# 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 correlation
- class input_params
- class simulation
- class system_state

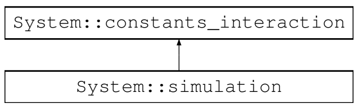
# Chapter 7

# Class Documentation

## 7.1 System::constants_interaction Class Reference

```
#include <MDGeneral/Backend/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 114 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 134 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] = $\epsilon$
interaction_const[i][j][1] = $\sigma$
interaction_const[i][j][2] = Cutoff radius/distance (r_cut)
interaction_const[i][j][3] = Truncated Potential (etrunc)
interaction_const[i][j][4] = $\sigma^6$
interaction_const[i][j][5] = Tail Energy (assuming constant distribution outside cutoff radius)

Definition at line 129 of file system.h.

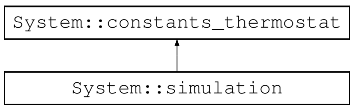
The documentation for this class was generated from the following file:

- MDGeneral/Backend/include/system.h

## 7.2 System::constants_thermostat Class Reference

```
#include <MDGeneral/Backend/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

  *This vector stores constants of the thermostats.*

### 7.2.1 Detailed Description

Definition at line 158 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 165 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
```

This vector stores constants of the thermostats.

Definition at line 161 of file system.h.

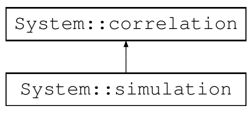
The documentation for this class was generated from the following file:

- MDGeneral/Backend/include/system.h

## 7.3 System::correlation Class Reference

```
#include <MDGeneral/Backend/include/system.h>
```

Inheritance diagram for System::correlation:



## Public Member Functions

- correlation (int n_types, int n_dimensions, std::vector< int > &n_particles, double runtime, double timestep)
- ∼correlation ()

## Public Attributes

- std::vector< std::vector< std::vector< double > > > velocity_initial

  *Stores the velocity of the particles at t=0.*
- std::vector< std::vector< double > > correlation_velocity

## 7.3.1 Detailed Description

Definition at line 185 of file system.h.

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 correlation()

```
System::correlation::correlation (
          int n_types,
          int n_dimensions,
          std::vector< int > & n_particles,
          double runtime,
          double timestep )  [inline]
```

This constructor reserves space for the correlation arrays and initial conditions

Definition at line 194 of file system.h.

### 7.3.2.2 ∼correlation()

```
System::correlation::∼correlation ( )
```

## 7.3.3 Member Data Documentation

#### 7.3.3.1 correlation_velocity

```
std::vector<std::vector<double> > System::correlation::correlation_velocity
```

Stores the velocity correlation for each particle type at each timestep (the n_types+1 th entry is the correlation over all types)
The format is correlation_velocity[step_number][particletype]

Definition at line 189 of file system.h.

#### 7.3.3.2 velocity_initial

```
std::vector<std::vector<std::vector<double> > > System::correlation::velocity_initial
```

Stores the velocity of the particles at t=0.

Definition at line 188 of file system.h.

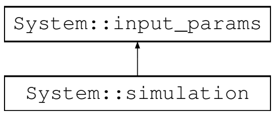
The documentation for this class was generated from the following file:

- MDGeneral/Backend/include/system.h

## 7.4 System::input_params Class Reference

```
#include <MDGeneral/Backend/include/system.h>
```

Inheritance diagram for System::input_params:



### Public Member Functions

- input_params (std::string input)

## Public Attributes

- int n_types

    *This represents the number of types of particles.*
- int n_dimensions

    *This represents the number of dimensions of the simulation (by default must be 3)*
- std::vector< int > n_particles

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

    *This defines the size of each timestep (dt)*
- double runtime

    *This defines the time for which to run the simulation.*
- int parallelize

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

    *This represents the mass of each type of particle (n_types sized)*
- std::vector< double > temperature_required

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

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

## 7.4.1 Detailed Description

Definition at line 68 of file system.h.

## 7.4.2 Constructor & Destructor Documentation

### 7.4.2.1 input_params()

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

Definition at line 81 of file system.h.

## 7.4.3 Member Data Documentation

### 7.4.3.1 mass

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

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

Definition at line 77 of file system.h.

#### 7.4.3.2 n_dimensions

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

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

Definition at line 72 of file system.h.

#### 7.4.3.3 n_particles

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

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

Definition at line 73 of file system.h.

#### 7.4.3.4 n_types

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

This represents the number of types of particles.

Definition at line 71 of file system.h.

#### 7.4.3.5 parallelize

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

If 1, parallelize. Else, do not parallelize.

Definition at line 76 of file system.h.

#### 7.4.3.6 periodic_boundary

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

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

Definition at line 79 of file system.h.

#### 7.4.3.7 runtime

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

This defines the time for which to run the simulation.

Definition at line 75 of file system.h.

#### 7.4.3.8 temperature_required

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

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

Definition at line 78 of file system.h.

#### 7.4.3.9 timestep

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

This defines the size of each timestep (dt)

Definition at line 74 of file system.h.

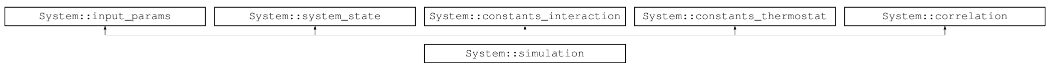
The documentation for this class was generated from the following file:

- MDGeneral/Backend/include/system.h

## 7.5 System::simulation Class Reference

```
#include <MDGeneral/Backend/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
- int total_steps

  *Total number of steps to be taken.*
- std::vector< int > dof

  *This stores the number of degrees of freedom for each molecule/particle type.*

### 7.5.1 Detailed Description

Definition at line 249 of file system.h.

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 simulation()

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

Definition at line 258 of file system.h.

#### 7.5.2.2 ∼simulation()

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

### 7.5.3 Member Data Documentation

#### 7.5.3.1 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 254 of file system.h.

### 7.5.3.2 dof

```
std::vector<int> System::simulation::dof
```

This stores the number of degrees of freedom for each molecule/particle type.

Definition at line 256 of file system.h.

### 7.5.3.3 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 253 of file system.h.

### 7.5.3.4 thermostat

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

This stores thermostats for different particle sets

Definition at line 252 of file system.h.

### 7.5.3.5 total_steps

```
int System::simulation::total_steps
```

Total number of steps to be taken.

Definition at line 255 of file system.h.

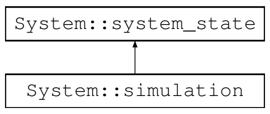
The documentation for this class was generated from the following file:

- MDGeneral/Backend/include/system.h

## 7.6 System::system_state Class Reference

```
#include <MDGeneral/Backend/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

  *The timestep number the system is in now.*
- int numpartot

  *Total number of particles.*

### 7.6.1 Detailed Description

Definition at line 14 of file system.h.

### 7.6.2 Constructor & Destructor Documentation

#### 7.6.2.1 system_state()

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

Definition at line 28 of file system.h.

### 7.6.3 Member Data Documentation

#### 7.6.3.1 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.

#### 7.6.3.2 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.

#### 7.6.3.3 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.

#### 7.6.3.4 energy_total

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

Defines the total energy at this instant

Definition at line 22 of file system.h.

#### 7.6.3.5 numpartot

```
int System::system_state::numpartot
```

Total number of particles.

Definition at line 27 of file system.h.

#### 7.6.3.6 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.

#### 7.6.3.7 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.

#### 7.6.3.8 state

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

The timestep number the system is in now.

Definition at line 26 of file system.h.

#### 7.6.3.9 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.

#### 7.6.3.10 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.

#### 7.6.3.11 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:

- MDGeneral/Backend/include/system.h

# Chapter 8

# File Documentation

## 8.1 MDGeneral/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 MDGeneral/GUI/Goals.txt File Reference

## 8.3 MDGeneral/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 MDGeneral/Backend/include/client.h File Reference

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

## 8.5 MDGeneral/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 MDGeneral/Backend/include/correlations.h File Reference

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

## Functions

- void initialize_correlations (System::simulation &sim)

  *This function initializes the correlation arrays by inputing intial values.*
- void correlate (System::simulation &sim)

  *Fills in the correlation vectors for this timestep.*

### 8.6.1 Function Documentation

#### 8.6.1.1 correlate()

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

Fills in the correlation vectors for this timestep.

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |

Definition at line 9 of file correlations.cpp.

#### 8.6.1.2 initialize_correlations()

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

This function initializes the correlation arrays by inputing intial values.

This function fills in the initial values (t=0) of the vectors over which correlation is to be found. This should be called only after initializing the initial vectors over which correlation is to be found.

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |

Definition at line 4 of file correlations.cpp.

## 8.7 MDGeneral/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 MDGeneral/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 64 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 MDGeneral/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.

Potential :

$$U(r) = 4\epsilon * [(\frac{\sigma}{r})^12 - (\frac{\sigma}{r})^6] - U(r_cut)$$

Force :

$$F(r) = 24\epsilon * \frac{\sigma^6}{r^7} * [2(\frac{\sigma}{r})^6 - 1]\hat{\mathbf{r}}$$

Here, the constants vector is as follows:
interaction_const[i][j][0] = $\epsilon$
interaction_const[i][j][1] = $\sigma$
interaction_const[i][j][2] = Cutoff radius/distance (r_cut)

interaction_const[i][j][3] = Truncated Potential (etrunc)
interaction_const[i][j][4] = $\sigma^6$
interaction_const[i][j][5] = Tail Energy (assuming constant distribution outside cutoff radius)

**Parameters**

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

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.

Potential :

$$U(r) = 4\epsilon * [(\frac{\sigma}{r})^1 2 - (\frac{\sigma}{r})^6] - U(r_c ut)$$

Force :

$$F(r) = 24\epsilon * \frac{\sigma^6}{r^7} * [2(\frac{\sigma}{r})^6 - 1]\hat{\mathbf{r}}$$

Here, the constants vector is as follows:
interaction_const[i][j][0] = $\epsilon$
interaction_const[i][j][1] = $\sigma$
interaction_const[i][j][2] = Cutoff radius/distance (r_cut)
interaction_const[i][j][3] = Truncated Potential (etrunc)
interaction_const[i][j][4] = $\sigma^6$
interaction_const[i][j][5] = Tail Energy (assuming constant distribution outside cutoff radius)

**Parameters**

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

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 MDGeneral/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::correlation
- class System::simulation

### Namespaces

- System

## 8.11 MDGeneral/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 MDGeneral/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 initialize_thermostats (System::simulation &sim)

    *Initializes the constant arrays of thermostats for speed.*
- 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.*
- void bussi (System::simulation &sim, int type)

    *Call when Bussi–Donadio–Parrinello 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

For this, a particle is given a speed from the gaussian distribution with 0 mean and $\frac{k_b*T_{req}}{m}$ variance with a probability of $\nu$ per unit time.

Here, the constants vector is as follows:
thermostat_const[i][0] = $\nu$
thermostat_const[i][1] = $\sqrt{\frac{k_b*T_{req}}{m}}$ for the particle type i

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |
| *type* | Type of particle for thermalization |

Definition at line 39 of file thermostat.cpp.

#### 8.12.1.2 bussi()

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

Call when Bussi–Donadio–Parrinello thermostat is to be used.

Applies the Bussi–Donadio–Parrinello thermostat on the required particle types. Implemented according to the algorithm described in J. Chem. Phys. 126, 014101 (2007); https://doi.org/10.1063/1.2408420
On every timestep, we scale the velocities by, $\alpha$, where,

$$\alpha^2 = a + \frac{b}{K}\sum_{i=1}^{N_f} R_i^2 + c\frac{R_1}{\sqrt{K}}$$

Where, $R_i$ are gaussian random numbers, K is kinetic energy of system, $N_f$ is number of degrees of freedom (translational)
Also,

$$a = e^{-\Delta t/\tau}$$
$$b = (1-a)\frac{\bar{K}}{N_f} = (1-a)\sqrt{T_{req}/2}$$
$$c = 2\sqrt{ab}$$

Here, the constants vector is as follows:
thermostat_const[i][0] = $tau$ (the relaxation time parameter)
thermostat_const[i][1] = $a$
thermostat_const[i][2] = $b$
thermostat_const[i][3] = $c$

Here, exponentiation is done by using a fourth order taylor expansion as it is upto 3 times more efficient and more accurate (even for |x| $<$ 10^-4 ) than the cpp implementation

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |
| *type* | Type of particle for thermalization |

Definition at line 66 of file thermostat.cpp.

#### 8.12.1.3 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 27 of file thermostat.cpp.

#### 8.12.1.4 initialize_thermostats()

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

Initializes the constant arrays of thermostats for speed.

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

**Parameters**

| | |
|---|---|
| *sim* | Simulation being initialized |

Definition at line 7 of file thermostat.cpp.

#### 8.12.1.5 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 35 of file thermostat.cpp.

## 8.13 MDGeneral/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 MDGeneral/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 MDGeneral/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 MDGeneral/Backend/src/correlations.cpp File Reference

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

## Functions

- void initialize_correlations (System::simulation &sim)

    *This function initializes the correlation arrays by inputing intial values.*
- void correlate (System::simulation &sim)

    *Fills in the correlation vectors for this timestep.*

### 8.16.1 Function Documentation

#### 8.16.1.1 correlate()

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

Fills in the correlation vectors for this timestep.

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |

Definition at line 9 of file correlations.cpp.

#### 8.16.1.2 initialize_correlations()

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

This function initializes the correlation arrays by inputing intial values.

This function fills in the initial values (t=0) of the vectors over which correlation is to be found.
This should be called only after initializing the initial vectors over which correlation is to be found.

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |

Definition at line 4 of file correlations.cpp.

## 8.17 MDGeneral/Backend/src/initialize.cpp File Reference

```
#include "initialize.h"
#include <trng/yarn5s.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 MDGeneral/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 64 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 MDGeneral/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 MDGeneral/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 MDGeneral/Backend/src/thermostat.cpp File Reference

```
#include "thermostat.h"
#include "trng/yarn5.hpp"
#include "trng/normal_dist.hpp"
#include "trng/uniform01_dist.hpp"
#include "trng/gamma_dist.hpp"
```

### Functions

- void initialize_thermostats (System::simulation &sim)

  *Initializes the constant arrays of thermostats for speed.*
- 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.*
- void bussi (System::simulation &sim, int type)

  *Call when Bussi–Donadio–Parrinello 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

For this, a particle is given a speed from the gaussian distribution with 0 mean and $\frac{k_b * T_{req}}{m}$ variance with a probability of $\nu$ per unit time.

Here, the constants vector is as follows:
thermostat_const[i][0] = $\nu$
thermostat_const[i][1] = $\sqrt{\frac{k_b * T_{req}}{m}}$ for the particle type i

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |
| *type* | Type of particle for thermalization |

Definition at line 39 of file thermostat.cpp.

#### 8.21.1.2 bussi()

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

Call when Bussi–Donadio–Parrinello thermostat is to be used.

Applies the Bussi–Donadio–Parrinello thermostat on the required particle types. Implemented according to the algorithm described in J. Chem. Phys. 126, 014101 (2007); `https://doi.org/10.1063/1.2408420`
On every timestep, we scale the velocities by, $\alpha$, where,

$$\alpha^2 = a + \frac{b}{K}\sum_{i=1}^{N_f} R_i^2 + c\frac{R_1}{\sqrt{K}}$$

Where, $R_i$ are gaussian random numbers, K is kinetic energy of system, $N_f$ is number of degrees of freedom (translational)
Also,

$$a = e^{-\Delta t/\tau}$$

$$b = (1-a)\frac{\bar{K}}{N_f} = (1-a)\sqrt{T_{req}/2}$$

$$c = 2\sqrt{ab}$$

Here, the constants vector is as follows:
thermostat_const[i][0] = $tau$ (the relaxation time parameter)
thermostat_const[i][1] = $a$
thermostat_const[i][2] = $b$
thermostat_const[i][3] = $c$

Here, exponentiation is done by using a fourth order taylor expansion as it is upto 3 times more efficient and more accurate (even for $|$x$| <$ 10^-4 ) than the cpp implementation

**Parameters**

| | |
|---|---|
| *sim* | Simulation being used |
| *type* | Type of particle for thermalization |

Definition at line 66 of file thermostat.cpp.

#### 8.21.1.3 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 27 of file thermostat.cpp.

#### 8.21.1.4 initialize_thermostats()

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

Initializes the constant arrays of thermostats for speed.

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

**Parameters**

| | |
|---|---|
| *sim* | Simulation being initialized |

Definition at line 7 of file thermostat.cpp.

#### 8.21.1.5 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 35 of file thermostat.cpp.

## 8.22 MDGeneral/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 MDGeneral/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 MDGeneral/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 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 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.  \n
     \n
    Here
```

Definition at line 21 of file codeDocumentation.txt.

#### 8.24.1.3 Potential

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

Definition at line 3 of file codeDocumentation.txt.

## 8.25 MDGeneral/Documentation/errorDocumentation.txt File Reference

## 8.26 MDGeneral/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 MDGeneral/README.md File Reference

# Index