Manual

Contents

Introduction

My name is Yuan-Chao Hu who is a PhD student currently in Institute of Physics, Chinese Academy of Sciences. You can reach me by email: ychu0213@gmail.com or visit my web: https://yuanchaohu.github.io/.

This package is designed for who are interested in analyzing the snapshots from molecular dynamics simulations, i.e. by LAMMPS. It is flexible for other computer simulations as long as you change the method of reading coordinates to suitable formats in ‘dump.py’. The modules in the package are written in Python3 by importing some high-efficiency modules like Numpy and Pandas. I strongly recommend the user to install Anaconda3 or/and Sublime Text 3 (with properly install Python and individual packages) to edit and run the python file.

To use the package efficiently, one intelligent way is to write a python script by importing desired modules and functions. In this way, all results can be obtained in sequence with suitable settings.

Read Snapshots

Syntax:

from dump import readdump
classname(inputfile).functioname()

- classname = readdump (for 2D and 3D systems)
- inputfile = snapshots from MD simulations (trajectories in one file)
- functionname = *read_onefile*

Example:

d = readdump('./dumpfile')
d.read_onefile()

Description:

This module reads the snapshots (or trajectories) from MD simulations. Only x, xs, xu coordinates are acceptable at current stage. The code is suitable for both two dimensional and three dimensional systems, but for 2D (x, y, z) are all needed. The coordinate queue should be '**id type x y z (…)**'. After executing the code in Example, all information including TimeStep, Particle Number, Box Lengths, Box Boundaries, Particle Types, Particle Positions is accessible. This module is the basis of the following analysis.

Important Notes: All snapshots should be in one file at this stage.

Pair Correlation Functions

Syntax:

from paircorrelationfunctions import gr3d (for 3D or gr2d for 2D)
classname(inputfile).functioname(args)

- classname = gr3d (3D systems) or gr2d (2D systems)
- inputfile = snapshots from MD simulations (trajectories in one file)
- functionname = *getresults, Unary, Binary, Ternary, Quarternary, Quinary, Senary* for 2D and 3D cases (see below for details)
- args = list of arguments to run the function (*outputfile, rdelta, ppp, results_path*)
  outputfile is the filename of outcomes without file path;
  rdelta is the bin size calculating g(r), the default value is 0.01;
  ppp is periodic boundary conditions along different directions, set 1 for yes and 0 for no at one direction. The default value is [1,1,1] for 3D and [1,1] for 2D; ***results_path*** is the file path of outputfile. The default value is '../../analysis/gr/'

Example:

gr3d('./dumpfile').getresults(outputfile = 'gr.dat', results_path = './gr/')

Please refer to the specific Class/Function lists below when using the functions. You can copy the function below and reset the parameters.

Description:

This module calculates the overall and partial pair correlation functions g(r) for three dimensional and two dimensional systems. g(r) is defined as:

$$g(r) = \frac{1}{N\rho}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\langle\delta(\vec{r}+\vec{r}_j-\vec{r}_i)\rangle$$

where *N* is particle number, ρ is number density. The code is written referring to ().

If you know the particle type number and want to get the returned numpy array of the results, please use functions from *Unary*() to *Senary*() according to your system. In these functions, the results will not only be written to an output file, but also will be returned as a numpy array for further analysis in Python. However, if you just want to get the analysis results in file, it is more convenient to choose the function *getresults*() without worrying about the particle type number. Because the code itself will set the type number based on the input file. However, no numpy arrays will be returned.

The cases in 3D and 2D are quite similar in the module. But only remember to change *ppp* argument because there are only two values in 2D.

Notes: At the current stage, *Senary*() only calculates the overall g(r).

Class/Function lists in the module (indentation indicates relationship):

```
Class gr3d (inputfile):
    getresults (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Unary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Binary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Ternary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Quarternary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Quinary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');
    Senary (outputfile, rdelta = 0.01, ppp = [1,1,1], results_path='../../analysis/gr/');

Class gr2d (inputfile):
    getresults (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Unary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Binary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Ternary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Quarternary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Quinary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
    Senary (outputfile, rdelta = 0.01, ppp = [1,1], results_path='../../analysis/gr/');
```

References:

Hu et al. Nature Communications, 6: 8310 (2015)
Hu et al. The Journal of Chemical Physics, **145** (10), 104503 (2016)
Hu et al. The Journal of Chemical Physics, **146** (2), 024507 (2017)
Hu et al. Physical Review E, **96** (2), 022613 (2017)

Structure Factors

Syntax:

from structurefactors import sq3d (for 3D or sq2d for 2D)
classname(inputfile).functioname(args)

- classname = sq3d (3D systems) or sq2d (2D systems)
- inputfile = snapshots from MD simulations (trajectories in one file)
- functionname = *getresults, Unary, Binary, Ternary, Quarternary, Quinary, Senary* for 2D and 3D cases (see below for details)
- args = list of arguments to run the function (*outputfile, results_path*)
  outputfile is the filename of outcomes without file path;
  results_path is the file path of outputfile. The default value is '../../analysis/gr/'

from structurefactors import functioname1
functioname1(args1)

- functionname1 = *wavevector3d, wavevector2d*
- args1 = *Numofq*
  Numofq is the considered number of wavenumber. Default is 500

Example:

sq3d('./dumpfile').getresults(outputfile = 'Sq.dat', results_path = './sq/')
wavevector3d(Numofq = 100)

Please refer to the specific Class/Function lists below when using the functions. You can copy the function below and reset the parameters.

Description:

This module calculates the overall and partial structure factors S(q) for three dimensional and two dimensional systems directly. S(q) is defined as:

$$\mathrm{S(q)} = N^{-1}\left\langle \sum_k \sum_j e^{-i\vec{\mathrm{q}}\cdot(\vec{\mathrm{r}}_k-\vec{\mathrm{r}}_j)} \right\rangle$$

where *N* is particle number. The code is written referring to (). In this code, if the box length *L* is smaller than 40.0, S(q) will be computed to *L*; but if *L* is larger than 40.0, S(q) will be computed to *L*/2. This aims to save the computer time and can be changed in the source code.

If you know the particle type number and want to get the returned numpy array of the results, please use functions from *Unary*() to *Senary*() according to your system. In these functions, the results will not only be written to an output file, but also will be returned as a numpy array for further analysis in Python. However, if you just want to get the analysis results in file, it is more convenient to choose the function *getresults*() without worrying about the particle type number. Because the code itself will set the type number based on the input file. However, no numpy arrays will be returned.

The module also provides wavenumber design method in functions *wavevector3d* and *wavevector2d*. A numpy array will be returned as [d, a, b, c] where $d = a^2 + b^2 + c^2$ for 3D or [d, a, b] where $d = a^2 + b^2$ for 2D. These functions are useful for further analysis related to 'structure factors' like the four-point dynamic structure factor.

The cases in 3D and 2D are quite similar in the module.

Class/Function lists in the module (indentation indicates relationship):

```
Class sq3d (inputfile):
    getresults(outputfile, results_path='../../analysis/sq/');
    Unary(outputfile, results_path='../../analysis/sq/');
    Binary(outputfile, results_path='../../analysis/sq/');
    Ternary(outputfile, results_path='../../analysis/sq/');
    Quarternary(outputfile, results_path='../../analysis/sq/');
    Quinary(outputfile, results_path='../../analysis/sq/');
    Senary(outputfile, results_path='../../analysis/sq/');

wavevector3d(Numofq = 500)
wavevector2d(Numofq = 500)

Class sq2d (inputfile):
    getresults(outputfile, results_path='../../analysis/sq/');
    Unary(outputfile, results_path='../../analysis/sq/');
    Binary(outputfile, results_path='../../analysis/sq/');
    Ternary(outputfile, results_path='../../analysis/sq/');
    Quarternary(outputfile, results_path='../../analysis/sq/');
    Quinary(outputfile, results_path='../../analysis/sq/');
    Senary(outputfile, results_path='../../analysis/sq/');
```

References:

Hu et al. The Journal of Chemical Physics, **146** (2), 024507 (2017)
Hu et al. Physical Review E, **96** (2), 022613 (2017)

Dynamical Properties

Syntax:
from dynamics import dynamics3d (for 3D or dynamics2d for 2D)
classname(inputfile).functioname(args)

- classname = dynamics3d (3D systems) or dynamics2d (2D systems)
- inputfile = snapshots from MD simulations (trajectories in one file)
- functionname = *total, partial, slowS4, fastS4* for 2D and 3D cases (see below for details)
- args = list of arguments to run the function (*outputfile*, *qmax, a, dt, results_path, X4time, X4timeset*)
  outputfile is the filename of outcomes without file path;
  qmax is the *q* values (usually the first peaks of structure factors) for calculating self-intermediate scattering functions; for the function *total*(), *qmax* is a value, but for the function *partial*(), *qmax* is a list containing the *q* values of different particle types in sequence;
  a is the cutoff value in the Overlap function Q(t), default is 1.0;
  dt is the timestep in MD simulations, default is 0.002;
  results_path is the file path of outputfile. The default value is '../../analysis/dynamics/';
  X4time is time scale (usually the peak time of the dynamic sysceptibility X4) for calculating four-point dynamic structure factor S4(q) in the function *slowS4*(). (Time Unit);
  X4timeset is similar to *X4time* above but for the function *fastS4*(). If set *X4timeset >0, fastS4*() will use the given value; but if set *X4timeset = 0*, *fastS4*() will use the internal calculated peak time scale of X4 of fast particles. (Time Unit)

Example:
dynamics3d('./dumpfile').total(outputfile = 'total.dat', qmax = 2.5, results_path = './dynamics/')

Please refer to the specific Class/Function lists below when using the functions. You can copy the function below and reset the parameters.

Description:
This module calculates the dynamical properties in 3D and 2D such as:
self-intermediate scattering function $F_s(q,t)$:

$$F_s(q,t) = \frac{1}{N}\left\langle \sum_{j=1}^{N} \exp\left[i\vec{q}\cdot\left(\vec{r}_j(t) - \vec{r}_j(0)\right)\right]\right\rangle$$

$F_s(q,t)$ susceptibility $\chi_4(t)$ ($F_s(q,t)$ is the non-averaged values):

$$\chi_4(t) = N^{-1}[\langle F_s(q,t)^2\rangle - \langle F_s(q,t)\rangle^2]$$

Overlap function $Q(t)$:

$$Q(t) = N^{-1}\langle \textstyle\sum_{j=1}^{N} \omega\left(|\vec{r}_j(0) - \vec{r}_j(t)|\right)\rangle$$

slow particles: where $\omega(r) = 1$ if $r \le a$ and zero otherwise

fast particles: where $\omega(r) = 1$ if $r \ge a$ and zero otherwise

Dynamic susceptibility $\chi_4(t)$ ($Q(t)$ is the non-averaged values):

$$\chi_4(t) = N^{-1}[\langle Q(t)^2\rangle - \langle Q(t)\rangle^2]$$

mean-square displacements $\langle \Delta r^2(t)\rangle$:

$$\langle \Delta r^2(t)\rangle = \frac{1}{N}\left\langle \sum_{j=1}^{N} \left[\vec{r}_j(t) - \vec{r}_j(0)\right]^2\right\rangle$$

Non-Gaussion parameter $\alpha_2(t)$:

$$\alpha_2(t) = \frac{3\langle \Delta r^4(t)\rangle}{5\langle \Delta r^2(t)\rangle^2} - 1 \ \ (3\text{D});\ \ \alpha_2(t) = \frac{\langle \Delta r^4(t)\rangle}{2\langle \Delta r^2(t)\rangle^2} - 1 \ \ (2\text{D})$$

To calculate $F_s(q,t)$, a wavenumber is required, which is usually the first peak of corresponding structure factors so you should first calculate the structure factors. If you are only interested in the overall dynamics without considering particle type, choose the function *total*() to calculate the above dynamic properties. The results in a numpy array will be returned. If you are interested in the dynamics of different particle types, choose the function *partial*() to calculate the above dynamic properties. A list containing results of different particle types (in numpy array) in sequence will be returned.

Four-point dynamic structure factor $S_4(q;t)$:

$$S_4(q;t) = N^{-1}[\langle W(\vec{q},t)W(-\vec{q},t)\rangle - \langle W(\vec{q},t)\rangle\langle W(-\vec{q},t)\rangle]$$

$$W(\vec{q};t) = \sum_{j=1}^{N} \exp\left[i\vec{q}\cdot\vec{r}_j(0)\right]\omega\left(\left|\vec{r}_j(t) - \vec{r}_j(0)\right|\right)$$

slow particles: where $\omega(r) = 1$ if $r \le a$ and zero otherwise

fast particles: where $\omega(r) = 1$ if $r \ge a$ and zero otherwise

To calculate $S_4(q;t)$, a time scale to calculate particle mobility is required, which is usually defined as the peak time scale of X4. In this code, S4 for slow and fast particles are calculable with function *slowS4*() and *fastS4*(), respectively. The difference lies in calculating the mobility field as defined above. In this code, if the box length *L* is smaller than 40.0, S(q) will be computed to *L*; but if *L* is larger than 40.0, S(q) will be computed to *L*/2. This aims to save the computer time can compare with the static structure factors and can be changed in the source code. After calculating S4, the four-point dynamic correlation length ξ_4 can be achieved by fitting the low wavenumber region to the function $S_4(q;\tau_p) = S_4(q=0;\tau_p)/[1+(q\xi_4)^2]$. (see Hu et al. The Journal of Chemical Physics, **146** (2), 024507 (2017))

The cases in 3D and 2D are quite similar in the module.

Notes: In the 2D case, the module only calculates the absolute dynamics without considering the Mermin-Wagner fluctuations.

Class/Function lists in the module (indentation indicates relationship):

```
Class dynamics3d (inputfile):
   total(outputfile, qmax, a = 1.0, dt = 0.002, results_path = '../../analysis/dynamics');
   partial(outputfile, qmax, a = 1.0, dt = 0.002, results_path = '../../analysis/dynamics');
   slowS4(outputfile, X4time, dt = 0.002, a= 1.0, results_path= '../../analysis/dynamics');
   fastS4(outputfile, a=1.0, dt=0.002, X4timeset=0, results_path= '../../analysis/dynamics');

Class dynamics2d (inputfile):
   total(outputfile, qmax, a = 1.0, dt = 0.002, results_path = '../../analysis/dynamics');
   partial(outputfile, qmax, a = 1.0, dt = 0.002, results_path = '../../analysis/dynamics');
   slowS4(outputfile, X4time, dt= 0.002, a = 1.0, results_path= '../../analysis/dynamics');
   fastS4(outputfile, a = 1.0, dt = 0.002, X4timeset = 0, results_path = '../../analysis/dynamics');
```

References:

Hu et al. Nature Communications, 6: 8310 (2015)
Hu et al. The Journal of Chemical Physics, **145** (10), 104503 (2016)
Hu et al. The Journal of Chemical Physics, **146** (2), 024507 (2017)
Hu et al. Physical Review E, **96** (2), 022613 (2017)