N/rho)**(1/3). The box is divided in M = N**(1/3) cells. The length of each cell is a = L/M.*
- subroutine initialize_velocities (n, v_ini, v)

  Initializes the velocities of N particles with a given velocity v_ini. The velocities are stored in the array v. The velocities are chosen randomly from a bimodal distribution.
- subroutine distribute_particles (n, rank, nprocs, imin, imax)

  Distributes the particles among the processors.

4.2.1 Detailed Description

Module containing initialization rutines for molecular dynamics simulations.

4.2.2 Function/Subroutine Documentation

4.2.2.1 distribute_particles()

```
subroutine initialization::distribute_particles (
            integer, intent(in) n,
            integer, intent(in) rank,
            integer, intent(in) nprocs,
            integer imin,
            integer imax )
```

Distributes the particles among the processors.

Parameters

N	Number of particles.
rank	Rank of the processor.
nprocs	Number of processors.
imin,imax	Indexes of the particles to be handled by the processor.
chunklength	Number of particles to be handled by each processor.
uneven_parts	Number of processors that will handle an extra particle.
i	Loop variable.

4.2.2.2 initialize_positions()

```
subroutine initialization::initialize_positions (
            integer, intent(in) n,
            real(8), intent(in) rho,
            real(8), dimension(n, 3), intent(out) r )
```

Initializes the positions of N particles in a cubic box with density rho. The box is centered at the origin. The positions are stored in the array r. The length of the box is calculated as L = (N/rho)∗∗(1/3). The box is divided in M = N∗∗(1/3) cells. The length of each cell is a = L/M.

Parameters

N	Number of particles.
rho	Density of the system.
r	Array containing the positions of the particles.
L	Length of the box.
a	Length of each cell.
x,y,z	Coordinates of the particle.
ini	Initial position of the box.
M	Number of cells in each direction.
i,j,k	Loop variables.
particle	Index of the particle.

4.2.2.3 initialize_velocities()

```
subroutine initialization::initialize_velocities (
            integer, intent(in) n,
            real(8) v_ini,
            real(8), dimension(n, 3), intent(out) v )
```

Initializes the velocities of N particles with a given velocity v_ini. The velocities are stored in the array v. The velocities are chosen randomly from a bimodal distribution.

Parameters

N	Number of particles.
v_ini	Initial velocity of the particles.

Parameters

v	Array containing the velocities of the particles.
v_i	Velocity of the particle.
rand	Random number.
particle	Index of the particle.
i	Loop variable.

4.3 integrators Module Reference

Module containing various integrators for molecular dynamics simulations.

Functions/Subroutines

- subroutine, public time_step_vverlet (r, vel, pot, n, l, cutoff, dt, ppot, nprocs, rank, counts_recv, displs_recv, imin, imax)

  Perform a time step using the velocity Verlet integration method. Parallel implementation. Calculates new positions and velocities for particles based on forces and previous positions/velocities.
- subroutine, public bm (ndat, xnums, sigma)

  Generate random numbers following a Box-Muller transformation. Generates normally distributed random numbers using the Box-Muller transformation.
- subroutine, public kinetic_energy (vel, k_energy, n)
- real(8) function, public inst_temp (n, k_energy)

  Calculate the instantaneous temperature of the system.
- subroutine, public momentum (vel, p, n)
- subroutine, public **therm_andersen** (vel, nu, sigma_gaussian, n, xnums)
- subroutine time_step_vverlet (r, vel, pot, n, l, cutoff, dt, ppot)

  Perform a time step using the velocity Verlet integration method. Calculates new positions and velocities for particles based on forces and previous positions/velocities.
- subroutine time_step_euler_pbc (r_in, r_out, vel, n, l, cutoff, dt, pot)

  Perform a time step using the Euler method with periodic boundary conditions. Calculates new positions and velocities for particles based on forces and previous positions/velocities.
- subroutine time_step_verlet (r, rold, vel, n, l, cutoff, dt, pot)

  Perform a time step using the Verlet integration method. Calculates new positions and velocities for particles based on forces and previous positions/velocities.
- subroutine **therm_andersen** (vel, nu, sigma_gaussian, n)

4.3.1 Detailed Description

Module containing various integrators for molecular dynamics simulations.

4.3.2 Function/Subroutine Documentation

4.3.2.1 bm()

```
subroutine integrators::bm (
            integer, intent(in) ndat,
            real(8), dimension(ndat), intent(out) xnums,
            real(8), intent(in) sigma )
```

Generate random numbers following a Box-Muller transformation. Generates normally distributed random numbers using the Box-Muller transformation.

Parameters

	ndat	Number of data points (must be even).
	xnums	Output array containing the generated random numbers.
	sigma	Standard deviation of the normal distribution.
in	*ndat*	Number of data points (must be even)
out	*xnums*	Output array containing generated random numbers
in	*sigma*	Standard deviation of the normal distribution

4.3.2.2 inst_temp()

```
real(8) function, public integrators::inst_temp (
            integer, intent(in) n,
            real(8), intent(in) k_energy )
```

Calculate the instantaneous temperature of the system.

Parameters

N	Number of particles.
K_energy	Kinetic energy of the particles.

Returns

Instantaneous temperature of the system.

Parameters

in	*n*	Number of particles
in	*k_energy*	Kinetic energy of the particles

Returns

Instantaneous temperature of the system

4.3.2.3 kinetic_energy()

```
subroutine integrators::kinetic_energy (
            real(8), dimension(n, 3), intent(in) vel,
            real(8), intent(out) k_energy,
            integer, intent(in) n )
```

Parameters

in	*n*	Number of particles
in	*vel*	Array containing particle velocities
out	*k_energy*	Output variable for the kinetic energy

4.3.2.4 momentum()

```
subroutine integrators::momentum (
            real(8), dimension(n, 3), intent(in) vel,
            real(8), intent(out) p,
            integer, intent(in) n )
```

Parameters

in	*n*	Number of particles
in	*vel*	Array containing particle velocities
out	*p*	Output variable for the total momentum

4.3.2.5 time_step_euler_pbc()

```
subroutine integrators::time_step_euler_pbc (
            real(8), dimension(n, 3), intent(in) r_in,
            real(8), dimension(n, 3), intent(out) r_out,
            real(8), dimension(n, 3) vel,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), intent(in) dt,
            real(8) pot )
```

Perform a time step using the Euler method with periodic boundary conditions. Calculates new positions and velocities for particles based on forces and previous positions/velocities.

Parameters

	r_in	Input array containing initial particle positions.
	r_out	Output array containing updated particle positions.
	vel	Input/Output array containing particle velocities.
	N	Number of particles.
	L	Box size.
	cutoff	Cutoff distance for LJ potential.
	dt	Time step size.
	pot	Output potential energy.
in	*n*	Number of particles
in	*r_in*	Initial particle positions
out	*r_out*	Updated particle positions
	vel	Particle velocities
in	*cutoff*	Box size, time step size, cutoff distance

4.3.2.6 time_step_verlet()

```
subroutine integrators::time_step_verlet (
            real(8), dimension(n, 3), intent(inout) r,
            real(8), dimension(n, 3), intent(inout) rold,
            real(8), dimension(n, 3), intent(inout) vel,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), intent(in) dt,
            real(8) pot )
```

Perform a time step using the Verlet integration method. Calculates new positions and velocities for particles based on forces and previous positions/velocities.

Parameters

	r	Input/Output array containing current particle positions.
	rold	Input/Output array containing previous particle positions.
	vel	Input/Output array containing particle velocities.
	N	Number of particles.
	L	Box size.
	cutoff	Cutoff distance for LJ potential.
	dt	Time step size.
	pot	Output potential energy.
`in`	*n*	Number of particles
`in,out`	*r*	Current particle positions
`in,out`	*rold*	Previous particle positions
`in,out`	*vel*	Particle velocities
`in`	*l*	Time step size, cutoff distance, box size

4.3.2.7 time_step_vverlet() [1/2]

```
subroutine integrators::time_step_vverlet (
            real(8), dimension(n, 3), intent(inout) r,
            real(8), dimension(n, 3), intent(inout) vel,
            real(8), intent(out) pot,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), intent(in) dt,
            real(8), intent(out) ppot )
```

Perform a time step using the velocity Verlet integration method. Calculates new positions and velocities for particles based on forces and previous positions/velocities.

Parameters

	r	Input/Output array containing particle positions.
	vel	Input/Output array containing particle velocities.
	pot	Output potential energy.
	N	Number of particles.
	L	Box size.
	cutoff	Cutoff distance for LJ potential.
	dt	Time step size.
`in`	*n*	Number of particles

Parameters

in,out	*r*	Particle positions
in,out	*vel*	Particle velocities
out	*ppot*	Potential energy
in	*cutoff*	Time step size, box size, cutoff distance

4.3.2.8 time_step_vverlet() [2/2]

```
subroutine, public integrators::time_step_vverlet (
            real(8), dimension(n, 3), intent(inout) r,
            real(8), dimension(n, 3), intent(inout) vel,
            real(8), intent(out) pot,
            integer, intent(in) n,
            real(8), intent(in) l,
            real(8), intent(in) cutoff,
            real(8), intent(in) dt,
            real(8), intent(out) ppot,
            integer nprocs,
            integer rank,
            integer, dimension(0:nprocs-1) counts_recv,
            integer, dimension(0:nprocs-1) displs_recv,
            integer imin,
            integer imax )
```

Perform a time step using the velocity Verlet integration method. Parallel implementation. Calculates new positions and velocities for particles based on forces and previous positions/velocities.

Parameters

	r	Input/Output array containing particle positions.
	vel	Input/Output array containing particle velocities.
	pot	Output potential energy.
	N	Number of particles.
	L	Box size.
	cutoff	Cutoff distance for LJ potential.
	dt	Time step size.
in	*n*	Number of particles
in,out	*r*	Particle positions
in,out	*vel*	Particle velocities
out	*ppot*	Potential energy
in	*cutoff*	Time step size, box size, cutoff distance
	displs_recv	Arrays for MPI_ALLGATHERV
	imax	Defines the range of particles for each MPI process

4.4 pbc_module Module Reference

Module containing subroutines to apply periodic boundary conditions to a N-dimensional system.

Functions/Subroutines

- subroutine pbc_mic (vector, l, d)

  Subroutine to apply periodic boundary conditions to a N-dimensional system, minimum image convention.
- subroutine real(8), dimension(d), intent(inout) pbc (vector, l, d)

  Subroutine to apply periodic boundary conditions to a N-dimensional system.

4.4.1 Detailed Description

Module containing subroutines to apply periodic boundary conditions to a N-dimensional system.

4.4.2 Function/Subroutine Documentation

4.4.2.1 pbc()

```
subroutine pbc_module::pbc (
             real(8), dimension(d), intent(inout) vector,
             real(8), intent(in) l,
             integer, intent(in) d )
```

Subroutine to apply periodic boundary conditions to a N-dimensional system.

Parameters

vector	Vector to apply PBC
L	Box size
D	Dimension of the system

4.4.2.2 pbc_mic()

```
subroutine pbc_module::pbc_mic (
             real(8), dimension(d), intent(inout) vector,
             real(8), intent(in) l,
             integer, intent(in) d )
```

Subroutine to apply periodic boundary conditions to a N-dimensional system, minimum image convention.

Parameters

vector	Vector to apply PBC
L	Box size
D	Dimension of the system

Chapter 5

File Documentation

5.1 parallel/src/main.f90 File Reference

Program to simulate a Lennard-Jones fluid using MPI.

Functions/Subroutines

- program **main**

5.1.1 Detailed Description

Program to simulate a Lennard-Jones fluid using MPI.

Author

A. Monclús, A. Plazas, M. Serrano, A. Gaya

Date

April 2024s

Program to simulate a Lennard-Jones fluid using MPI

The program simulates a Lennard-Jones fluid using the velocity Verlet algorithm. The program is parallelized using MPI. The program reads the parameters from a namelist file called namMD.nml. The program writes the initial and final positions and velocities to files called pos_ini.dat, vel_ini.dat, pos_out.dat and vel_fin.dat. The program writes the energy, temperature and pressure to files called energy_verlet.dat, Temperatures_verlet.dat and pressure_↩ verlet.dat. The program writes the distribution of positions and velocities for the last 10% of the simulation to files called pos_out.dat and vel_fin.dat. The program writes the time elapsed to the standard output.

5.2 serial/src/main.f90 File Reference

Main program for the molecular dynamics simulation.

Functions/Subroutines

- program **main**

5.2.1 Detailed Description

Main program for the molecular dynamics simulation.

Author

A. Monclús, A. Plazas, M. Serrano, A. Gaya

Date

April 2024

This program runs a molecular dynamics simulation using the velocity Verlet algorithm and the Andersen thermostat. The system is composed of N particles with Lennard-Jones interactions. The program reads the parameters from the file namMD.nml and writes the initial and final positions and velocities to files pos_ini.dat, pos_out.dat, vel↩ _ini.dat and vel_fin.dat. The program also writes the energy, temperature and pressure of the system to files energy_verlet.dat, Temperatures_verlet.dat and pressure_verlet.dat. The program also writes the velocities of the particles for the last 10% of the simulation to file vel_fin_verlet.dat.

Index