Molecular Dynamics of Liquid Argon

ICTP - CMP Diploma 2019-2020

Numerical methods II - Final Project

Jesus Andres Espinoza-Valverde

Folder Organization

Our package contains the following directories:

- **sources**: This Folder contains the source code.
- **input**: Here we store the input files.
- **results**: Within this folder we store the raw output of the simulation.
- **executable**: This folder is meant to contain our executable binary.
- **scripts**: Here we have helper bash scripts.
- **data_analysis**: This folder contains a python script that we used to analyze the simulation data.
- **documents**: Here we find references and important documents.

Source Code Files

The source code is composed by the following files:

- **md.f90**: The main program, it contains the main molecular dynamics loop.
- **physics.f90**: This file contains physics-related modules and routines.
- **io.f90**: This file contains input/output and file management modules and routines.
- **utils.f90**: Here we find miscellaneous modules and routines.

Simulation Modules

MODULE Physical Constants (physconst)

Description: This module holds the needed physical constants used in the simulation.

MODULE Molecular Dynamics System (mdsys)

Description: This module holds the complete system information.

MODULE High precision kinds

Description: This module contains redefinitions for precision of floating point number and length file size names.

MODULE Utilities (utils)

Description: This module contains helper and miscellaneous routines.

Routines:

- **pbc routine (pbc)**: applies minimum image convention.
- **Box-Muller Method (box_muller_method)**: Generates Gaussian random numbers.

MODULE Physics

Description: This module contains all physically-related routines.

Routines:

- **Get kinetic energy routine (getekin)**: Computes total kinetic energy of the system.
- **Get temperature routine (gettemp)**: Computes the temperature of the system.
- **Force routine (force)**: Computes the temperature of the system.
- **Velocity Verlet routine (velverlet)**: Updates velocities and p via Verlet Algorithm.
- **Poors-man Thermostat routine (thermostat)**: Rescales temperature of the system.
- **Maxwell Bolzmann velocity initializator (MaxBoltz_Dist_vel_init)**: Initialize velocities according to MB velocity distribution at a given temperature.
- **FCC lattice positions iniatializator (fcc_lattice_positions_init):** Initialize positions in a FCC lattice.
- **Force to Zero routine (force_to_zero)**: Resets Force values.
- **Get distances routine (get_distances):** Computes distances between the particles in the system.

MODULE Input/Output

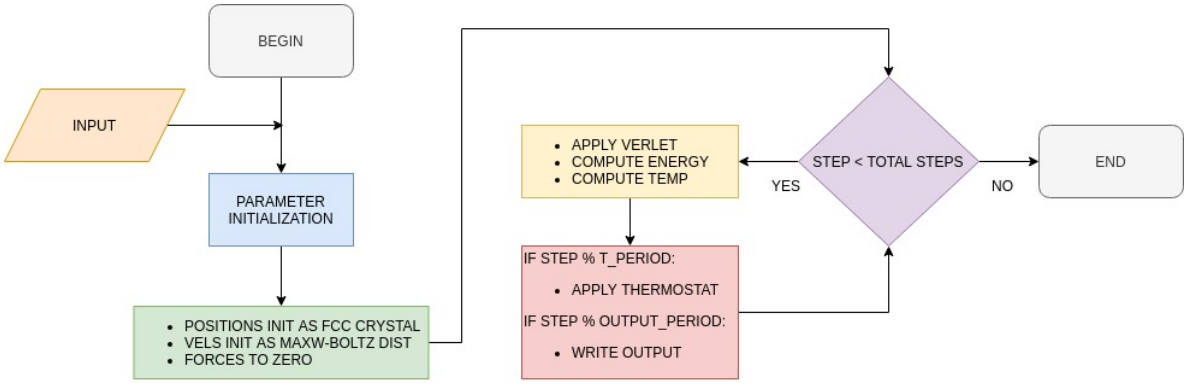
Description: This module is in charge of the reading and writing files with simulation data.

Routines:

- **ioopen routine (ioopen):** Opens all the needed files.
- **ioclose routine (ioclose)**: Closes all the needed files.
- **output routine (output):** Writes simulation data into files.
- **output temps routine (output_temps):** writes temperature data only.

Logic Flow of the code

The logic flow of the implementation is represented graphically in the following flow chart:



There are two special remarks regarding the implementation, they are the following:

Fcc Positions Initialization

For a considerably high density of particles a uniform random initialization is prone to induce infinite (NaN) potential energies. Cubic lattice initialization was used to go to even higher densities but it broke down beyond 343 particles. To reach 864 particles we have decided to initialize the atomic positions by putting them on a Fcc lattice. The following code snippet shows the procedure, where (i,j,k) represents a lattice point and each of the four triplets correspond to one of the four basis vectors:

```
        DO i=1, cbrt_cells
            DO j=1, cbrt_cells
                DO k=1, cbrt_cells

                    ! 1st basis vector (0, 0, 0)
                    rx(cell_idx + 0) = (i+0.0_dbl)*lattice_spacing
                    ry(cell_idx + 0) = (j+0.0_dbl)*lattice_spacing
                    rz(cell_idx + 0) = (k+0.0_dbl)*lattice_spacing

                    ! 2nd basis vector (1/2, 1/2, 0)
                    rx(cell_idx + 1) = (i+0.5_dbl)*lattice_spacing
                    ry(cell_idx + 1) = (j+0.5_dbl)*lattice_spacing
                    rz(cell_idx + 1) = (k+0.0_dbl)*lattice_spacing

                    ! 3rd basis vector (1/2, 0, 1/2)
                    rx(cell_idx + 2) = (i+0.5_dbl)*lattice_spacing
                    ry(cell_idx + 2) = (j+0.0_dbl)*lattice_spacing
                    rz(cell_idx + 2) = (k+0.5_dbl)*lattice_spacing

                    ! 4th basis vector (0, 1/2, 1/2)
                    rx(cell_idx + 3) = (i+0.0_dbl)*lattice_spacing
                    ry(cell_idx + 3) = (j+0.5_dbl)*lattice_spacing
                    rz(cell_idx + 3) = (k+0.5_dbl)*lattice_spacing

                    cell_idx = cell_idx + 4
                END DO
            END DO
        END DO
```

Poors-man thermostat

Notice the temperature of the system can be computed directly from:

$$T = \frac{M}{3Nk_B}\sum_{i=1}^{N}\mathbf{v}_i^2$$

This means we can keep the temperature approximately constant, let say at T_0, by just rescaling all the velocities by a factor of $\sqrt{T_0/T}$:

$$\mathbf{v}_i \leftarrow \sqrt{\frac{T_0}{T}}\mathbf{v}_i$$

We do this every every **N_thermstat** steps.

How to use

First we have to set the simulation parameters. To do so, open the file *input.in* located in the folder **input**. The default options are the following:

```
864                     # natoms
39.948                  # mass in AMU
0.2379                  # epsilon in kcal/mol
3.405                   # sigma in angstrom
7.65                    # rcut in angstrom
34.7786                 # box length (in angstrom)
10000                   # nr MD steps
5.0                     # MD time step (in fs)
100                     # output print frequency
94.4                    # reservoir temperature
10                      # thermo update rate
```

Once this is ready we can compile the code. In order to do that, run:

```
$ make
```

Now that the code compiled the next step is to run it. To run:

```
$ make run
```

Once the simulation is completed, we can proceed with the data analysis. To do do, run:

```
$ make analyze
```

Plots will pop out. This is everything. Now to clean out all the binaries, run:

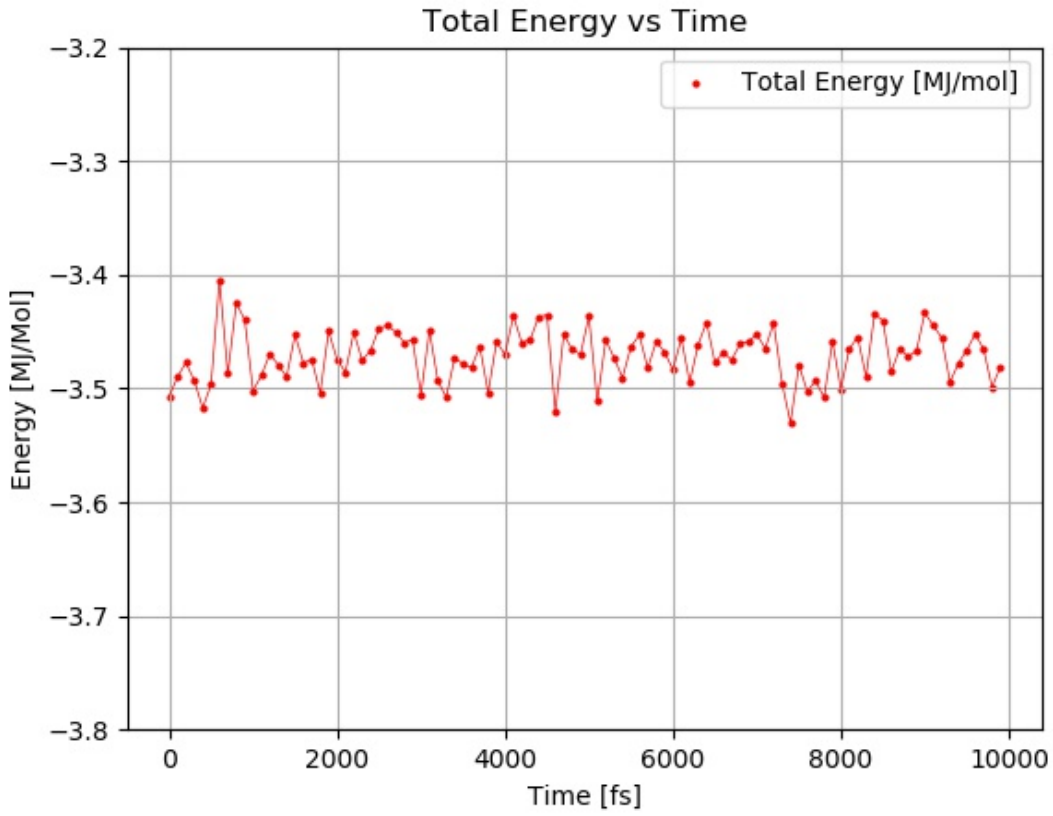
```
$ make clean
```

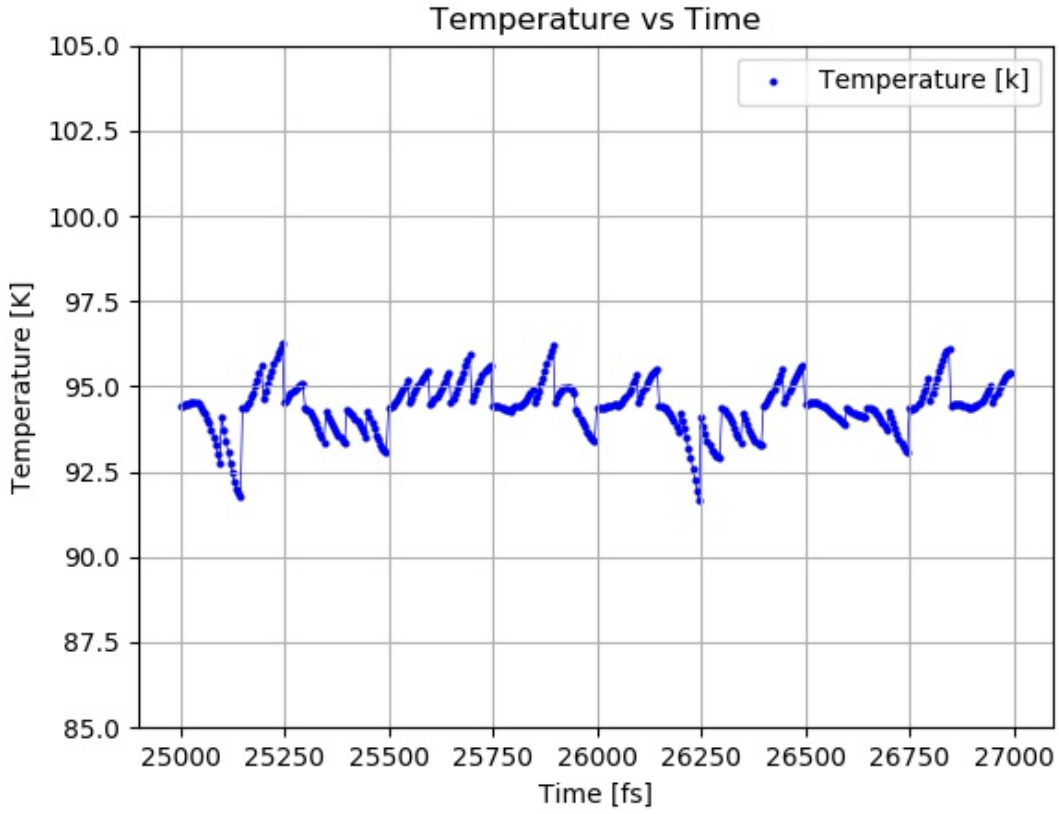
To remove everything, including output files and images, run:

```
$ make flush
```

Results

Energy vs time and Temperature vs time



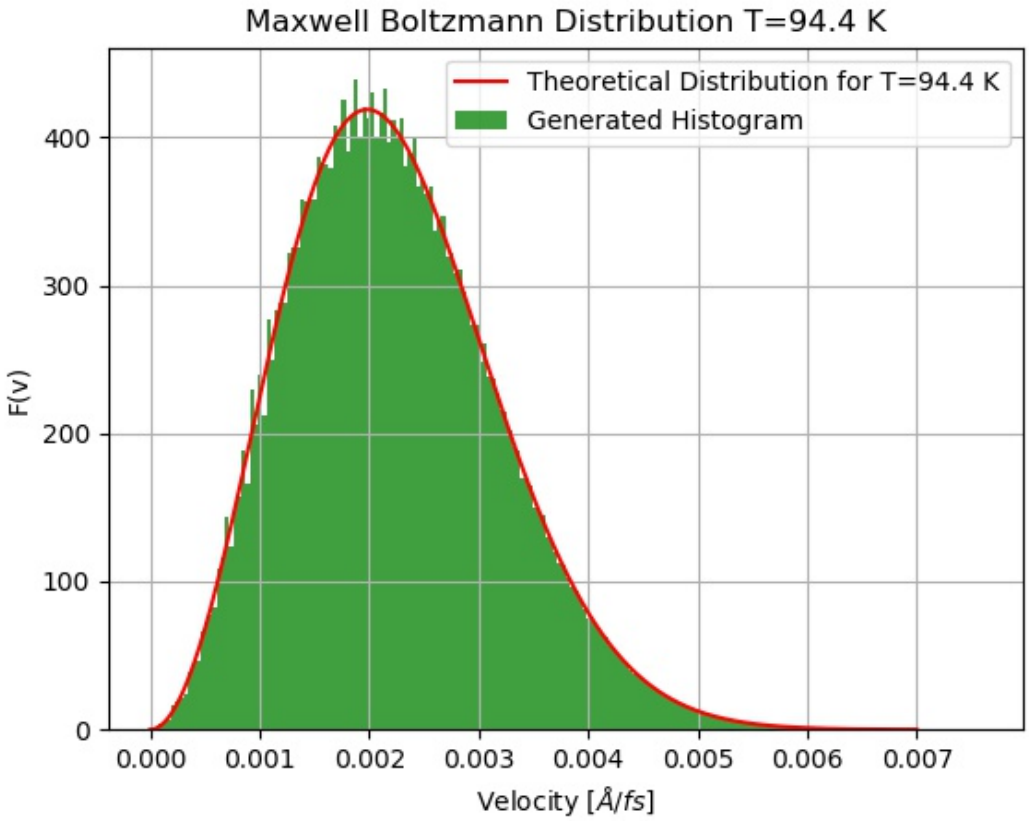


Notice that the periodic jumps in the temperature vs time plot. The frequency of the jumps (50 fs) corresponds with the frequency in which our thermostat pumps kinetic energy into the system.

Distribution on velocities

A histograms of velocities is prepared. The red curve in the following plot corresponds to the theoretical result for a temperature $T = 94.4\,K$. It is computed as:

$$f(v) = 4\pi\left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} v^2 e^{\frac{-mv^2}{2k_B T}}$$



Pair Correlation Function

The pair correlation function or radial distribution function is given by:

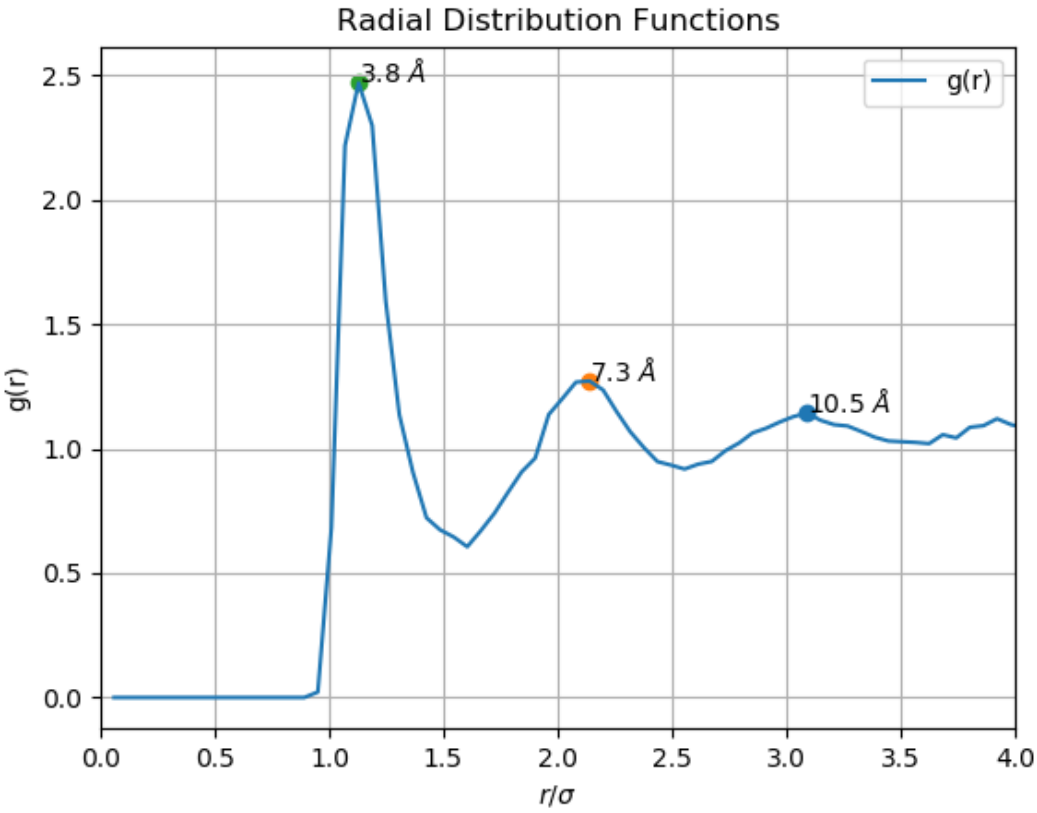
$$g(r) = \frac{1}{\rho}\frac{\delta n(r, r+\delta r)}{4\pi r^2 \delta r}$$

where:

$\delta n(r, r+\delta r)$: number of particles in the shell of inner radius r and outer radius $r+\delta r$.

$4\pi r^2 \delta r$: Shell volume.

ρ: total particle density



This result matches very nicely the Rahman results.

Animation

A cool video of this simulation using VMD software can be found here:

https://youtu.be/M-hA6Vg5Mno