Documentation and User Guide

Molecular Dynamics Program for Argon

Pauline Schütt & Albert Makhmudov

Version 1.0

Updated on: 10/01/2025

1 The program

This project contains a program for running molecular dynamics (MD) simulations for Argon. The program reads atomic coordinates and masses from an input file and performs a MD simulation using the Verlet algorithm and the Lennard-Jones potential. The trajectory in XYZ format, as well as an extended trajectory file including velocities, the accelerations and energies for each step are generated as output.The user can specify whether a velocity-rescale thermostat should be used.

2 Installation

2.1 Prerequisites

Ensure you have the following installed on your system.

```
gcc
make
```

2.2 Installation

1. Clone the repository:

```
git clone https://github.com/pauline-schtt/tccm-homeworks/tree/master/project3
cd project3
```

2. Compile the program using `make`:

```
make
```

3. Then, run the program:

```
./MD <path_to_the_input_file>
```

Installation instructions can also be found in the `INSTALL.md` file.

3 Usage

To run the molecular dynamics simulation, provide the full path to the input file containing the atomic coordinates and masses as an argument to the program. An example input file can be found in `data/inp.txt`.

Example:

```
./MD data/inp.txt
```

The number of steps in the simulation and the time step can be adjusted from the command line using the options `-n` and `-t`.

Example:

```
./MD data/inp.txt -n 2000 -t 0.1
```

If the velocity-rescale thermostat should be activated, specify `-v` when running the program followed by the desired temperature.

Example:

```
./MD data/inp.txt -v 10
```

4 Input format

Atomic coordinates are provided in nm in the following format:

```
<number of atoms>
<x_1>   <y_1>   <z_1>   <mass_1>
<x_2>   <y_2>   <z_2>   <mass_2>
...
<x_n>   <y_n>   <z_n>   <mass_n>
```

Example:

```
3
0.0   0.0    0.0   39.948
0.0   0.0    0.5   39.948
0.1   0.2   -0.5   39.948
```

An example input file can be found in the `data` folder.

5 Output format

The program produces the following output files:

- **trajectory.xyz**: atomic coordinates for each step of the simulation in XYZ format, kinetic, potential and total energy are provided on the comment line
- **trajectory_velocities.xyz**: similar to trajectory.xyz, but additionally contains the velocities for each atom
- **energies**: kinetic, potential and total energy for each step of the simulation
- **accelerations**: acceleration in x, y and z direction for each atom for each step of the simulation

The trajectory in XYZ format can be opened and visualized, e.g. with the VMD or JMOL software.

Example output files can be found in the `tests` folder.

6 Command line options and default values

The program uses the following default values:

- number of steps: 1000
- time step: 0.2

- mass of Argon: 39.948 g/mol
- ϵ: 0.661 j/mol
- σ: 0.3345 nm

The number of steps in the MD simulation can be adjusted by specifying `-n <number of steps>` when running the program.

Similarly, the option `-t <time step>` can be used to adjust the value of the time step in the simulation.

6.1 Thermostat

If the thermostat is deactivated (default), velocities are initialized to zero and the conversation of the total energy is checked at each step. The user is warned if the total energy varies by more than 10 % in one of the steps.

The thermostat can be activated by using the `-v` option when running the program followed by a temperature. This will lead to initialization of the velocities equal to the mean thermal velocity and with a random direction. After each velocity update, the actual temperature T is calculated from the kinetic energy K and the number of atoms N using Eq. (1), where R is the ideal gas constant.

$$T = \frac{2 \cdot N \cdot R}{K} \tag{1}$$

Then, all velocities are scaled by a factor λ as calculated in Eq. (2).

$$\lambda = \sqrt{\frac{T}{T_{\text{desired}}}} \tag{2}$$

7 Tests

The program was tested on MacOS Sequoia 15.2 and Ubuntu 24.04. You can find an example input file in the `data` folder, while the corresponding output files can be found in the `test` folder. As no random velocity initialization is performed, you should obtain qualitatively similar results when running the program with the test input.

8 Cleaning up

To clean up the compiled files, run:

```
make clean
```

9 Troubleshooting

If you encounter any issues during the installation, ensure that your versions of the prerequisites meet the required versions mentioned above. In case you're still facing issues, feel free to reach out to any of the contributors or opening an issue on GitHub.

10 License

11 Acknowledgments

This project is based on data and instructions provided by Abdallah Ammar, Yann Damour, Peter Reinhardt and Anthony Scemama, available at https://github.com/scemama/tccm-homeworks.

Appendix

Detailed documentation on the file contents, functions and variables in the code are provided in the documentation as generated using Doxygen (https://www.doxygen.nl/) in the appendix.

Molecular Dynamics Code for Noble Gases

1.0

Generated by Doxygen 1.9.8

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

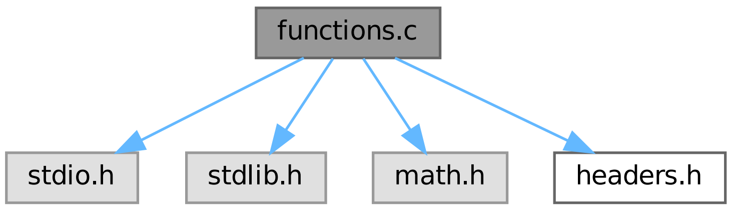
File Documentation

2.1 functions.c File Reference

Contains the functions associated with the molecular dynamics simulation.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "headers.h"
```

Include dependency graph for functions.c:



Functions

- double ∗∗ allocate_2d_array (int rows, int cols)

  Allocates a 2D array of doubles.
- void free_2d_array (double ∗∗array, int rows)

  Frees a 2D array from memory.
- int read_natoms (const char ∗filename)

  Reads the number of atoms from an input file.
- void read_coords_and_masses (const char ∗filename, double ∗∗coords, double ∗masses, int n_atoms)

  Reads atomic coordinates and masses from an input file.
- int validate_atoms (double ∗masses, double ∗epsilon, double ∗sigma, int n_atoms)

  Validates that all atoms in the system are argon.

- FILE * open_output (const char *filename)

  Opens a file for writing output.
- void print_output (FILE *trajectory_file, FILE *energy_file, FILE *extended_file, FILE *acceleration_file, int n_atoms, int step, double kinetic_energy, double potential_energy, double total_energy, double **coords, double **velocities, double **accelerations)

  Prints simulation output.
- void initialize_velocities (double **velocities, double *masses, double temperature, int n_atoms)

  Initialize random velocities.
- void calculate_distances (double **coords, double **distances, int n_atoms)

  Calculates distances between all pairs of atoms.
- static double lennard_jones_potential (double r, double epsilon, double sigma)

  Calculates the Lennard-Jones potential between two atoms.
- double calculate_potential_energy (double **distances, int n_atoms, double epsilon, double sigma)

  Calculates total potential energy of the system.
- double calculate_kinetic_energy (double **velocities, double *masses, int n_atoms)

  Calculates total kinetic energy of the system.
- double calculate_total_energy (double kinetic_energy, double potential_energy)

  Calculates total energy of the system.
- void check_energy (double previous_energy, double total_energy, int step)

  Checks energy conservation between simulation steps.
- void thermostat (double kinetic_energy, double temperature, double **velocities, int n_atoms)

  Velocity-rescaling thermostat.
- static double calculate_U (double r, double epsilon, double sigma)

  Helper function for acceleration calculation.
- void calculate_accelerations (double **coords, double *masses, int n_atoms, double epsilon, double sigma, double **distances, double **accelerations)

  Calculates acceleration vectors for all atoms.
- void update_positions (double **coords, double **velocities, double **accelerations, double dt, int n_atoms)

  Updates atomic positions using Verlet algorithm.
- void update_velocities (double **velocities, double **accelerations, double dt, int n_atoms)

  Updates atomic velocities using Verlet algorithm.

2.1.1 Detailed Description

Contains the functions associated with the molecular dynamics simulation.

2.1.2 Function Documentation

2.1.2.1 allocate_2d_array()

```
double ** allocate_2d_array (
              int rows,
              int cols )
```

Allocates a 2D array of doubles.

Parameters

rows	Number of rows in the array
cols	Number of columns in the array

Returns

Pointer to the allocated 2D array

Exceptions

Exits	with code 1 if memory allocation fails

2.1.2.2 calculate_accelerations()

```
void calculate_accelerations (
             double ** coords,
             double * masses,
             int n_atoms,
             double epsilon,
             double sigma,
             double ** distances,
             double ** accelerations )
```

Calculates acceleration vectors for all atoms.

Parameters

coords	Array of atomic coordinates
masses	Array of atomic masses
n_atoms	Number of atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential
distances	2D array of pairwise distances
accelerations	Array to store calculated accelerations

2.1.2.3 calculate_distances()

```
void calculate_distances (
             double ** coords,
             double ** distances,
             int n_atoms )
```

Calculates distances between all pairs of atoms.

Parameters

coords	Array of atomic coordinates
distances	2D array to store pairwise distances
n_atoms	Number of atoms

2.1.2.4 calculate_kinetic_energy()

```
double calculate_kinetic_energy (
             double ** velocities,
             double * masses,
             int n_atoms )
```

Calculates total kinetic energy of the system.

Parameters

velocities	Array of atomic velocities
masses	Array of atomic masses
n_atoms	Number of atoms

Returns

Total kinetic energy of the system

2.1.2.5 calculate_potential_energy()

```
double calculate_potential_energy (
             double ** distances,
             int n_atoms,
             double epsilon,
             double sigma )
```

Calculates total potential energy of the system.

Parameters

distances	2D array of pairwise distances
n_atoms	Number of atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential

Returns

Total potential energy of the system

2.1.2.6 calculate_total_energy()

```
double calculate_total_energy (
             double kinetic_energy,
             double potential_energy )
```

Calculates total energy of the system.

Parameters

kinetic_energy	Current kinetic energy
potential_energy	Current potential energy

Returns

Total energy of the system

2.1.2.7 calculate_U()

```
static double calculate_U (
            double r,
            double epsilon,
            double sigma )  [static]
```

Helper function for acceleration calculation.

Parameters

r	Distance between atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential

Returns

U value

2.1.2.8 check_energy()

```
void check_energy (
            double previous_energy,
            double total_energy,
            int step )
```

Checks energy conservation between simulation steps.

Parameters

previous_energy	Energy from previous step
total_energy	Current total energy
step	Current simulation step

Warning

Prints warning if energy varies by more than 10%

2.1.2.9 free_2d_array()

```
void free_2d_array (
            double ** array,
            int rows )
```

Frees a 2D array from memory.

Parameters

array	Pointer to the 2D array to be freed
rows	Number of rows in the array

2.1.2.10 initialize_velocities()

```
void initialize_velocities (
            double ** velocities,
            double * masses,
            double temperature,
            int n_atoms )
```

Initialize random velocities.

Parameters

velocities	Array of velocities
masses	Array of masses
temperature	Temperature
n_atoms	Number of atoms

2.1.2.11 lennard_jones_potential()

```
static double lennard_jones_potential (
            double r,
            double epsilon,
            double sigma )  [static]
```

Calculates the Lennard-Jones potential between two atoms.

Parameters

r	Distance between atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential

Returns

Value of the Lennard-Jones potential

2.1.2.12 open_output()

```
FILE * open_output (
             const char * filename )
```

Opens a file for writing output.

Parameters

filename	Name of the output file

Returns

Pointer to the opened file

Exceptions

Exits	with code 1 if file cannot be opened

2.1.2.13 print_output()

```
void print_output (
             FILE * trajectory_file,
             FILE * energy_file,
             FILE * extended_file,
             FILE * acceleration_file,
             int n_atoms,
             int step,
             double kinetic_energy,
             double potential_energy,
             double total_energy,
             double ** coords,
             double ** velocities,
             double ** accelerations )
```

Prints simulation output.

Parameters

trajectory_file	File pointer for trajectory output
energy_file	File pointer for energy output
extended_file	File pointer for extended information
acceleration_file	File pointer for acceleration data
n_atoms	Number of atoms
step	Current simulation step
kinetic_energy	Current kinetic energy
potential_energy	Current potential energy
total_energy	Current total energy
coords	Array of atomic coordinates
velocities	Array of atomic velocities
accelerations	Array of atomic accelerations

2.1.2.14 read_coords_and_masses()

```
void read_coords_and_masses (
             const char * filename,
             double ** coords,
             double * masses,
             int n_atoms )
```

Reads atomic coordinates and masses from an input file.

Parameters

filename	Name of the input file
coords	2D array to store coordinates
masses	Array to store atomic masses
n_atoms	Number of atoms

Exceptions

Exits	with code 1 if file cannot be opened

2.1.2.15 read_natoms()

```
int read_natoms (
             const char * filename )
```

Reads the number of atoms from an input file.

Parameters

filename	Name of the input file

Returns

Number of atoms

Exceptions

Exits	with code 1 if file cannot be opened

2.1.2.16 thermostat()

```
void thermostat (
             double kinetic_energy,
             double temperature,
             double ** velocities,
             int n_atoms )
```

Velocity-rescaling thermostat.

Parameters

kinetic_energy	Kinetic energy
temperature	Desired temperature
velocities	Velocities
n_atoms	Number of atoms

2.1.2.17 update_positions()

```
void update_positions (
             double ** coords,
             double ** velocities,
             double ** accelerations,
             double dt,
             int n_atoms )
```

Updates atomic positions using Verlet algorithm.

Parameters

coords	Array of atomic coordinates
velocities	Array of atomic velocities
accelerations	Array of atomic accelerations
dt	Time step
n_atoms	Number of atoms

2.1.2.18 update_velocities()

```
void update_velocities (
             double ** velocities,
             double ** accelerations,
             double dt,
             int n_atoms )
```

Updates atomic velocities using Verlet algorithm.

Parameters

velocities	Array of atomic velocities
accelerations	Array of atomic accelerations
dt	Time step
n_atoms	Number of atoms

2.1.2.19 validate_atoms()

```
int validate_atoms (
             double * masses,
             double * epsilon,
             double * sigma,
             int n_atoms )
```

Validates that all atoms in the system are argon.

Parameters

masses	Array of atomic masses
epsilon	Pointer to store the epsilon parameter
sigma	Pointer to store the sigma parameter
n_atoms	Number of atoms

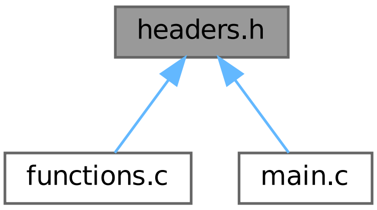
Returns

1 if all atoms are valid, 0 otherwise

2.2 headers.h File Reference

Contains the function headers.

This graph shows which files directly or indirectly include this file:



Macros

- #define CONSTANTS_H
- #define ARGON_MASS 39.948
- #define ARGON_EPSILON 0.0661
- #define ARGON_SIGMA 0.3345
- #define PI 3.14159265358979323846
- #define R 8.31446261815324

Functions

- double ∗∗ allocate_2d_array (int rows, int cols)

  Allocates a 2D array of doubles.
- void free_2d_array (double ∗∗array, int rows)

  Frees a 2D array from memory.
- int read_natoms (const char ∗filename)

  Reads the number of atoms from an input file.
- void read_coords_and_masses (const char ∗filename, double ∗∗coords, double ∗masses, int n_atoms)

  Reads atomic coordinates and masses from an input file.
- int validate_atoms (double ∗masses, double ∗epsilon, double ∗sigma, int n_atoms)

  Validates that all atoms in the system are argon.
- void initialize_velocities (double ∗∗velocities, double ∗masses, double temperature, int n_atoms)

  Initialize random velocities.
- void calculate_distances (double ∗∗coords, double ∗∗distances, int n_atoms)

  Calculates distances between all pairs of atoms.
- double calculate_potential_energy (double ∗∗distances, int n_atoms, double epsilon, double sigma)

  Calculates total potential energy of the system.
- double calculate_kinetic_energy (double ∗∗velocities, double ∗masses, int n_atoms)

  Calculates total kinetic energy of the system.
- double calculate_total_energy (double kinetic_energy, double potential_energy)

  Calculates total energy of the system.
- void thermostat (double kinetic_energy, double temperature, double ∗∗velocities, int n_atoms)

  Velocity-rescaling thermostat.
- void check_energy (double previous_energy, double total_energy, int step)

  Checks energy conservation between simulation steps.
- void calculate_accelerations (double ∗∗coords, double ∗masses, int n_atoms, double epsilon, double sigma, double ∗∗distances, double ∗∗accelerations)

  Calculates acceleration vectors for all atoms.
- void update_positions (double ∗∗coords, double ∗∗velocities, double ∗∗accelerations, double dt, int n_atoms)

  Updates atomic positions using Verlet algorithm.
- void update_velocities (double ∗∗velocities, double ∗∗accelerations, double dt, int n_atoms)

  Updates atomic velocities using Verlet algorithm.
- FILE ∗ open_output (const char ∗filename)

  Opens a file for writing output.
- void print_output (FILE ∗trajectory_file, FILE ∗energy_file, FILE ∗extended_file, FILE ∗acceleration_file, int n_atoms, int step, double kinetic_energy, double potential_energy, double total_energy, double ∗∗coords, double ∗∗velocities, double ∗∗accelerations)

  Prints simulation output.

2.2.1 Detailed Description

Contains the function headers.

2.2.2 Macro Definition Documentation

2.2.2.1 ARGON_EPSILON

```
#define ARGON_EPSILON 0.0661
```

2.2.2.2 ARGON_MASS

```
#define ARGON_MASS 39.948
```

2.2.2.3 ARGON_SIGMA

```
#define ARGON_SIGMA 0.3345
```

2.2.2.4 CONSTANTS_H

```
#define CONSTANTS_H
```

2.2.2.5 PI

```
#define PI 3.14159265358979323846
```

2.2.2.6 R

```
#define R 8.31446261815324
```

2.2.3 Function Documentation

2.2.3.1 allocate_2d_array()

```
double ** allocate_2d_array (
             int rows,
             int cols )
```

Allocates a 2D array of doubles.

Parameters

rows	Number of rows in the array
cols	Number of columns in the array

Returns

Pointer to the allocated 2D array

Exceptions

Exits	with code 1 if memory allocation fails

2.2.3.2 calculate_accelerations()

```
void calculate_accelerations (
             double ** coords,
             double * masses,
             int n_atoms,
             double epsilon,
             double sigma,
             double ** distances,
             double ** accelerations )
```

Calculates acceleration vectors for all atoms.

Parameters

coords	Array of atomic coordinates
masses	Array of atomic masses
n_atoms	Number of atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential
distances	2D array of pairwise distances
accelerations	Array to store calculated accelerations

2.2.3.3 calculate_distances()

```
void calculate_distances (
             double ** coords,
             double ** distances,
             int n_atoms )
```

Calculates distances between all pairs of atoms.

Parameters

coords	Array of atomic coordinates
distances	2D array to store pairwise distances
n_atoms	Number of atoms

2.2.3.4 calculate_kinetic_energy()

```
double calculate_kinetic_energy (
             double ** velocities,
             double * masses,
             int n_atoms )
```

Calculates total kinetic energy of the system.

Parameters

velocities	Array of atomic velocities
masses	Array of atomic masses
n_atoms	Number of atoms

Returns

Total kinetic energy of the system

2.2.3.5 calculate_potential_energy()

```
double calculate_potential_energy (
            double ** distances,
            int n_atoms,
            double epsilon,
            double sigma )
```

Calculates total potential energy of the system.

Parameters

distances	2D array of pairwise distances
n_atoms	Number of atoms
epsilon	Epsilon parameter for LJ potential
sigma	Sigma parameter for LJ potential

Returns

Total potential energy of the system

2.2.3.6 calculate_total_energy()

```
double calculate_total_energy (
            double kinetic_energy,
            double potential_energy )
```

Calculates total energy of the system.

Parameters

kinetic_energy	Current kinetic energy
potential_energy	Current potential energy

Returns

Total energy of the system

2.2.3.7 check_energy()

```
void check_energy (
            double previous_energy,
            double total_energy,
            int step )
```

Checks energy conservation between simulation steps.

Parameters

previous_energy	Energy from previous step
total_energy	Current total energy
step	Current simulation step

Warning

Prints warning if energy varies by more than 10%

2.2.3.8 free_2d_array()

```
void free_2d_array (
             double ** array,
             int rows )
```

Frees a 2D array from memory.

Parameters

array	Pointer to the 2D array to be freed
rows	Number of rows in the array

2.2.3.9 initialize_velocities()

```
void initialize_velocities (
             double ** velocities,
             double * masses,
             double temperature,
             int n_atoms )
```

Initialize random velocities.

Parameters

velocities	Array of velocities
masses	Array of masses
temperature	Temperature
n_atoms	Number of atoms

2.2.3.10 open_output()

```
FILE * open_output (
             const char * filename )
```

Opens a file for writing output.

Parameters

filename	Name of the output file

Returns

Pointer to the opened file

Exceptions

Exits	with code 1 if file cannot be opened

2.2.3.11 print_output()

```
void print_output (
            FILE * trajectory_file,
            FILE * energy_file,
            FILE * extended_file,
            FILE * acceleration_file,
            int n_atoms,
            int step,
            double kinetic_energy,
            double potential_energy,
            double total_energy,
            double ** coords,
            double ** velocities,
            double ** accelerations )
```

Prints simulation output.

Parameters

trajectory_file	File pointer for trajectory output
energy_file	File pointer for energy output
extended_file	File pointer for extended information
acceleration_file	File pointer for acceleration data
n_atoms	Number of atoms
step	Current simulation step
kinetic_energy	Current kinetic energy
potential_energy	Current potential energy
total_energy	Current total energy
coords	Array of atomic coordinates
velocities	Array of atomic velocities
accelerations	Array of atomic accelerations

2.2.3.12 read_coords_and_masses()

```
void read_coords_and_masses (
            const char * filename,
            double ** coords,
            double * masses,
            int n_atoms )
```

Reads atomic coordinates and masses from an input file.

Parameters

filename	Name of the input file
coords	2D array to store coordinates
masses	Array to store atomic masses
n_atoms	Number of atoms

Exceptions

Exits	with code 1 if file cannot be opened

2.2.3.13 read_natoms()

```
int read_natoms (
             const char * filename )
```

Reads the number of atoms from an input file.

Parameters

filename	Name of the input file

Returns

Number of atoms

Exceptions

Exits	with code 1 if file cannot be opened

2.2.3.14 thermostat()

```
void thermostat (
             double kinetic_energy,
             double temperature,
             double ** velocities,
             int n_atoms )
```

Velocity-rescaling thermostat.

Parameters

kinetic_energy	Kinetic energy
temperature	Desired temperature
velocities	Velocities
n_atoms	Number of atoms

2.2.3.15 update_positions()

```
void update_positions (
             double ** coords,
             double ** velocities,
             double ** accelerations,
             double dt,
             int n_atoms )
```

Updates atomic positions using Verlet algorithm.

Parameters

coords	Array of atomic coordinates
velocities	Array of atomic velocities
accelerations	Array of atomic accelerations
dt	Time step
n_atoms	Number of atoms

2.2.3.16 update_velocities()

```
void update_velocities (
             double ** velocities,
             double ** accelerations,
             double dt,
             int n_atoms )
```

Updates atomic velocities using Verlet algorithm.

Parameters

velocities	Array of atomic velocities
accelerations	Array of atomic accelerations
dt	Time step
n_atoms	Number of atoms

2.2.3.17 validate_atoms()

```
int validate_atoms (
             double * masses,
             double * epsilon,
             double * sigma,
             int n_atoms )
```

Validates that all atoms in the system are argon.

Parameters

masses	Array of atomic masses
epsilon	Pointer to store the epsilon parameter
sigma	Pointer to store the sigma parameter
n_atoms	Number of atoms

Returns

1 if all atoms are valid, 0 otherwise

2.3 headers.h

Go to the documentation of this file.

```

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

double** allocate_2d_array(int rows, int cols);
void free_2d_array(double** array, int rows);
int read_natoms(const char* filename);
void read_coords_and_masses(const char* filename, double** coords, double* masses, int n_atoms);
int validate_atoms(double* masses, double* epsilon, double* sigma, int n_atoms);
void initialize_velocities(double** velocities, double* masses, double temperature, int n_atoms);
void calculate_distances(double** coords, double** distances, int n_atoms);
double calculate_potential_energy(double** distances, int n_atoms, double epsilon, double sigma);
double calculate_kinetic_energy(double** velocities, double* masses, int n_atoms);
double calculate_total_energy(double kinetic_energy, double potential_energy);
void thermostat(double kinetic_energy, double temperature, double** velocities, int n_atoms);
void check_energy(double previous_energy, double total_energy, int step);
void calculate_accelerations(double** coords, double* masses, int n_atoms, double epsilon, double sigma, double** distances, double** accelerations);
void update_positions(double** coords, double** velocities, double** accelerations, double dt, int n_atoms);
void update_velocities(double** velocities, double** accelerations, double dt, int n_atoms);
FILE* open_output(const char* filename);
void print_output(FILE* trajectory_file, FILE* energy_file, FILE* extended_file, FILE* acceleration_file, int n_atoms, int step, double kinetic_energy, double potential_energy, double total_energy, double** coords, double** velocities, double** accelerations);
#endif

// Constants
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define ARGON_MASS 39.948
#define ARGON_EPSILON 0.0661 // j/mol
#define ARGON_SIGMA 0.3345 // nm
#define PI 3.14159265358979323846
#define R 8.31446261815324 // Ideal gas constant in J/K/mol

#endif
```

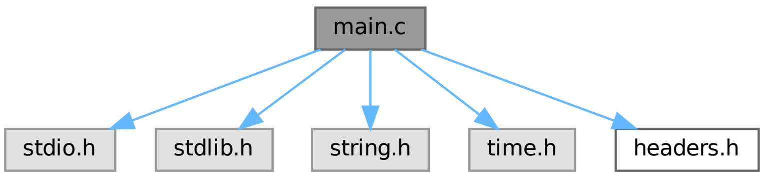
2.4 main.c File Reference

Contains the main program.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <time.h>
#include "headers.h"
```

Include dependency graph for main.c:



Functions

- int main (int argc, char ∗argv[ ])

  The main entry point of the program.

2.4.1 Detailed Description

Contains the main program.

2.4.2 Function Documentation

2.4.2.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

The main entry point of the program.

This program manages reading of the input, performing the MD simualtion and generation of output.

Returns

int Returns 0 upon successful execution.

2.4.3 Variables

- `int n_steps`: Number of simulation steps.
- `double dt`: Time step.
- `double temperature`: Temperature.
- `int thermo`: If set to 1, the thermostat is activated, if set to 0 it is deactivated.
- `const char* filename`: Name of the input file.
- `int n_atoms`: Number of atoms.
- `double** coords`: 2D array of coordinates consisting of x, y, z for each atom.
- `double** distances`: 2D array of distances for each pair of atoms.
- `double* masses`: Array of masses of each atom.
- `double epsilon`: Epsilon parameter for the Lennard-Jones potential in j/mol.
- `double sigma`: Sigma parameter for the Lennard-Jones potential in nm.
- `double** velocities`: 2D array of velocities consisting of vx, vy, vz for each atom.
- `double** accelerations`: 2D array of accelerations consisting of ax, ay, az for each atom.
- `const char* trajectory_name`: Name of the file where the trajectory output is written.
- `FILE* trajectory_file`: File where the trajectory output is written.
- `const char* energy_name`: Name of the file where the energies are written.
- `FILE* energy_file`: File where the energies are written.
- `const char* extended_name`: Name of the file where the extended trajectory with velocities is written.
- `FILE* extended_file`: File where the extended trajectory with velocities is written.
- `const char* acceleration_name`: Name of the file where the accelerations are written.
- `FILE* acceleration_file`: File where the accelerations are written.
- `double kinetic_energy`: Variable for storing the kinetic energy.
- `double potential_energy`: Variable for storing the potential energy.
- `double total_energy`: Variable for storing the total energy.
- `double previous_energy`: Variable for storing the total energy of the previous step.Name of the input file

Index