MDGeneral

Generated by Doxygen 1.8.18

Chapter 1

MDGeneral

This is a repository for making a fast and general molecular dynamics program

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 System Namespace Reference

Classes

- class constants_interaction
- class constants_thermostat
- class input_params
- class simulation
- class system_state

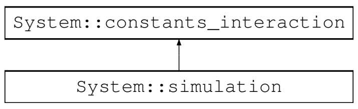
Chapter 7

Class Documentation

7.1 System::constants_interaction Class Reference

```
#include <system.h>
```

Inheritance diagram for System::constants_interaction:



Public Member Functions

- constants_interaction (int n_types)

Public Attributes

- std::vector< std::vector< std::vector< double > > > interaction_const

  Stores the constants of interaction for all the interactions between particles.

7.1.1 Detailed Description

Definition at line 111 of file system.h.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 constants_interaction()

```
System::constants_interaction::constants_interaction (
            int n_types )  [inline]
```

Parameters

n_types	Number of types of particles

Definition at line 131 of file system.h.

7.1.3 Member Data Documentation

7.1.3.1 interaction_const

```
std::vector<std::vector<std::vector<double> > > System::constants_interaction::interaction↩
_const
```

Stores the constants of interaction for all the interactions between particles.

Stores the constants of interaction for all the interactions between particles. For interaction of particles of type1 and type2 the constants are stored in vector interaction_const[type1][type2]
If the interaction is of the Lennard Jones type,
interaction_const[i][j][0] = ϵ
interaction_const[i][j][1] = σ
interaction_const[i][j][2] = Cutoff radius/distance (r_cut)
interaction_const[i][j][3] = Truncated Potential (etrunc)
interaction_const[i][j][4] = σ^6
interaction_const[i][j][5] = Tail Energy (assuming constant distribution outside cutoff radius)

Definition at line 126 of file system.h.

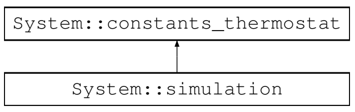
The documentation for this class was generated from the following file:

- Backend/include/system.h

7.2 System::constants_thermostat Class Reference

```
#include <system.h>
```

Inheritance diagram for System::constants_thermostat:



Public Member Functions

- constants_thermostat (int n_types)

Public Attributes

- std::vector< std::vector< double > > thermostat_const

7.2.1 Detailed Description

Definition at line 156 of file system.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 constants_thermostat()

```
System::constants_thermostat::constants_thermostat (
             int n_types )  [inline]
```

Definition at line 163 of file system.h.

7.2.3 Member Data Documentation

7.2.3.1 thermostat_const

```
std::vector<std::vector<double> > System::constants_thermostat::thermostat_const
```

Definition at line 159 of file system.h.

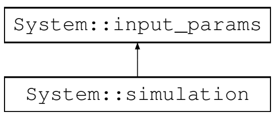
The documentation for this class was generated from the following file:

- Backend/include/system.h

7.3 System::input_params Class Reference

```
#include <system.h>
```

Inheritance diagram for System::input_params:



Public Member Functions

- input_params (std::string input)

    Use periodic boundary conditions if 1. If 0, use rigid walls.

Public Attributes

- int n_types
- int n_dimensions

    This represents the number of types of particles.
- std::vector< int > n_particles

    This represents the number of dimensions of the simulation (by default must be 3)
- double timestep

    This represents the number of particles of each type (n_types sized)
- double runtime

    This defines the size of each timestep (dt)
- int parallelize

    This defines the time for which to run the simulation.
- std::vector< double > mass

    If 1, parallelize. Else, do not parallelize.
- std::vector< double > temperature_required

    This represents the mass of each type of particle (n_types sized)
- int periodic_boundary

    This is the vector of the temperatures required to be mainted for each particle type by the thermostat.

7.3.1 Detailed Description

Definition at line 65 of file system.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 input_params()

```
System::input_params::input_params (
             std::string input )   [inline]
```

Use periodic boundary conditions if 1. If 0, use rigid walls.

Definition at line 78 of file system.h.

7.3.3 Member Data Documentation

7.3.3.1 mass

```
std::vector<double> System::input_params::mass
```

If 1, parallelize. Else, do not parallelize.

Definition at line 74 of file system.h.

7.3.3.2 n_dimensions

```
int System::input_params::n_dimensions
```

This represents the number of types of particles.

Definition at line 69 of file system.h.

7.3.3.3 n_particles

```
std::vector<int> System::input_params::n_particles
```

This represents the number of dimensions of the simulation (by default must be 3)

Definition at line 70 of file system.h.

7.3.3.4 n_types

```
int System::input_params::n_types
```

Definition at line 68 of file system.h.

7.3.3.5 parallelize

```
int System::input_params::parallelize
```

This defines the time for which to run the simulation.

Definition at line 73 of file system.h.

7.3.3.6 periodic_boundary

```
int System::input_params::periodic_boundary
```

This is the vector of the temperatures required to be mainted for each particle type by the thermostat.

Definition at line 76 of file system.h.

7.3.3.7 runtime

```
double System::input_params::runtime
```

This defines the size of each timestep (dt)

Definition at line 72 of file system.h.

7.3.3.8 temperature_required

```
std::vector<double> System::input_params::temperature_required
```

This represents the mass of each type of particle (n_types sized)

Definition at line 75 of file system.h.

7.3.3.9 timestep

```
double System::input_params::timestep
```

This represents the number of particles of each type (n_types sized)

Definition at line 71 of file system.h.

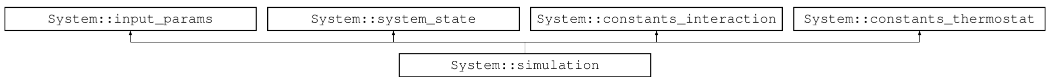
The documentation for this class was generated from the following file:

- Backend/include/system.h

7.4 System::simulation Class Reference

```
#include <system.h>
```

Inheritance diagram for System::simulation:



Public Member Functions

- simulation (std::string input, double size[ ])
- ∼simulation ()

Public Attributes

- std::vector< void(∗)(simulation &, int)> thermostat
- std::vector< std::vector< void(∗)(simulation &, int, int)> > interaction
- std::vector< double > box_size_limits

7.4.1 Detailed Description

Definition at line 183 of file system.h.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 simulation()

```
System::simulation::simulation (
            std::string input,
            double size[ ] )  [inline]
```

Definition at line 190 of file system.h.

7.4.2.2 ∼simulation()

```
System::simulation::∼simulation ( )
```

7.4.3 Member Data Documentation

7.4.3.1 box_size_limits

```
std::vector<double> System::simulation::box_size_limits
```

We assume that the initial limits are all (0,0,0,...,0) to whatever the limits define for a box (allocate to n_dimensions size)

Definition at line 188 of file system.h.

7.4.3.2 interaction

```
std::vector<std::vector<void (*)(simulation&, int, int)> > System::simulation::interaction
```

This defines the set of functions for interaction between different particle types. Also allows for non-symmetric interaction.

Definition at line 187 of file system.h.

7.4.3.3 thermostat

```
std::vector<void (*)(simulation&, int)> System::simulation::thermostat
```

This stores thermostats for different particle sets

Definition at line 186 of file system.h.

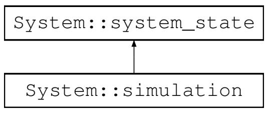
The documentation for this class was generated from the following file:

- Backend/include/system.h

7.5 System::system_state Class Reference

```
#include <system.h>
```

Inheritance diagram for System::system_state:



Public Member Functions

- system_state (int n_types, int n_dimensions, std::vector< int > &n_particles)

Public Attributes

- std::vector< std::vector< std::vector< double > > > position
- std::vector< std::vector< std::vector< double > > > orientation
- std::vector< std::vector< std::vector< double > > > velocity
- std::vector< std::vector< std::vector< double > > > acceleration
- std::vector< double > temperature
- double energy_total
- double energy_potential
- std::vector< double > energy_kinetic
- double time
- int state

7.5.1 Detailed Description

Definition at line 14 of file system.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 system_state()

```
System::system_state::system_state (
             int n_types,
             int n_dimensions,
             std::vector< int > & n_particles )  [inline]
```

Definition at line 27 of file system.h.

7.5.3 Member Data Documentation

7.5.3.1 acceleration

```
std::vector<std::vector<std::vector<double> > > System::system_state::acceleration
```

Vector of n_types X n_particles[of each type] X n_dimensions size storing accelerations of particles

Definition at line 20 of file system.h.

7.5.3.2 energy_kinetic

```
std::vector<double> System::system_state::energy_kinetic
```

Defines the kinetic energy of each particle type

Definition at line 24 of file system.h.

7.5.3.3 energy_potential

```
double System::system_state::energy_potential
```

Defines the total potential energy of interaction at this instant

Definition at line 23 of file system.h.

7.5.3.4 energy_total

```
double System::system_state::energy_total
```

Defines the total energy at this instant

Definition at line 22 of file system.h.

7.5.3.5 orientation

```
std::vector<std::vector<std::vector<double> > > System::system_state::orientation
```

Vector of n_types X n_particles[of each type] X n_dimensions size storing orientation of particles

Definition at line 18 of file system.h.

7.5.3.6 position

```
std::vector<std::vector<std::vector<double> > > System::system_state::position
```

Vector of n_types X n_particles[of each type] X n_dimensions size storing positions of particles

Definition at line 17 of file system.h.

7.5.3.7 state

```
int System::system_state::state
```

Definition at line 26 of file system.h.

7.5.3.8 temperature

```
std::vector<double> System::system_state::temperature
```

This defines the temperatures of the n_types particle sets

Definition at line 21 of file system.h.

7.5.3.9 time

```
double System::system_state::time
```

This is the amount of time passed since the beginning of the simulation

Definition at line 25 of file system.h.

7.5.3.10 velocity

```
std::vector<std::vector<std::vector<double> > > System::system_state::velocity
```

Vector of n_types X n_particles[of each type] X n_dimensions size storing velocity of particles

Definition at line 19 of file system.h.

The documentation for this class was generated from the following file:

- Backend/include/system.h

Chapter 8

File Documentation

8.1 Backend/Goals.txt File Reference

Variables

- So here are some ways we get get this thing off the ground
- So here are some ways we get get this thing off the and its quite similar to RK It also works surprisingly well for a simulations ns Timesteps will be of the order of femtoseconds Create a module to sample I m not quite sure about this yet Primarty Goal
- So here are some ways we get get this thing off the and its quite similar to RK It also works surprisingly well for a simulations ns Timesteps will be of the order of femtoseconds Create a module to sample I m not quite sure about this yet Primarty with periodic boundaries and randomly initiated velocities and no forces The particles will just pass through each other

8.1.1 Variable Documentation

8.1.1.1 Goal

```
So here are some ways we get get this thing off the and its quite similar to RK It also works
surprisingly well for a simulations ns Timesteps will be of the order of femtoseconds Create a
module to sample I m not quite sure about this yet Primarty Goal
```

Definition at line 10 of file Goals.txt.

8.1.1.2 ground

```
So here are some ways we get get this thing off the ground
```

Definition at line 1 of file Goals.txt.

8.1.1.3 other

```
So here are some ways we get get this thing off the and its quite similar to RK It also works
surprisingly well for a simulations ns Timesteps will be of the order of femtoseconds Create
a module to sample I m not quite sure about this yet Primarty with periodic boundaries and
randomly initiated velocities and no forces The particles will just pass through each other
```

Definition at line 11 of file Goals.txt.

8.2 GUI/Goals.txt File Reference

8.3 Backend/include/algorithm_constants.h File Reference

Macros

- #define CUTOFF_RATIO_LJ 2.5
- #define ANDERSON_NU 0.1

8.3.1 Macro Definition Documentation

8.3.1.1 ANDERSON_NU

```
#define ANDERSON_NU 0.1
```

Definition at line 10 of file algorithm_constants.h.

8.3.1.2 CUTOFF_RATIO_LJ

```
#define CUTOFF_RATIO_LJ 2.5
```

Definition at line 7 of file algorithm_constants.h.

8.4 Backend/include/client.h File Reference

```
#include "system.h"
#include "initialize.h"
#include "constants.h"
```

8.5 Backend/include/constants.h File Reference

Macros

- #define PLANK_SI 0.000000000000000000000000000000000662607004
- #define PLANK_EV 0.0000000000000041357
- #define N_A 602214076000000000000000
- #define BOLTZ_SI 0.000000000000000000000013806485

8.5.1 Macro Definition Documentation

8.5.1.1 BOLTZ_SI

```
#define BOLTZ_SI 0.000000000000000000000013806485
```

Definition at line 20 of file constants.h.

8.5.1.2 N_A

```
#define N_A 602214076000000000000000
```

Definition at line 16 of file constants.h.

8.5.1.3 PLANK_EV

```
#define PLANK_EV 0.0000000000000041357
```

Definition at line 12 of file constants.h.

8.5.1.4 PLANK_SI

```
#define PLANK_SI 0.000000000000000000000000000000000662607004
```

Definition at line 8 of file constants.h.

8.6 Backend/include/correlations.h File Reference

8.7 Backend/include/initialize.h File Reference

```
#include <cmath>
#include <random>
#include "system.h"
#include "constants.h"
```

Functions

- void init_sim (System::simulation &sim)

8.7.1 Function Documentation

8.7.1.1 init_sim()

```
void init_sim (
            System::simulation & sim )
```

Definition at line 5 of file initialize.cpp.

8.8 Backend/include/integrate.h File Reference

```
#include <cmath>
#include <algorithm>
#include "system.h"
#include "constants.h"
#include "interaction.h"
#include "thermostat.h"
```

Functions

- void integrate_verdet_periodic (System::simulation &sim)
- void integrate_verdet_box (System::simulation &sim)

8.8.1 Function Documentation

8.8.1.1 integrate_verdet_box()

```
void integrate_verdet_box (
            System::simulation & sim )
```

This function integrates the equation of motion for rigid box conditions This function integrates using the Leapfrog Algorithm

Parameters

sim	Simulation being integrated over

Definition at line 62 of file integrate.cpp.

8.8.1.2 integrate_verdet_periodic()

```
void integrate_verdet_periodic (
            System::simulation & sim )
```

This function integrates the equation of motion for periodic boundary conditions This function integrates using the Leapfrog Algorithm

Parameters

sim	Simulation being integrated over

Definition at line 10 of file integrate.cpp.

8.9 Backend/include/interaction.h File Reference

```
#include "system.h"
```

Functions

- void initialize_interactions (System::simulation &sim)

    Initializes the constant arrays for interactions for speed.
- double distance_periodic (System::simulation &sim, int type1, int n1, int type2, int n2)

    Returns the distance between two particles for periodic boundary conditions.
- void interact (System::simulation &sim)

    This function calls all the required interaction functions between the particles.
- void free_particles (System::simulation &sim, int type1, int type2)

    Setup the free particle interaction between two particle types.
- void lj_periodic (System::simulation &sim, int type1, int type2)

    Setup the Lennard-Jones potential for periodic boundary conditions between two particle types.
- void lj_box (System::simulation &sim, int type1, int type2)

    Setup the Lennard-Jones potential for rigid box boundary conditions between two particle types.

8.9.1 Function Documentation

8.9.1.1 distance_periodic()

```
double distance_periodic (
            System::simulation & sim,
            int type1,
            int n1,
            int type2,
            int n2 )
```

Returns the distance between two particles for periodic boundary conditions.

Returns the distance between two particles labelled number n1,n2 of type1,type2 respectively in the position array assuming periodic boundary conditions

Parameters

sim	Simulation being used
type1	Particle type of particle 1
type2	Particle type of particle 2
n1	Particle index of particle 1 in position[type1] array
n2	Particle index of particle 2 in position[type2] array

Definition at line 67 of file interaction.cpp.

8.9.1.2 free_particles()

```
void free_particles (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the free particle interaction between two particle types.

Setup of free particle interaction between particle types type1 and type2. This does nothing. The function is empty.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 124 of file interaction.cpp.

8.9.1.3 initialize_interactions()

```
void initialize_interactions (
            System::simulation & sim )
```

Initializes the constant arrays for interactions for speed.

The function initializes the arrays for more efficient computation by precomputing the required factors

Parameters

sim	Simulation being initialized

Definition at line 17 of file interaction.cpp.

8.9.1.4 interact()

```
void interact (
            System::simulation & sim )
```

This function calls all the required interaction functions between the particles.

Calls all the interactiosn and updates acceleration and energy arrays

Parameters

sim	Simulation being used

Definition at line 87 of file interaction.cpp.

8.9.1.5 lj_box()

```
void lj_box (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the Lennard-Jones potential for rigid box boundary conditions between two particle types.

Setup of Lennard-Jones potential for rigid box boundary conditions between particle types type1 and type2.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 216 of file interaction.cpp.

8.9.1.6 lj_periodic()

```
void lj_periodic (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the Lennard-Jones potential for periodic boundary conditions between two particle types.

Setup of Lennard-Jones potential for periodic boundary conditions between particle types type1 and type2. This requires the cutoff <= half the box size because it only checks the nearest images.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 139 of file interaction.cpp.

8.10 Backend/include/system.h File Reference

```
#include <iostream>
#include <omp.h>
#include <string>
#include <sstream>
#include <vector>
#include <cmath>
#include "algorithm_constants.h"
```

Classes

- class System::system_state
- class System::input_params
- class System::constants_interaction
- class System::constants_thermostat
- class System::simulation

Namespaces

- System

8.11 Backend/include/thermo.h File Reference

```
#include "system.h"
#include "constants.h"
#include <cmath>
```

Functions

- void trans_ke (System::simulation &sim)

8.11.1 Function Documentation

8.11.1.1 trans_ke()

```
void trans_ke (
            System::simulation & sim )
```

Definition at line 3 of file thermo.cpp.

8.12 Backend/include/thermostat.h File Reference

```
#include "system.h"
#include "interaction.h"
#include "constants.h"
#include "integrate.h"
#include "algorithm_constants.h"
#include <cmath>
#include <algorithm>
#include <random>
```

Functions

- void call_thermostat (System::simulation &sim)

  This function calls all the required thermostats.
- void no_thermostat (System::simulation &sim, int type)

  Call when no thermostat is to be used.
- void anderson (System::simulation &sim, int type)

  Call when anderson thermostat is to be used.

8.12.1 Function Documentation

8.12.1.1 anderson()

```
void anderson (
            System::simulation & sim,
            int type )
```

Call when anderson thermostat is to be used.

Applies the anderson thermostat on the required particle types

Parameters

sim	Simulation being used
type	Type of particle for thermalization

Definition at line 18 of file thermostat.cpp.

8.12.1.2 call_thermostat()

```
void call_thermostat (
            System::simulation & sim )
```

This function calls all the required thermostats.

Calls all the thermostats and updates the velocities of particles according to the chosen thermostat.

Parameters

sim	Simulation being used

Definition at line 6 of file thermostat.cpp.

8.12.1.3 no_thermostat()

```
void no_thermostat (
            System::simulation & sim,
            int type )
```

Call when no thermostat is to be used.

Does nothing

Parameters

sim	Simulation being used
type	Type of particle for thermalization

Definition at line 14 of file thermostat.cpp.

8.13 Backend/include/universal_functions.h File Reference

Functions

- double gamma_func (double x)

- double vol_unit_sphere (int dim)
- double surface_unit_sphere (int dim)

8.13.1 Function Documentation

8.13.1.1 gamma_func()

```
double gamma_func (
             double x )
```

Definition at line 5 of file universal_functions.cpp.

8.13.1.2 surface_unit_sphere()

```
double surface_unit_sphere (
             int dim )
```

Definition at line 325 of file universal_functions.cpp.

8.13.1.3 vol_unit_sphere()

```
double vol_unit_sphere (
             int dim )
```

Definition at line 215 of file universal_functions.cpp.

8.14 Backend/include/write.h File Reference

```
#include "system.h"
```

Functions

- void write_traj (System::simulation &sim)

8.14.1 Function Documentation

8.14.1.1 write_traj()

```
void write_traj (
            System::simulation & sim )
```

Definition at line 4 of file write.cpp.

8.15 Backend/src/client.cpp File Reference

```
#include "client.h"
```

Functions

- int main ()

8.15.1 Function Documentation

8.15.1.1 main()

```
int main ( )
```

Definition at line 3 of file client.cpp.

8.16 Backend/src/correlations.cpp File Reference

```
#include "correlations.h"
```

8.17 Backend/src/initialize.cpp File Reference

```
#include "initialize.h"
#include <trng/yarn2.hpp>
#include <trng/uniform01_dist.hpp>
```

Functions

- void init_sim (System::simulation &sim)

8.17.1 Function Documentation

8.17.1.1 init_sim()

```
void init_sim (
             System::simulation & sim )
```

Definition at line 5 of file initialize.cpp.

8.18 Backend/src/integrate.cpp File Reference

```
#include "integrate.h"
```

Functions

- void integrate_verdet_periodic (System::simulation &sim)
- void integrate_verdet_box (System::simulation &sim)

8.18.1 Function Documentation

8.18.1.1 integrate_verdet_box()

```
void integrate_verdet_box (
             System::simulation & sim )
```

This function integrates the equation of motion for rigid box conditions This function integrates using the Leapfrog Algorithm

Parameters

sim	Simulation being integrated over

Definition at line 62 of file integrate.cpp.

8.18.1.2 integrate_verdet_periodic()

```
void integrate_verdet_periodic (
             System::simulation & sim )
```

This function integrates the equation of motion for periodic boundary conditions This function integrates using the Leapfrog Algorithm

Parameters

sim	Simulation being integrated over

Definition at line 10 of file integrate.cpp.

8.19 Backend/src/interaction.cpp File Reference

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include "universal_functions.h"
#include "interaction.h"
```

Functions

- void initialize_interactions (System::simulation &sim)

  Initializes the constant arrays for interactions for speed.
- double distance_periodic (System::simulation &sim, int type1, int n1, int type2, int n2)

  Returns the distance between two particles for periodic boundary conditions.
- void interact (System::simulation &sim)

  This function calls all the required interaction functions between the particles.
- void free_particles (System::simulation &sim, int type1, int type2)

  Setup the free particle interaction between two particle types.
- void lj_periodic (System::simulation &sim, int type1, int type2)

  Setup the Lennard-Jones potential for periodic boundary conditions between two particle types.
- void lj_box (System::simulation &sim, int type1, int type2)

  Setup the Lennard-Jones potential for rigid box boundary conditions between two particle types.

8.19.1 Function Documentation

8.19.1.1 distance_periodic()

```
double distance_periodic (
            System::simulation & sim,
            int type1,
            int n1,
            int type2,
            int n2 )
```

Returns the distance between two particles for periodic boundary conditions.

Returns the distance between two particles labelled number n1,n2 of type1,type2 respectively in the position array assuming periodic boundary conditions

Parameters

sim	Simulation being used
type1	Particle type of particle 1
type2	Particle type of particle 2
n1	Particle index of particle 1 in position[type1] array
n2	Particle index of particle 2 in position[type2] array

Definition at line 67 of file interaction.cpp.

8.19.1.2 free_particles()

```
void free_particles (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the free particle interaction between two particle types.

Setup of free particle interaction between particle types type1 and type2. This does nothing. The function is empty.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 124 of file interaction.cpp.

8.19.1.3 initialize_interactions()

```
void initialize_interactions (
            System::simulation & sim )
```

Initializes the constant arrays for interactions for speed.

The function initializes the arrays for more efficient computation by precomputing the required factors

Parameters

sim	Simulation being initialized

Definition at line 17 of file interaction.cpp.

8.19.1.4 interact()

```
void interact (
            System::simulation & sim )
```

This function calls all the required interaction functions between the particles.

Calls all the interactiosn and updates acceleration and energy arrays

Parameters

sim	Simulation being used

Definition at line 87 of file interaction.cpp.

8.19.1.5 lj_box()

```
void lj_box (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the Lennard-Jones potential for rigid box boundary conditions between two particle types.

Setup of Lennard-Jones potential for rigid box boundary conditions between particle types type1 and type2.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 216 of file interaction.cpp.

8.19.1.6 lj_periodic()

```
void lj_periodic (
            System::simulation & sim,
            int type1,
            int type2 )
```

Setup the Lennard-Jones potential for periodic boundary conditions between two particle types.

Setup of Lennard-Jones potential for periodic boundary conditions between particle types type1 and type2. This requires the cutoff <= half the box size because it only checks the nearest images.

Parameters

sim	Simulation being used
type1	First type of particle interacting
type2	Second type of particle interacting

Definition at line 139 of file interaction.cpp.

8.20 Backend/src/thermo.cpp File Reference

```
#include "thermo.h"
```

Functions

- void trans_ke (System::simulation &sim)

8.20.1 Function Documentation

8.20.1.1 trans_ke()

```
void trans_ke (
            System::simulation & sim )
```

Definition at line 3 of file thermo.cpp.

8.21 Backend/src/thermostat.cpp File Reference

```
#include "thermostat.h"
#include "trng/yarn2.hpp"
#include "trng/normal_dist.hpp"
#include "trng/uniform01_dist.hpp"
```

Functions

- void call_thermostat (System::simulation &sim)

  This function calls all the required thermostats.
- void no_thermostat (System::simulation &sim, int type)

  Call when no thermostat is to be used.
- void anderson (System::simulation &sim, int type)

  Call when anderson thermostat is to be used.

8.21.1 Function Documentation

8.21.1.1 anderson()

```
void anderson (
            System::simulation & sim,
            int type )
```

Call when anderson thermostat is to be used.

Applies the anderson thermostat on the required particle types

Parameters

sim	Simulation being used
type	Type of particle for thermalization

Definition at line 18 of file thermostat.cpp.

8.21.1.2 call_thermostat()

```
void call_thermostat (
            System::simulation & sim )
```

This function calls all the required thermostats.

Calls all the thermostats and updates the velocities of particles according to the chosen thermostat.

Parameters

sim	Simulation being used

Definition at line 6 of file thermostat.cpp.

8.21.1.3 no_thermostat()

```
void no_thermostat (
            System::simulation & sim,
            int type )
```

Call when no thermostat is to be used.

Does nothing

Parameters

sim	Simulation being used
type	Type of particle for thermalization

Definition at line 14 of file thermostat.cpp.

8.22 Backend/src/universal_functions.cpp File Reference

```
#include "universal_functions.h"
#include <cmath>
#include <iostream>
```

Functions

- double gamma_func (double x)
- double vol_unit_sphere (int dim)
- double surface_unit_sphere (int dim)

8.22.1 Function Documentation

8.22.1.1 gamma_func()

```
double gamma_func (
            double x )
```

Definition at line 5 of file universal_functions.cpp.

8.22.1.2 surface_unit_sphere()

```
double surface_unit_sphere (
            int dim )
```

Definition at line 325 of file universal_functions.cpp.

8.22.1.3 vol_unit_sphere()

```
double vol_unit_sphere (
            int dim )
```

Definition at line 215 of file universal_functions.cpp.

8.23 Backend/src/write.cpp File Reference

```
#include "write.h"
```

Functions

- void write_traj (System::simulation &sim)

8.23.1 Function Documentation

8.23.1.1 write_traj()

```
void write_traj (
            System::simulation & sim )
```

Definition at line 4 of file write.cpp.

8.24 Documentation/codeDocumentation.txt File Reference

Variables

- Implementation of Potentials Lennard Jones Potential
- Implementation of Potentials Lennard Jones the constants vector is as follows
- Implementation of Potentials Lennard Jones the constants vector is as this potential can only be used for upto space dimensions Implementation of Thermostats Anderson Thermostat
- Implementation of Potentials Lennard Jones the constants vector is as this potential can only be used for upto space dimensions Implementation of Thermostats Anderson a particle is given a speed from the gaussian distribution with mean and frac

8.24.1 Variable Documentation

8.24.1.1 follows

```
Implementation of Potentials Lennard Jones the constants vector is as follows
```

Definition at line 8 of file codeDocumentation.txt.

8.24.1.2 frac

```
Implementation of Potentials Lennard Jones the constants vector is as this potential can only
be used for upto space dimensions Implementation of Thermostats Anderson a particle is given a
speed from the gaussian distribution with mean and frac
```

Initial value:
```
{k_{b}*T_{req}}{m} variance with a probability of \nu per unit time.
    Here
```

Definition at line 24 of file codeDocumentation.txt.

8.24.1.3 Potential

```
Implementation of Potentials Lennard Jones Potential
```

Definition at line 3 of file codeDocumentation.txt.

8.24.1.4 Thermostat

```
Implementation of Potentials Lennard Jones the constants vector is as this potential can only
be used for upto space dimensions Implementation of Thermostats Anderson Thermostat
```

Definition at line 22 of file codeDocumentation.txt.

8.25 Documentation/errorDocumentation.txt File Reference

8.26 Documentation/installationDocumentation.txt File Reference

Functions

- To build this you would need to install trng library from go to MDGeneral Backend src For normal perform the profiling install and then run the program (mdgen_back). After this

Variables

- To build this program
- To build this you would need to install trng library from https
- To build this you would need to install trng library from go to MDGeneral Backend src Now
- To build this you would need to install trng library from go to MDGeneral Backend src For normal install

8.26.1 Function Documentation

8.26.1.1 program()

```
To build this you would need to install trng library from go to MDGeneral Backend src For
normal perform the profiling install and then run the program (
             mdgen_back  )
```

8.26.2 Variable Documentation

8.26.2.1 https

```
To build this you would need to install trng library from https
```

Definition at line 1 of file installationDocumentation.txt.

8.26.2.2 install

```
To build this you would need to install trng library from go to MDGeneral Backend src For
normal install
```

Definition at line 5 of file installationDocumentation.txt.

8.26.2.3 Now

```
To build this you would need to install trng library from go to MDGeneral Backend src Now
```

Definition at line 3 of file installationDocumentation.txt.

8.26.2.4 program

```
To build this program
```

Definition at line 1 of file installationDocumentation.txt.

8.27 README.md File Reference

Index