Molecular Dynamics Code for Argon

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

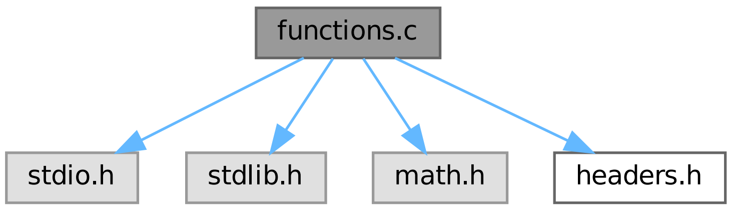
File Documentation

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 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.
- 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 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

2.1.2.11 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

2.1.2.12 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

2.1.2.13 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

Parameters

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

2.1.2.14 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

2.1.2.15 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

2.1.2.16 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

2.1.2.17 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

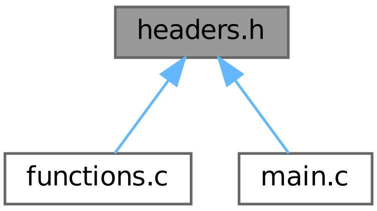
Returns

1 if all atoms are valid, 0 otherwise

2.2 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

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 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 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
```

2.2.3 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 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

2.2.3.10 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

2.2.3.11 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

2.2.3.12 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

2.2.3.13 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

2.2.3.14 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

2.2.3.15 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.

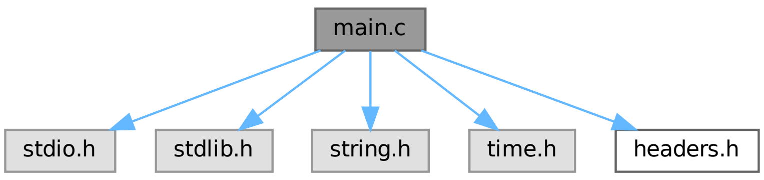
```c

#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 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 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

#endif
```

2.4 main.c File Reference

Contains the main program.

```c
#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

- `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.
- `int n_steps`: Number of simulation steps.
- `double dt`: Time step.
- `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