lammps-interface

Release 0.1.1

Peter Boyd, Mohamad Moosavi, Matthew Witman

Aug 22, 2019

CONTENTS:

1 Development **1**
- 1.1 Build the documentation locally . . . 1
- 1.2 Sphinx cheat sheet . . . 1

2 lammps_interface package **3**
- 2.1 Submodules . . . 3
- 2.2 lammps_interface.BTW module . . . 3
- 2.3 lammps_interface.CIFIO module . . . 3
- 2.4 lammps_interface.Dubbeldam module . . . 4
- 2.5 lammps_interface.ForceFields module . . . 4
- 2.6 lammps_interface.InputHandler module . . . 15
- 2.7 lammps_interface.MOFFF module . . . 15
- 2.8 lammps_interface.Molecules module . . . 15
- 2.9 lammps_interface.atomic module . . . 17
- 2.10 lammps_interface.ccdc module . . . 17
- 2.11 lammps_interface.create_cluster module . . . 17
- 2.12 lammps_interface.create_cluster_v2 module . . . 17
- 2.13 lammps_interface.dreiding module . . . 21
- 2.14 lammps_interface.gas_models module . . . 21
- 2.15 lammps_interface.generic_raspa module . . . 21
- 2.16 lammps_interface.lammps_main module . . . 21
- 2.17 lammps_interface.lammps_potentials module . . . 23
- 2.18 lammps_interface.mof_sbus module . . . 43
- 2.19 lammps_interface.structure_data module . . . 43
- 2.20 lammps_interface.uff module . . . 48
- 2.21 lammps_interface.uff4mof module . . . 48
- 2.22 lammps_interface.uff_nonbonded module . . . 49
- 2.23 lammps_interface.water_models module . . . 49
- 2.24 Module contents . . . 49

3 Installation **51**

4 Usage **53**
- 4.1 Command line interface . . . 53
- 4.2 Jupyter notebook . . . 53

5 Licence **55**

6 Citation **57**

7 Indices and tables **59**

Python Module Index **61**

Index **63**

CHAPTER

ONE

DEVELOPMENT

1.1 Build the documentation locally

Build the documentation using:

```
pip install -e .[docs]  # install docs extra
cd docs/
make html  # build html documentation
```

After building, find the documentation in `docs/build/html/index.html`.

1.2 Sphinx cheat sheet

- Add code with syntax highlighting like this:

```
def f(var):
    return "string"
```

- Write your mathematical formulae using LaTeX, in line $\exp(-i2\pi)$ or displayed

$$f(x) = \int_0^\infty \exp\left(\frac{x^2}{2}\right) dx$$

- You want to refer to a particular function or class? You can!

  class lammps_interface.structure_data.**MolecularGraph**(***kwargs*)
  Contains all information relating a structure file to a fully described classical system.

  Important specific arguments for atomic nodes:

  - mass
  - force_field_type
  - charge
  - cartesian_coordinates
  - description {contains all information about electronic environment to make a decision on the final force_field_type}
  - hybridization [sp3, sp2, sp, aromatic]

  Important arguments for bond edges:

  - weight = 1
  - length
  - image_flag
  - force_field_type

- Check out the source of any page via the link in the bottom right corner.

reST source of this page:

```
Development
===========

Build the documentation locally
-------------------------------

Build the documentation using::

    pip install -e .[docs]  # install docs extra
    cd docs/
    make html  # build html documentation

After building, find the documentation in ``docs/build/html/index.html``.

Sphinx cheat sheet
------------------

  * Add code with syntax highlighting like this:

    .. code:: python

        def f(var):
            return "string"

  * Write your mathematical formulae using LaTeX,
    in line :math:`\exp(-i2\pi)` or displayed

    .. math:: f(x) = \int_0^\infty \exp\left(\frac{x^2}{2}\right)\,dx

  * You want to refer to a particular function or class? You can!

    .. autoclass:: lammps_interface.structure_data.MolecularGraph
       :noindex:

  * Check out the source of any page via the link
    in the bottom right corner.

|

reST source of this page:

.. literalinclude:: development.rst
```

CHAPTER

TWO

LAMMPS_INTERFACE PACKAGE

2.1 Submodules

2.2 lammps_interface.BTW module

Parameters for BTW-FF.

2.3 lammps_interface.CIFIO module

CIF format file I/O operations.

class lammps_interface.CIFIO.**CIF**(*name='structure'*, *file=None*)

Bases: object

__dict__ = mappingproxy({'atom_type_partial_charge': <staticmethod object>, 'cell_ang

__init__(*name='structure'*, *file=None*)

Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.CIFIO'

__str__()

Return str(self).

__weakref__

list of weak references to the object (if defined)

add_data(*block*, ***kwargs*)

static atom_site_constraints(*x*)

static atom_site_description(*x*)

static atom_site_fract_x(*x*)

static atom_site_fract_y(*x*)

static atom_site_fract_z(*x*)

static atom_site_fragment(*x*)

static atom_site_label(*x*)

static atom_site_type_symbol(*x*)

static atom_type_partial_charge(*x*)

static ccdc_geom_bond_type(*x*)

static cell_angle_alpha(*x*)

static cell_angle_beta(*x*)

static cell_angle_gamma(*x*)

static cell_length_a(*x*)

static cell_length_b(*x*)

static cell_length_c(*x*)

static general_label(*x*)

static geom_bond_atom_site_label_1(*x*)

static geom_bond_atom_site_label_2(*x*)

static geom_bond_distance(*x*)

static geom_bond_site_symmetry_2(*x*)

get_element_label(*el*)

get_non_loop_block(*line*)

get_time()

insert_block_order(*name*, *index=None*, *_REPLACE=False*)
Adds a block to the cif file in a specified order, unless index is specified, will not override existing order

static label(*x*)
special cases

read(*filename*)

lammps_interface.CIFIO.**get_time**()

2.4 lammps_interface.Dubbeldam module

Parameers for Dubbeldam force field.

2.5 lammps_interface.ForceFields module

Force field methods.

class lammps_interface.ForceFields.**BTW_FF**(***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
class2 angle

NOTE: We ignored the 5and6 order terms of polynomial since the functional is not implemented in LAMMPS!!

bond_term(*edge*)
class2 bond: 4-order polynomial

detect_ff_terms()
Assigning force field type of atoms

dihedral_term(*dihedral*)
fourier diherdral

improper_term(*improper*)
class2 improper

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Buckingham equation in MM3 type is used!

special_commands()

class lammps_interface.ForceFields.**Dreiding**(*graph=None*, *h_bonding=False*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, *h_bonding=False*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic cosine angle

E = 0.5*C*[cos(theta) - cos(theta0)]^2

This is available in LAMMPS as the cosine/squared angle style (NB. the prefactor includes the usual 1/2 term.)

if theta0 == 180, use

E = K*(1 + cos(theta))

This is available in LAMMPS as the cosine angle style

bond_term(*edge*)
The DREIDING Force Field contains two possible bond terms, harmonic and Morse. The authors recommend using harmonic as a default, and Morse potentials for more 'refined' calculations. Here we will assume a harmonic term by default, then the user can chose to switch to Morse if they so choose. (change type argument to 'morse')

E = 0.5 * K * (R - Req)^2

E = D * [exp{-(alpha*R - Req)} - 1]^2

detect_ff_terms()

dihedral_term(*dihedral*)
The DREIDING dihedral is of the form

E = 0.5*V*[1 - cos(n*(phi - phi0))]

LAMMPS has a similar potential 'charmm' which is described as

E = K*[1 + cos(n*phi - d)]

In this case the 'd' term must be multiplied by 'n' before inputting to lammps. In addition a +180 degrees out-of-phase shift must be added to 'd' to ensure that the potential behaves the same as the DREIDING article intended.

hbond_pot(*node*, *nbpot*, *hnode*)
DREIDING can describe hbonded donor and acceptors using a lj function or a morse potential

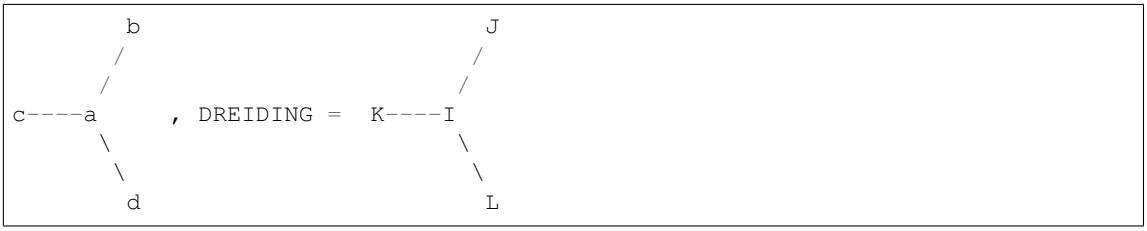
the morse potential is apparently better, so it will be default here

DREIDING III h-bonding terms specified in 10.1021/ja8100227.

Table S3 of the SI of 10.1021/ja8100227 is poorly documented, I have parameterized, to the best of my ability, what was intended in that paper. This message posted on the lammps-users message board http://lammps.sandia.gov/threads/msg36158.html was helpful N_3H == tertiary amine N_3P == primary amine N_3HP == protonated primary amine

nb. need connectivity information - this is accessed by self.graph

improper_term(*improper*)
Dreiding improper term.



For all non-planar configurations, DREIDING uses:

```
E = 0.5*C*(cos(phi) - cos(phi0))^2
```

For systems with planar equilibrium geometries, phi0 = 0

```
E = K*[1 - cos(phi)].
```

This is available in LAMMPS as the 'umbrella' improper potential.

pair_terms(*node*, *data*, *cutoff*, *nbpot='LJ'*, *hbpot='morse'*, *charges=True*)
DREIDING can adopt the exponential-6 or Ex6 = A*exp{-C*R} - B*R^{-6}

the Lennard-Jones type interactions. Elj = A*R^{-12} - B*R^{-6}

This will eventually be user-defined

special_commands()

class lammps_interface.ForceFields.**Dubbeldam**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

detect_ff_terms()
Instead of the painful experience of coding a huge set of 'if' statements over extended bonding neighbours to identify IRMOF-1, 10 and 16, all of the force field types will be detected from maximum cliques.

This means that failing to find the SBUs will result in a bad parameterization.

dihedral_term(*dihedral*)
The Dihedral potential has the form

U(torsion) = k * (1 + cos(m*theta - theta0))

in LAMMPS this potential can be accessed by the dihedral_style charmm

E = K * [1 + d*cos(n*theta - d)]

improper_term(*improper*)
The Improper potential has the form

U(torsion) = k * (1 + cos(m*theta - theta0))

in LAMMPS this potential can be accessed by the improper_style cvff

E = K * [1 + d*cos(n*theta)]

NO out of phase shift! to get a minima at 180 set d to -1. # all of Dubbeldam's terms are for 180 degrees out-of-plane movement so # this is fine.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)

special_commands()

class lammps_interface.ForceFields.**EPM2_CO2**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic angle term.

E = 0.5 * K * (theta - theta0)^2

Can be rigid or not with EPM2

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

just a placeholder in LAMMPS. The bond is rigid therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

detect_ff_terms()
CO2 consists of C and O, not too difficult.

dihedral_term(*dihedral*)
No dihedral potential in EPM2 CO2 model.

improper_term(*improper*)
No improper potential in EPM2 CO2 model.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Lennard - Jones potential for Cx and Ox.

special_commands()

class lammps_interface.ForceFields.**FMOFCu**(***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
class2 angle

bond_term(*edge*)
class2 bond

detect_ff_terms()

dihedral_term(*dihedral*)
fourier diherdral

improper_term(*improper*)
class2 diherdral

insert_graph(*graph*)

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Buckingham equation in MM3 type is used!

special_commands()

class lammps_interface.ForceFields.**ForceField**
Bases: object

__dict__ = mappingproxy({'__doc__': None, 'improper_term': <function ForceField.impr

__metaclass__
alias of abc.ABCMeta

__module__ = 'lammps_interface.ForceFields'

__weakref__
list of weak references to the object (if defined)

abstract angle_term()
Computes the angle parameters

abstract bond_term()
Computes the bond parameters

compute_angle_terms()

compute_atomic_pair_terms()

compute_bond_terms()

compute_dihedral_terms()

compute_force_field_terms()

compute_improper_terms()

abstract dihedral_term()
Computes the dihedral parameters

abstract improper_term()
Computes the improper dihedral parameters

class lammps_interface.ForceFields.**MOF_FF**(***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
class2 angle

Be careful that the 5and6 order terms are vanished here since they are not implemented in LAMMPS!! Etheta = 0.021914*Ktheta*(theta-theta0)^2[1-0.014(theta-theta0)+5.6(10^-5)*(theta-theta0)^2-7.0*(10^-7)*(theta-theta0)^3+9.0*(10^-10)*(theta-theta0)^4] (Allinger et. al. J.Am.Chem.Soc., Vol. 111, No. 23, 1989)

bond_term(*edge*)
class2 bond Es=71.94*Ks*(l-l0)^2[1-2.55(l-l0)+(7/12)*2.55*(l-l0)^2] (Allinger et. al. J.Am.Chem.Soc., Vol. 111, No. 23, 1989)

detect_ff_terms()
MOF-FF contains force field descriptions for three different inorganic SBUs: Cu-paddle-wheel Zn cluster –> IRMOF series Zr cluster –> UiO series

dihedral_term(*dihedral*)
fourier diherdral

Ew = (V1/2)(1 + cos w) + (V2/2)(1 - cos 2*w)+(V3/2)(1 + cos 3*w)+(V4/2)(1 + cos 4*w) (Allinger et. al. J.Am.Chem.Soc., Vol. 111, No. 23, 1989)

improper_term(*improper*)
Harmonic improper

pair_coul_term(*node1*, *node2*, *data*)
MOF-FF uses a damped gaussian for close-range interactions. This is added in a tabulated form.

k = 332.063711 kcal*angstroms/e^2

E_ij = k*qi*qj*erf(r_ij/sig_ij)/r_ij

— From Wolfram Alpha — F_ij = - (K*qi*qj) * (2/(pi^(.5) * sig_ij)) * [e^(-r_ij^2/sig_ij^2) / r_ij - erf(r_ij/sig_ij)/r_ij^2] —

N is set to 1000 by default

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Buckingham equation in MM3 type is used!

Also, Table for short range coulombic interactions

special_commands()

class lammps_interface.ForceFields.**OverwriteFF**(*struct*, *base_FF*)
Bases: *lammps_interface.ForceFields.ForceField*

Prepare a nanoprous material FF from a given structure for a known FF type.

Then overwrite any parameters that are supplied by user_input.txt

Methods are duplicated from UserFF, can reduce redundancy of code later if desired

__init__(*struct*, *base_FF*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

map_pair_unique_bond(*pair*, *descriptor*)

map_quadruplet_unique_dihedral(*quadruplet*, *descriptor*)

map_quadruplet_unique_improper(*quadruplet*, *descriptor*)

map_triplet_unique_angle(*triplet*, *descriptor*)

map_user_to_unique_atom(*descriptor*)

parse_user_input(*filename*)

write_missing_uniques(*description*)

class lammps_interface.ForceFields.**SPC_E**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic angle term.

E = 0.5 * K * (theta - theta0)^2

just a placeholder in LAMMPS. The angle is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

just a placeholder in LAMMPS. The bond is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

detect_ff_terms()
Water consists of O and H, not too difficult.

dihedral_term(*dihedral*)
No dihedral potential in SPC/E water model.

improper_term(*improper*)
No improper potential in SPC/E water model.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Lennard - Jones potential for OW and HW.

cutoff should be set to 9 angstroms, but this may be unrealistic. Also, no long range treatment of coulombic term! Otherwise this isn't technically the SPC/E model but Ewald should be used for periodic materials.

special_commands()

class lammps_interface.ForceFields.**TIP3P**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic angle term.

E = 0.5 * K * (theta - theta0)^2

just a placeholder in LAMMPS. The angle is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

just a placeholder in LAMMPS. The bond is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

detect_ff_terms()
Water consists of O and H, not too difficult.

dihedral_term(*dihedral*)
No dihedral potential in TIP3P water model.

improper_term(*improper*)
No improper potential in TIP3P water model.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Lennard - Jones potential for OW and HW.

special_commands()

class lammps_interface.ForceFields.**TIP4P**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*, *lammps_interface.Molecules.TIP4P_Water*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic angle term.

E = 0.5 * K * (theta - theta0)^2

just a placeholder in LAMMPS. The angle is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

just a placeholder in LAMMPS. The bond is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

detect_ff_terms()
Water consists of O and H, not too difficult.

dihedral_term(*dihedral*)
No dihedral potential in TIP4P water model.

improper_term(*improper*)
No improper potential in TIP4P water model.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Lennard - Jones potential for OW and HW.

special_commands()

class lammps_interface.ForceFields.**TIP5P**(*graph=None*, ***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph=None*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Harmonic angle term.

E = 0.5 * K * (theta - theta0)^2

just a placeholder in LAMMPS. The angle is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

bond_term(*edge*)
Harmonic term

E = 0.5 * K * (R - Req)^2

just a placeholder in LAMMPS. The bond is rigid and fixed by SHAKE in LAMMPS therefore an unambiguous extra flag must be added here to ensure the potential is not grouped with identical (however unlikely) potentials which are not rigid.

detect_ff_terms()
Water consists of O and H, not too difficult.

dihedral_term(*dihedral*)
No dihedral potential in TIP5P water model.

improper_term(*improper*)
No improper potential in TIP5P water model.

pair_terms(*node*, *data*, *cutoff*, ***kwargs*)
Lennard - Jones potential for OW and HW.

special_commands()

class lammps_interface.ForceFields.**UFF**(***kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

Parameterize the periodic material with the UFF parameters. NB: I have come across important information regarding the implementation of UFF from the author of MCCCS TOWHEE. It can be found here: (as of 05/11/2015) http://towhee.sourceforge.net/forcefields/uff.html

The ammendments mentioned that document are included here

__init__(***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
several cases exist where the type of atom in a particular environment is considered in both the parameters and the functional form of the term.

A small cosine fourier expansion in (theta) E_0 = K_{IJK} * {sum^{m}_{n=0}}C_{n} * cos(n*theta)

Linear, trigonal-planar, square-planar, and octahedral: two-term expansion of the above equation, n=0 as well as n=1, n=3, n=4, and n=4 for the above geometries. E_0 = K_{IJK}/n^2 * [1 - cos(n*theta)]

in Lammps, the angle syle called 'fourier/simple' can be used to describe this functional form.

general non-linear case: three-term Fourier expansion E_0 = K_{IJK} * [C_0 + C_1*cos(theta) + C_2*cos(2*theta)]

in Lammps, the angle style called 'fourier' can be used to describe this functional form.

Both 'fourier/simple' and 'fourier' are available from the USER_MISC package in Lammps, so be sure to compile Lammps with this package.

bond_term(*edge*)
Harmonic assumed

detect_ff_terms()

dihedral_term(*dihedral*)
Use a small cosine Fourier expansion

E_phi = 1/2*V_phi * [1 - cos(n*phi0)*cos(n*phi)]

this is available in Lammps in the form of a harmonic potential E = K * [1 + d*cos(n*phi)]

NB: the d term must be negated to recover the UFF potential.

improper_term(*improper*)
The improper function can be described with a fourier function

E = K*[C_0 + C_1*cos(w) + C_2*cos(2*w)]

NB: not sure if keep metal geometry is important here.

pair_terms(*node*, *data*, *cutoff*, *charges=True*)
Add L-J term to atom

special_commands()

uff_angle_type(*b*)

class lammps_interface.ForceFields.**UFF4MOF**(**kwargs*)
Bases: *lammps_interface.ForceFields.ForceField*

Parameterize the periodic material with the UFF4MOF parameters.

__init__(**kwargs*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
several cases exist where the type of atom in a particular environment is considered in both the parameters and the functional form of the term.

A small cosine fourier expansion in (theta) E_0 = K_{IJK} * {sum^{m}_{n=0}}C_{n} * cos(n*theta)

Linear, trigonal-planar, square-planar, and octahedral: two-term expansion of the above equation, n=0 as well as n=1, n=3, n=4, and n=4 for the above geometries. E_0 = K_{IJK}/n^2 * [1 - cos(n*theta)]

in Lammps, the angle syle called 'fourier/simple' can be used to describe this functional form.

general non-linear case: three-term Fourier expansion E_0 = K_{IJK} * [C_0 + C_1*cos(theta) + C_2*cos(2*theta)]

in Lammps, the angle style called 'fourier' can be used to describe this functional form.

Both 'fourier/simple' and 'fourier' are available from the USER_MISC package in Lammps, so be sure to compile Lammps with this package.

bond_term(*edge*)
Harmonic assumed

detect_ff_terms()
All new terms are associated with inorganic clusters, and the number of cases are extensive. Implementation of all of the metal types would require a bit of effort, but should be done in the near future.

dihedral_term(*dihedral*)
Use a small cosine Fourier expansion

E_phi = 1/2*V_phi * [1 - cos(n*phi0)*cos(n*phi)]

this is available in Lammps in the form of a harmonic potential E = K * [1 + d*cos(n*phi)]

NB: the d term must be negated to recover the UFF potential.

improper_term(*improper*)
Improper term described by a Fourier function

E = K*[C_0 + C_1*cos(w) + C_2*cos(2*w)]

pair_terms(*node*, *data*, *cutoff*, *charges=True*)
Add L-J term to atom

special_commands()

uff_angle_type(*b*)

class lammps_interface.ForceFields.**UserFF**(*graph*)
Bases: *lammps_interface.ForceFields.ForceField*

__init__(*graph*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.ForceFields'

angle_term(*angle*)
Computes the angle parameters

bond_term(*bond*)
Computes the bond parameters

compute_force_field_terms()

dihedral_term(*dihedral*)
Computes the dihedral parameters

improper_term(*improper*)
Computes the improper dihedral parameters

map_pair_unique_bond(*pair*, *descriptor*)

map_quadruplet_unique_dihedral(*quadruplet*, *descriptor*)

map_quadruplet_unique_improper(*quadruplet*, *descriptor*)

map_triplet_unique_angle(*triplet*, *descriptor*)

map_user_to_unique_atom(*descriptor*)

overwrite_force_field_terms()

parse_user_input(*filename*)

unique_angles()

unique_atoms()

unique_bonds()

unique_dihedrals()

unique_impropers()
How many times to list the same set of atoms ???

van_der_waals_pairs()

write_missing_uniques(*description*)

2.6 lammps_interface.InputHandler module

Argument parser for command line interface.

class lammps_interface.InputHandler.**Options**
Bases: object

__dict__ = mappingproxy({'_set_attr': <function Options._set_attr>, '__doc__': None,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.InputHandler'

__weakref__
list of weak references to the object (if defined)

_set_attr(*args*)

run_command_line_options()

lammps_interface.InputHandler.**git_revision_hash**()

2.7 lammps_interface.MOFFF module

MOF-FF parameters.

2.8 lammps_interface.Molecules module

Molecule class.

class lammps_interface.Molecules.**CO2**(***kwargs*)
Bases: *lammps_interface.Molecules.Molecule*

Carbon dioxide parent class, containing functions applicable to all CO2 models.

property O_coord
Define the oxygen coordinates assuming carbon is centered at '0'. angle gives the two oxygen atoms an orientation that deviates randomly from the default (lying along the x-axis).

__module__ = 'lammps_interface.Molecules'

approximate_positions(*C_pos=None*, *O_pos1=None*, *O_pos2=None*)
Input a set of approximate positions for the carbon and oxygens of CO2, and determine the lowest RMSD that would give the idealized model.

class lammps_interface.Molecules.**EPM2_CO2**(***kwargs*)
Bases: *lammps_interface.Molecules.CO2*

RCO = 1.149

__init__(***kwargs*)
Elementary Physical Model 2 (EPM2) of Harris & Yung (JPC 1995 99 12021) dx.doi.org/10.1021/j100031a034

__module__ = 'lammps_interface.Molecules'

class lammps_interface.Molecules.**Molecule**(***kwargs*)
Bases: *lammps_interface.structure_data.MolecularGraph*

__module__ = 'lammps_interface.Molecules'

property _type_

compute_all_angles()

rotation_from_vectors(*v1*, *v2*)
Obtain rotation matrix from sets of vectors. the original set is v1 and the vectors to rotate to are v2.

rotation_matrix(*axis*, *angle*)
returns a 3x3 rotation matrix based on the provided axis and angle

str(*atom_types={}*, *bond_types={}*, *angle_types={}*, *dihedral_types={}*, *improper_types={}*)
Create a molecule template string for writing to a file. Ideal for using fix gcmc or fix deposit in LAMMPS.

class lammps_interface.Molecules.**TIP4P_Water**(***kwargs*)
Bases: *lammps_interface.Molecules.Water*

HOH = 104.52

ROH = 0.9572

Rdum = 0.125

__init__(***kwargs*)
Class that provides a template molecule for TIP4P Water.

LAMMPS has some builtin features for this molecule which are not taken advantage of here.

I haven't bothered constructing a robust way to implement this special case, where there is no need to explicilty describe the dummy atom. This is handeled internally by LAMMPS.

__module__ = 'lammps_interface.Molecules'

property dummy

class lammps_interface.Molecules.**TIP5P_Water**(***kwargs*)
Bases: *lammps_interface.Molecules.Water*

DOD = 109.47

HOH = 104.52

ROH = 0.9572

Rdum = 0.7

__init__(***kwargs*)
Class that provides a template molecule for TIP5P Water.

No built in features for TIP5P so the dummy atoms must be explicitly described. Geometric features are evaluated to ensure the proper configuration to support TIP5P.

Initially set up a default TIP5P water configuration, then update if needed if superposing a TIP5P particle on an existing water.

__module__ = 'lammps_interface.Molecules'

property dummy
Given that the H_coords are determined from an angle with the x-axis in the xy plane, here we will also use the x-axis, only project the dummy atoms in the xz plane. This will produce, if all angles are 109.5 and distances the same, a perfect tetrahedron, however if the angles and distances are different, will produce a geometry with the highest possible symmetry.

class lammps_interface.Molecules.**Water**(***kwargs*)
Bases: *lammps_interface.Molecules.Molecule*

Water parent class, containing functions applicable to all water models.

property H_coord
Define the hydrogen coords based on HOH angle for the specific force field. Default axis for distributing the hydrogen atoms is the x-axis.

__module__ = 'lammps_interface.Molecules'

approximate_positions(*O_pos=None*, *H_pos1=None*, *H_pos2=None*)
Input a set of approximate positions for the oxygen and hydrogens of water, and determine the lowest RMSD that would give the idealized water model.

compute_midpoint_vector(*centre_vec*, *side1_vec*, *side2_vec*)
Define a vector oriented away from the centre_vec which is half-way between side1_vec and side2_vec. Ideal for TIP4P to define the dummy atom.

compute_orthogonal_vector(*centre_vec*, *side1_vec*, *side2_vec*)
Define a vector oriented orthogonal to two others, centred by the 'centre_vec'.

Useful for other water models with dummy atoms, as this can be used as a '4th' vector for the 'rotation_from_vectors' calculation (since 3 vectors defined by O, H, and H is not enough to orient properly). The real dummy atoms can then be applied once the proper rotation has been found.

2.9 lammps_interface.atomic module

Elemental information.

2.10 lammps_interface.ccdc module

Bond order information.

2.11 lammps_interface.create_cluster module

2.12 lammps_interface.create_cluster_v2 module

main.py

the program starts here.

class lammps_interface.create_cluster_v2.**Cluster**(*mgraph*, *xyz*, *offset*, *rcut*)
Bases: object

__dict__ = mappingproxy({'compute_cluster_in_disgraph': <function Cluster.compute_clu

__init__ (*mgraph*, *xyz*, *offset*, *rcut*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.create_cluster_v2'

__weakref__
list of weak references to the object (if defined)

cap_1D_organic()

cap_3D_organic()
Cap a MOF (or COF, COP, whatever), that is a 3-D network (i.e. doens't have 1D rods)

cap_by_material()

cap_primary_cluster()

cap_zeolite()
Cap a zeolite, not an extremeley difficult case

cap_zeolite_v2()
Cap a zeolite, not an extremeley difficult case

cart_dist (*pts1*, *pts2*)
Cartesian distance between 2 points

compute_cluster_in_disgraph()

compute_cluster_in_tree()

compute_primary_cluster()

compute_required_caps()

create_cluster_around_point()

create_cluster_around_point_v2()
A BFS search approach to creating clusters

create_cluster_around_point_v3()
Basic strategy here is to disconnect the graph at valid truncations

All the connected components are then condensed into a single node to form a secondary graph

This is the best way to make sure that in the end we perform the capping properly

cut_cappable_bonds()

cxtd_comp_cap()

cxtd_comp_continuous()
This part is critical, make sure that the final list of connected components forms a continuous network in secondary graph

In other words, the final cluster MUST not be a disconnected graph

cxtd_comp_convert_to_orig()
Temporary convert so that the files can be written

cxtd_comp_from_undirected()
Find connected components of the disconnect graph

cxtd_comp_num_keep()

cxtd_comp_secondary_graph()
Create a secondary graph with all the connected components

cxtd_comp_to_keep()
Identify which connected components have an atom within the cutoff radius

debug_edges_to_cut()

disconnect_1D_building_blocks()
Disconnect 1D rods so they can actually be capped

Best thing to do is still apply the same disconnection algorithm, only this time we are allowed to break a bond between a type in self.metals and [6,7,8]

disconnect_external_building_blocks()
Break a super simulation box into every possible component where each disconnected bond represents a cappable bond BUT we only break bonds that straddle or are external to the cluster cutoff radius

get_BFS_tree()

get_nth_heirarcy_BFS_tree(*n*)

get_start_and_kept_nodes()
Analyze a super box of a nanoporous material

Determine which nodes are inside the cutoff and which nodes are outside

Return the index of the starting node we will use to build the cluster

identify_1D_building_blocks()
By going through each component determined from all_external_building_blocks() we can determine if the MOF is 1D rod. If a component has two edges that eg have 'symflag' attribute of (4,x,x) and (6,x,x) respectively, then we found a component that spans across one crystallographic direction and reconnects with itself. This is the definition of a 1D rod MOF

identify_all_truncations()
Identify all truncations to make in the graph

identify_mat_type()
For now just a basic check to classify a zeolite vs an organic (MOF, COF, etc)

iterative_BFS_tree_structure(*v*)
Construct a dict with key that indexes depth of BFS tree, and the value is a set of all nodes at that depth

modify_structure_w_hydrogens()

nodes_that_DNE_in_origraph()

nodes_w_2plus_parents()

parse_sym_flag_for_directionality(*string*)
symm flag looks 'like 1_455' where 4 denotes a periodic bond in the x direction where 5 denotes a non periodic bond in the y, z direction

preliminary_truncate_BFS_tree()

truncate_all()

truncation_criteria(*up_node*, *down_node*)

update_num_keep()

visualize_n_levels_of_tree(*n*)

write_LSDALTON(*location*)
Write the LSDALTON file for DEC-MP2

write_cluster_to_host_xyz_v2()
Write the computed and capped cluster to an xyz file

write_cluster_to_xyz()
Write the computed and capped cluster to an xyz file

write_cluster_to_xyz_host_guest_v2(*include_guest*)
Write the computed and capped cluster to an xyz file

write_cutoff()
Write the cutoff that was used to make this cluster

write_map_from_MOLECULE_to_unique_types()

write_map_from_cluster_xyz_to_unique_types()

class lammps_interface.create_cluster_v2.**LammpsSimulation**(*options*)
Bases: object

__dict__ = mappingproxy({'count_dihedrals': <function LammpsSimulation.count_dihedral

__init__(*options*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.create_cluster_v2'

__weakref__
list of weak references to the object (if defined)

add_water_model(*ngraph*, *ff*)

assign_force_fields()

assign_molecule_ids(*graph*)

compute_cluster_box_size()
Added by MW b/c simbox size can need to be even bigger than a normal simbox when we are making a cluster from one unit cell

compute_molecules(*size_cutoff=0.5*)
Ascertain if there are molecules within the porous structure

compute_simulation_size()

construct_data_file()

construct_input_file()
Input file will depend on what the user wants to do

count_angles()

count_dihedrals()

count_impropers()

cut_molecule(*nodes*)

define_styles()

fixcount(*count=[]*)

groups(*ints*)

increment_graph_sizes(*inc=1*)

merge_graphs()

molecule_template(*mol*)
Construct a molecule template for reading and insertions in a LAMMPS simulation.

Not sure how the bonding, angle, dihedral, improper, and pair terms will be dealt with yet..

set_MDMC_config(*MDMC_config*)

set_cell(*cell*)

set_graph(*graph*)

split_graph()

unique_angles()

unique_atoms()
Computes the number of unique atoms in the structure

unique_bonds()
Computes the number of unique bonds in the structure

unique_dihedrals()

unique_impropers()

unique_pair_terms()

write_lammps_files()

lammps_interface.create_cluster_v2.**main**()

lammps_interface.create_cluster_v2.**read_xyz_center**(*filename*)

2.13 lammps_interface.dreiding module

Parameters for DREIDING force field.

2.14 lammps_interface.gas_models module

Gas models.

2.15 lammps_interface.generic_raspa module

RASPA file format and default parameters.

2.16 lammps_interface.lammps_main module

Lammps interface main program. Lammps simulations are setup here.

class lammps_interface.lammps_main.**LammpsSimulation**(*options*)
Bases: object

__dict__ = mappingproxy({'write_lammps_files': <function LammpsSimulation.write_lammps

__init__(*options*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_main'

__weakref__
list of weak references to the object (if defined)

add_co2_model(*ngraph*, *ff*)

add_water_model(*ngraph*, *ff*)

assign_force_fields()

assign_molecule_ids(*graph*)

compute_molecules(*size_cutoff=0.5*)
Ascertain if there are molecules within the porous structure

compute_simulation_size()

construct_data_file()

construct_input_file()
Input file construction based on user-defined inputs.

NB: This function is getting huge. We should probably break it up into logical sub-sections.

cut_molecule(*nodes*)

define_styles()

fixcount(*count=[]*)

groups(*ints*)

increment_graph_sizes(*inc=1*)

merge_graphs()

molecule_template(*mol*)
Construct a molecule template for reading and insertions in a LAMMPS simulation.

This combines two classes which have been separated conceptually - ForceField and Molecules. For some molecules, the force field is implicit within the structure (e.g. TIP5P_Water molecule must be used with the TIP5P ForceField). But one can imagine cases where this is not true (alkanes? CO2?).

set_MDMC_config(*MDMC_config*)

set_cell(*cell*)

set_graph(*graph*)

split_graph()

unique_angles(*g*)

unique_atoms(*g*)
Computes the number of unique atoms in the structure

unique_bonds(*g*)
Computes the number of unique bonds in the structure

unique_dihedrals(*g*)

unique_impropers(*g*)

unique_pair_terms()

write_lammps_files(*wd=None*)

lammps_interface.lammps_main.**main**()

2.17 lammps_interface.lammps_potentials module

Lammps potentital types.

class lammps_interface.lammps_potentials.**AnglePotential**
Bases: object

Class to hold angle styles that are implemented in lammps

class Charmm
Bases: object

Potential defined as

E = K*(theta - theta0)^2 + Kub*(r - Rub)^2

Input parameters: K, theta0, Kub, Rub

__dict__ = mappingproxy({'__str__': <function AnglePotential.Charmm.__str__>, '__d

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Class2
Bases: object

Potential defined as

E = Ea + Ebb + Eba Ea = K2*(theta - theta0)^2 + K3*(theta - theta0)^3 + K4*(theta - theta0)^4 Ebb = M*(r_ij - r1)*(r_jk - r2) Eba = N1*(r_ij - r1)*(theta - theta0) + N2*(rjk - r2)*(theta - theta0)

Input parameters for each potential: Angle: theta0 (degrees), K2(energy/rad^2), K3(energy/rad^3),K4(energy/rad^4) BondBond: bb, M(energy/distance^2), r1(distance), r2(distance) BondAngle: ba, N1(energy/distance^2), N2(energy/distance^2),r1(distance), r2(distance)

class BondAngle
Bases: object

Potential defined as —-> Eba = N1*(r_ij - r1)*(theta - theta0) + N2*(r_jk - r2)*(theta - theta0) <—-

__dict__ = mappingproxy({'__str__': <function AnglePotential.Class2.BondAngle.

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class BondBond
Bases: object

Potential defined as —-> Ebb = M*(r_ij - r1)*(r_jk - r2) <—-

__dict__ = mappingproxy({'__str__': <function AnglePotential.Class2.BondBond.__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'BondBond': <class 'lammps_interface.lammps_potentials.An

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Cosine
Bases: object

Potential defined as

E = K*[1 - cos(theta)]

Input parameters: K

__dict__ = mappingproxy({'__str__': <function AnglePotential.Cosine.__str__>, '__d

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosineDelta
Bases: object

Potential defined as

E = K*[1 - cos(theta-theta0)]

Input parameters: K, theta0

__dict__ = mappingproxy({'__str__': <function AnglePotential.CosineDelta.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosinePeriodic
Bases: object

Potential defined as

E = C*[1 - B*(-1)^n*cos(n*theta)]

Input parameters: C, B, n

__dict__ = mappingproxy({'__str__': <function AnglePotential.CosinePeriodic.__str_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosineShift
Bases: object

Potential defined as

E = -Umin/2 * [1 + cos(theta - theta0)]

Input parameters: Umin, theta0

__dict__ = mappingproxy({'__str__': <function AnglePotential.CosineShift.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosineShiftExp
Bases: object

Potential defined as

E = -Umin * [e^{-a*U(theta,theta0)} - 1] / [e^a - 1]

where U(theta,theta0) = -0.5*(1 + cos(theta - theta0))

Input parameters: Umin, theta0, a

__dict__ = mappingproxy({'__str__': <function AnglePotential.CosineShiftExp.__str_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosineSquared
Bases: object

Potential defined as

E = K*[cos(theta) - cos(theta0)]^2

Input parameters: K, theta0

__dict__ = mappingproxy({'__str__': <function AnglePotential.CosineSquared.__str__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Dipole
Bases: object

Potential defined as

E = K*(cos(gamma) - cos(gamma0))^2

Input parameters: K, gamma0

__dict__ = mappingproxy({'__str__': <function AnglePotential.Dipole.__str__>, '__d

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Fourier
Bases: object

Potential defined as

E = K*[C0 + C1*cos(theta) + C2*cos(2*theta)]

Input parameters: K, C0, C1, C2

__dict__ = mappingproxy({'__str__': <function AnglePotential.Fourier.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class FourierSimple
Bases: object

Potential defined as

E = K*[1 + c*cos(n*theta)]

Input parameters: K, c, n

__dict__ = mappingproxy({'__str__': <function AnglePotential.FourierSimple.__str__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Harmonic
Bases: object

Potential defined as

E = K*(theta - theta0)^2

Input parameters: K, theta0

special_flag is used to distinguish any extra inputs required for LAMMPS to run properly.

eg. special_flag="shake" will flag this angle potential as a shake angle.

__dict__ = mappingproxy({'__str__': <function AnglePotential.Harmonic.__str__>, '

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Quartic
Bases: object

Potential defined as

E = K2*(theta - theta0)^2 + K3*(theta - theta0)^3 + K4(theta - theta0)^4

Input parameters: theta0, K2, K3, K4

__dict__ = mappingproxy({'__str__': <function AnglePotential.Quartic.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Sdk
Bases: object

Potential defined as

E = K*(theta - theta0)^2

Input parameters: K, theta0

__dict__ = mappingproxy({'__str__': <function AnglePotential.Sdk.__str__>, '__doc_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Table
Bases: object

__dict__ = mappingproxy({'__doc__': None, '__weakref__': <attribute '__weakref__'

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'Table': <class 'lammps_interface.lammps_potentials.AnglePot

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

class lammps_interface.lammps_potentials.**BondPotential**
Bases: object

Class to hold bond styles that are implemented in lammps Purpose is to store info that the user wants to use to overwrite standard UFF output of lammps_interface

class Class2
Bases: object

Potential defined as

E = K2*(r-R0)^2 + K3*(r-R0)^3 + K4*(r-R0)^4

Input parameters: R0, K2, K3, K4.

__dict__ = mappingproxy({'__str__': <function BondPotential.Class2.__str__>, '__do

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Fene
Bases: object

Potential defined as

E = -0.5*K*R0^2 * ln[1-(r/R0)^2] + 4*eps*[(sig/r)^12 - (sig/r)^6] + eps

Input parameters: K, R0, eps, sig

__dict__ = mappingproxy({'__str__': <function BondPotential.Fene.__str__>, '__doc_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class FeneExpand
Bases: object

Potential defined as

E = -0.5*K*R0^2 * ln[1-(r-delta/R0)^2] + 4*eps*[(sig/r-delta)^12 - (sig/r-delta)^6] + eps

Input parameters: K, R0, eps, sig, delta

__dict__ = mappingproxy({'__str__': <function BondPotential.FeneExpand.__str__>, '

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Harmonic
Bases: object

Potential defined as

E = K*(r - R0)^2

Input parameters: K, R0

special_flag is used to distinguish any extra inputs required for LAMMPS to run properly.

eg. special_flag="shake" will flag this bond potential as a shake bond.

__dict__ = mappingproxy({'__str__': <function BondPotential.Harmonic.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class HarmonicShift
Bases: object

Potential defined as

E = Umin/(R0 - Rc)^2 * [(r-R0)^2 - (Rc - R0)^2]

Input parameters: Umin, R0, Rc

__dict__ = mappingproxy({'__str__': <function BondPotential.HarmonicShift.__str__>

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class HarmonicShiftCut
Bases: object

Potential defined as

E = Umin/(R0 - Rc)^2 * [(r-R0)^2 - (Rc - R0)^2]

Input parameters: Umin, R0, Rc

__dict__ = mappingproxy({'__str__': <function BondPotential.HarmonicShiftCut.__str

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Morse
Bases: object

Potential defined as

E = D*[1 - e^(-alpha*(r-R0))]^2

Input parameters: D, alpha, R0

__dict__ = mappingproxy({'__str__': <function BondPotential.Morse.__str__>, '__doc

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class NonLinear
Bases: object

Potential defined as

E = eps*(r-R0)^2 / [lamb^2 - (r-R0)^2]

Input parameters: eps, R0, lamb

__dict__ = mappingproxy({'__str__': <function BondPotential.NonLinear.__str__>, '_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Quartic
Bases: object

Potential defined as

E = K*(r-Rc)^2 * (r - Rc - B1) * (r - Rc - B2) + U0 + 4*eps*[(sig/r)^12 - (sig/r)^6] + eps

Input parameters: K, B1, B2, Rc, U0

__dict__ = mappingproxy({'__str__': <function BondPotential.Quartic.__str__>, '__c

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Table
Bases: object

Potential read from file.

__dict__ = mappingproxy({'__doc__': 'Potential read from file.', '__weakref__': <

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'Class2': <class 'lammps_interface.lammps_potentials.BondPot

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

class lammps_interface.lammps_potentials.**DihedralPotential**
Bases: object

Class to hold dihedral styles that are implemented in lammps

class Charmm
Bases: object

Potential defined as

E = K*[1 + cos(n*phi - d)]

Input parameters: K, n, d, w (weighting for 1 - 4 non-bonded interactions)

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Charmm.__str__>, '

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Class2
Bases: object

Potential deined as E = Ed + Embt + Eebt + Eat + Eaat + Ebb13 Ed = K1*[1 - cos(phi - phi1)] + K2*[1 - cos(2*phi - phi2)] + K3*[1 - cos(3*phi - phi3)] Embt = (r_jk - r2)*[A1*cos(phi) + A2*cos(2phi) + A3*cos(3phi)] Eebt = (r_ij - r1)*[B1*cos(phi) + B2*cos(2phi) + B3*cos(3phi)] + (r_kl - r3)*[C1*cos(phi) + C2*cos(2phi) + C3*cos(3phi)] Eat = (theta_ijk - theta1)*[D1*cos(phi) + D2*cos(2*phi) + D3*cos(3*phi)] + (theta_jkl - theta2)*[E1*cos(phi) + E2*cos(2*phi) + E3*cos(3phi)] Eaa = M*(theta_ijk - theta1)*(theta_jkl - theta2)*cos(phi) Ebb13 = N*(r_ij-r1)*(r_kl-r3)

class AngleAngleTorsion
Bases: object

Eaa = M*(theta_ijk - theta1)*(theta_jkl - theta2)*cos(phi)

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.AngleAng

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class AngleTorsion
Bases: object

Eat = (theta_ijk - theta1)*[D1*cos(phi) + D2*cos(2*phi) + D3*cos(3*phi)] + (theta_jkl - theta2)*[E1*cos(phi) + E2*cos(2*phi) + E3*cos(3phi)]

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.AngleTor

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class BondBond13
Bases: object

Ebb13 = N*(r_ij-r1)*(r_kl-r3)

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.BondBond

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class EndBondTorsion
Bases: object

Eebt = (r_ij - r1)*[B1*cos(phi) + B2*cos(2phi) + B3*cos(3phi)] + (r_kl - r3)*[C1*cos(phi) + C2*cos(2phi) + C3*cos(3phi)]

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.EndBond

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class MiddleBondTorsion
Bases: object

Embt = (r_jk - r2)*[A1*cos(phi) + A2*cos(2phi) + A3*cos(3phi)]

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.MiddleBo

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Class2.__str__>, '

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class CosineShiftExp
Bases: object

Potential defined as

E = -Umin*[e^{-a*U(theta,theta0)} - 1] / (e^a -1)

where U(theta, theta0) = -0.5*(1 + cos(theta-theta0))

Input parameters: Umin, theta0, a

__dict__ = mappingproxy({'__str__': <function DihedralPotential.CosineShiftExp.__s

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Fourier
Bases: object

Potential defined as

E = sum_i=1,m { Ki*[1.0 + cos(ni*theta - di)] }

Input parameters: m, Ki, ni, di

NB m is an integer dictating how many terms to sum, there should be 3*m + 1 total parameters.

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Fourier.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Harmonic
Bases: object

Potential defined as

E = K*[1 + d*cos(n*phi)]

Input parameters: K, d, n

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Harmonic.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Helix
Bases: object

Potential defined as

E = A*[1 - cos(theta)] + B*[1 + cos(3*theta)] + C*[1 + cos(theta + pi/4)]

Input parameters: A, B, C

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Helix.__str__>, '_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class MultiHarmonic
Bases: object

Potential defined as

E = sum_n=1,5{ An*cos^(n-1)(theta)}

Input parameters: A1, A2, A3, A4, A5

__dict__ = mappingproxy({'__str__': <function DihedralPotential.MultiHarmonic.__st

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Opls
Bases: object

Potential defined as

E = 0.5*K1*[1 + cos(theta)] + 0.5*K2*[1 - cos(2*theta)] + 0.5*K3*[1 + cos(3*theta)] + 0.5*K4*[1 - cos(4*theta)]

Input parameters: K1, K2, K3, K4

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Opls.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Quadratic
Bases: object

Potential defined as

E = K*(theta - theta0)^2

Input parameters: K, phi0

__dict__ = mappingproxy({'__str__': <function DihedralPotential.Quadratic.__str__>

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Table
Bases: object

Potential read from file.

__dict__ = mappingproxy({'__doc__': 'Potential read from file.', '__weakref__': <

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'Charmm': <class 'lammps_interface.lammps_potentials.Dihedra

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

class nHarmonic
Bases: object

Potential defined as

E = sum_i=1,n { Ai*cos^{i-1}(theta)

Input parameters: n, Ai

NB n is an integer dictating how many terms to sum, there should be n + 1 total parameters.

__dict__ = mappingproxy({'__str__': <function DihedralPotential.nHarmonic.__str__>

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class lammps_interface.lammps_potentials.**ImproperPotential**
Bases: object

Class to hold improper styles that are implemented in lammps

class Class2
Bases: object

Potential defined as

E = Ei + Eaa

Ei = K*[(chi_ijkl + chi_kjli + chi_ljik)/3 - chi0]^2 Eaa = M1*(theta_ijk - theta1)*(theta_kjl - theta3) +

M2*(theta_ijk - theta1)*(theta_ijl - theta2) + M3*(theta_ijl - theta2)*(theta_kjl - theta3)

Input parameters: K, chi0

class AngleAngle
Bases: object

Potential defined as Eaa = M1*(theta_ijk - theta1)*(theta_kjl - theta3) +
M2*(theta_ijk - theta1)*(theta_ijl - theta2) + M3*(theta_ijl - theta2)*(theta_kjl - theta3)
Input parameters: M1 M2 M3 theta1 theta2 theta3

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Class2.AngleAng

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Class2.__str__>, '

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Cossq
Bases: object

Potential defined as

E = 0.5*K*cos^2(chi - chi0)

Input parameters: K, chi0

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Cossq.__str__>, '_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Cvff
Bases: object

Potential defined as

E = K*[1 + d*cos(n*theta)]

Input parameters: K, d, n

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Cvff.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Fourier
Bases: object

Potential defined as

E = K*[C0 + C1*cos(omega) + C2*cos(2*omega)]

Input parameters: K, C0, C1, C2, a

the parameter a allows all three angles to be taken into account in an improper dihedral. It is not clear in the lammps manual what to set this to to turn it off/on, but the usual assumptions are 0/1.

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Fourier.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Harmonic
Bases: object

Potential defined as

E = K*(chi - chi0)^2

Input parameters: K, chi0

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Harmonic.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Ring
Bases: object

Potential defined as

E = 1/6*K*(delta_ijl + delta_ijk + delta_kjl)^6

where delta_ijl = cos(theta_ijl) - cos(theta0)

Input parameters: K, theta0

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Ring.__str__>, '__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Umbrella
Bases: object

Potential defined as

E = 0.5*K*[1 + cos(omega0)/sin(omega0)]^2 * [cos(omega) - cos(omega0)] if omega0 .ne. 0 (deg) E = K*[1 - cos(omega)] if omega0 = 0 (deg)

Input parameters: K, omega0

__dict__ = mappingproxy({'__str__': <function ImproperPotential.Umbrella.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'Umbrella': <class 'lammps_interface.lammps_potentials.Impro

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

class lammps_interface.lammps_potentials.**PairPotential**
Bases: object

Class to hold Pair styles that are implemented in lammps NB: list here is HUGE, update as needed..

class Buck
Bases: object

Potential defined as

E = A*exp{-r/rho} - C/r^{6}

__dict__ = mappingproxy({'__str__': <function PairPotential.Buck.__str__>, '__doc_

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class BuckCoulLong
Bases: object

Potential defined as

E = A*exp{-r/rho} - C/r^{6}

__dict__ = mappingproxy({'__str__': <function PairPotential.BuckCoulLong.__str__>,

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class HbondDreidingMorse
Bases: object

Potential defined as

E = D0*[exp{-2*alpha*(r-R0)} - 2*exp{-alpha*(r-R0)}]*cos^n(theta)

__dict__ = mappingproxy({'__str__': <function PairPotential.HbondDreidingMorse.__s

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class LjCharmmCoulLong
Bases: object

Potential defined as

E = 4*eps*[(sig/r)^12 - (sig/r)^6] r < rc

and coulombic terms dealt with a kspace solver

__dict__ = mappingproxy({'__str__': <function PairPotential.LjCharmmCoulLong.__str

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class LjCut
Bases: object

Potential defined as

E = 4*eps*[(sig/r)^12 - (sig/r)^6] r < rc

__dict__ = mappingproxy({'__str__': <function PairPotential.LjCut.__str__>, '__doc

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class LjCutCoulLong
Bases: object

Potential defined as

E = 4*eps*[(sig/r)^12 - (sig/r)^6] r < rc

and coulombic terms dealt with a kspace solver

__dict__ = mappingproxy({'__str__': <function PairPotential.LjCutCoulLong.__str__>

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class LjCutTip4pLong
Bases: object

Potential defined as

E = 4*eps*[(sig/r)^12 - (sig/r)^6] r < rc

and coulombic terms dealt with a kspace solver TIP4P water implicit charge points are included in ewald sum.

__dict__ = mappingproxy({'__str__': <function PairPotential.LjCutTip4pLong.__str__

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

class Table
Bases: object

A tabulated potential is used

LAMMPS keyword arguments are passes as kwargs

__dict__ = mappingproxy({'__str__': <function PairPotential.Table.__str__>, '__doc

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.lammps_potentials'

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

__dict__ = mappingproxy({'Table': <class 'lammps_interface.lammps_potentials.PairPote

__module__ = 'lammps_interface.lammps_potentials'

__weakref__
list of weak references to the object (if defined)

2.18 lammps_interface.mof_sbus module

MOF sbus.

lammps_interface.mof_sbus.**add_distance_matrix**(*graph*)

2.19 lammps_interface.structure_data module

Molecular graph and structure io methods.

class lammps_interface.structure_data.**Cell**
Bases: object

_Cell__mkcell()
Update the cell representation to match the parameters.

_Cell__mklammps()

_Cell__mkparam()
Update the parameters to match the cell.

__dict__ = mappingproxy({'get_cell': <function Cell.get_cell>, 'c': <property object

__init__()
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.structure_data'

__weakref__
list of weak references to the object (if defined)

property a
Magnitude of cell a vector.

property alpha
Cell angle alpha.

property b
Magnitude of cell b vector.

property beta
Cell angle beta.

property c
Magnitude of cell c vector.

property cell
Get the 3x3 vector cell representation.

property crystal_system
Return the IUCr designation for the crystal system.

property gamma
Cell angle gamma.

get_cell()
Get the 3x3 vector cell representation.

get_cell_inverse()
Get the 3x3 vector cell representation.

get_params()
Get the six parameter cell representation as a tuple.

property inverse
Inverted cell matrix for converting to fractional coordinates.

property lx

property ly

property lz

minimum_supercell(*cutoff*)
Calculate the smallest supercell with a half-cell width cutoff.

Increment from smallest cell vector to largest. So the supercell is not considering the 'unit cell' for each cell dimension.

property minimum_width
The shortest perpendicular distance within the cell.

mod_to_UC(*num*)
Retrun any fractional coordinate back into the unit cell

orthogonal_transformation()
Compute the transformation from the original unit cell to a supercell which has 90 degree angles between it's basis vectors. This is somewhat approximate, and the angles will not be EXACTLY 90 deg.

property params
Get the six parameter cell representation as a tuple.

set_cell(*value*)
Set cell and params from the cell representation.

set_params(*value*)
Set cell and params from the cell parameters.

update_supercell(*tuple*)

property volume
Calculate cell volume a.bxc.

property xy

property xz

property yz

class lammps_interface.structure_data.**MDMC_config**(*lammps_sim*)
Bases: object

Very sloppy for now but just doing the bare minimum to get this up and running for methane in flexible materials

__dict__ = mappingproxy({'__doc__': '\n Very sloppy for now but just doing the bare m

__init__(*lammps_sim*)
Initialize self. See help(type(self)) for accurate signature.

__module__ = 'lammps_interface.structure_data'

__weakref__
list of weak references to the object (if defined)

class lammps_interface.structure_data.**MolecularGraph**(***kwargs*)
Bases: networkx.classes.graph.Graph

Contains all information relating a structure file to a fully described classical system.

Important specific arguments for atomic nodes:

- mass
- force_field_type
- charge
- cartesian_coordinates
- description {contains all information about electronic environment to make a decision on the final force_field_type}
- hybridization [sp3, sp2, sp, aromatic]

Important arguments for bond edges:

- weight = 1
- length
- image_flag
- force_field_type

__iadd__(*newgraph*)

__init__(***kwargs*)
MolecularGraph constructor.

__module__ = 'lammps_interface.structure_data'

__or__(*graph*)

add_atomic_node(***kwargs*)
Insert nodes into the graph from the cif file

add_bond_edge(***kwargs*)
Add bond edges (weight factor = 1)

atomic_node_sanity_check()
Check for specific keyword/value pairs. Exit if non-existent

build_supercell(*sc*, *lattice*, *track_molecule=False*, *molecule_len=0*, *redefine=None*)
Construct a graph with nodes supporting the size of the supercell (sc) Oh man.. so ugly. NB: this replaces and overwrites the original unit cell data

with a supercell. There may be a better way to do this if one needs to keep both the super- and unit cells.

compute_angle_between(*l*, *m*, *r*)

compute_angles()
angles are attached to specific nodes, this way if a node is cut out of a graph, the angle comes with it.

b——a

/

/

c

Must be updated with different adjacent nodes if a supercell is requested, and the angle crosses a periodic image.

compute_bond_image_flag(*n1*, *n2*, *cell*)
Update bonds to contain bond type, distances, and min img shift.

compute_bond_typing()
Compute bond types and atom types based on the local edge environment. Messy, loads of 'ifs' is there a better way to catch chemical features?

compute_bonding(*cell*, *scale_factor=0.9*)
Computes bonds between atoms based on covalent radii.

compute_cartesian_coordinates(*cell*)
Compute the cartesian coordinates for each atom node

compute_dihedral_between(*a*, *b*, *c*, *d*)

compute_dihedrals()

Dihedrals are attached to specific edges in the graph.

a

b – c d

the edge between b and c will contain all possible dihedral angles between the neighbours of b and c (this includes a and d and other possible bonded atoms)

compute_improper_dihedrals()

Improper Dihedrals are attached to specific nodes in the graph.

a

b – c

d

the node b will contain all possible improper dihedral angles between the neighbours of b

compute_init_typing()
Find possible rings in the structure and initialize the hybridization for each atom. More refined determinations of atom and bond types is computed below in compute_bond_typing

compute_min_img_distances(*cell*)

compute_topology_information(*cell*, *tol*, *num_neighbours*)

coplanar(*node*)
Determine if this node, and it's neighbors are all co-planar.

correspondence_graph(*graph*, *tol*, *general_metal=False*, *node_subset=None*)
Generate a correspondence graph between the nodes and the SBU. tolerance is the distance tolerance for the edge generation in the correspondence graph.

count_angles()

count_dihedrals()

count_impropers()

detect_clusters(*num_neighbors*, *tol*, *type='Inorganic'*, *general_metal=False*)
Detect clusters such as the copper paddlewheel using maximum clique detection. This will assign specific atoms with a special flag for use when building their force field.

setting general_metal to True will allow for cluster recognition of inorganic SBUs while ignoring the specific element type of the metal, so long as it is a metal, it will be paired with other metals. This may increase the time for SBU recognition.

edges_iter2(*data=True*)

fractional(*coord*)

img_offset(*cells*, *cell*, *maxcell*, *flag*, *redefine*, *n1=0*)

in_cell(*coord*)

min_img(*coord*)

min_img_distance(*coords1*, *coords2*, *cell*)

node_dict_factory
alias of `collections.OrderedDict`

nodes_iter2(*data=True*)
Oh man, fixing to networkx 2.0

This probably breaks a lot of stuff in the code. THANKS NETWORKX!!!!!!!1 Extensive testing under way...

recurse_bonds_to_end(*node*, *pool=[]*, *visited=[]*)

recurse_linear_chains(*node*, *visited=[]*, *excluded=[]*)
Messy recursion function to return all unique chains from a set of atoms between two metals (or terminal atoms in the case of molecules)

redefine_lattice(*redefinition*, *lattice*)
Redefines the lattice based on the old lattice vectors. This was designed to convert non-orthogonal cells to orthogonal boxes, but it could in principle be used to convert any cell to any other cell. (As long as the redefined lattice are integer multiples of the old vectors)

reorder_labels(*reorder_dic*)
Re-order the labels of the nodes so that LAMMPS doesn't complain. This issue only arises when a supercell is built, but isolated molecules are not replicated in the supercell (by user request). This creates a discontinuity in the indices of the atoms, which breaks some features in LAMMPS.

show()

sorted_edge_list()

sorted_node_list()

store_original_size()

subgraph(*nodelist*)
Have to override subgraph from networkx. No idea why, but when Graph.subgraph is called in nx v >= 2.0, it returns a Graph object, not a MolecularGraph.

unwrap_node_coordinates(*cell*)
Must be done before supercell generation. This is a recursive method iterating over all edges. The design is totally unpythonic and written in about 5 mins.. so be nice (PB)

update_symflag(*cell*, *symflag*, *mincell*, *maxcell*)

lammps_interface.structure_data.**clean**(*name*)

lammps_interface.structure_data.**del_parenth**(*string*)

lammps_interface.structure_data.**from_CIF**(*cifname*)
Reads the structure data from the CIF - currently does not read the symmetry of the cell - does not unpack the assymetric unit (assumes P1) - assumes that the appropriate keys are in the cifobj (no error checking)

lammps_interface.structure_data.**write_CIF**(*graph*, *cell*)
Currently used for debugging purposes

lammps_interface.structure_data.**write_PDB**(*graph*, *cell*)
Program to write pdb file from structure graph. Doing this because MS v. 6 doesn't do the CONECT lines correctly.

lammps_interface.structure_data.**write_RASPA_CIF**(*graph*, *cell*, *classifier=0*)

Same as debugging cif write routine except _atom_site_label is now equal to _atom_site_description b/c RASPA uses _atom_site_label as the type for assigning FF params

lammps_interface.structure_data.**write_RASPA_sim_files**(*lammps_sim*, *classifier=0*)
Write the RASPA pseudo_atoms.def file for this MOF All generic adsorbates info automatically included Additional framework atoms taken from lammps_interface anaylsis

classifier = 0 -> UFF atom label is used for pseudo atoms classifier = 1 -> cif label is used for pseudo atoms (useful for

interface/defect systems where symmetry of charge density is very reduced)

2.20 lammps_interface.uff module

Parameters for Universal Force Field.

2.21 lammps_interface.uff4mof module

Parameters for UFF4MOF.

2.22 lammps_interface.uff_nonbonded module

Nonbonded parameters for Universal Force Field.

2.23 lammps_interface.water_models module

Water models.

2.24 Module contents

Lammps interface.

This program was designed for easy interface between the crystallographic information file (.cif) and the Large-scale Atomic Molecular Massively Parallel Simulator (Lammps).

CHAPTER

THREE

INSTALLATION

Simply install from PyPI:

```
pip install lammps-interface
```

For development purposes, clone the repository and install it from source:

```
pip install -e .
```

Note: In both cases, this adds `lammps-interface` to your `PATH`.

CHAPTER

FOUR

USAGE

4.1 Command line interface

To see the optional arguments type:

```
lammps-interface --help
```

To create Lammps simulation files for a given cif file type:

```
lammps-interface cif_file.cif
```

This will create Lammps simulation files with UFF parameters.

4.2 Jupyter notebook

In order to implement module to your project check out Jupyter notebooks provided in this repository in `/notebooks` for usage examples.

CHAPTER

FIVE

LICENCE

MIT licence (see LICENCE file)

CHAPTER

SIX

CITATION

The publication associated with this code is found here:

Boyd, P. G., Moosavi, S. M., Witman, M. & Smit, B. Force-Field Prediction of Materials Properties in Metal-Organic Frameworks. J. Phys. Chem. Lett. 8, 357–363 (2017).

https://dx.doi.org/10.1021/acs.jpclett.6b02532

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

l

lammps_interface, 49
lammps_interface.atomic, 17
lammps_interface.BTW, 3
lammps_interface.ccdc, 17
lammps_interface.CIFIO, 3
lammps_interface.create_cluster_v2, 17
lammps_interface.dreiding, 21
lammps_interface.Dubbeldam, 4
lammps_interface.ForceFields, 4
lammps_interface.gas_models, 21
lammps_interface.generic_raspa, 21
lammps_interface.InputHandler, 15
lammps_interface.lammps_main, 21
lammps_interface.lammps_potentials, 23
lammps_interface.mof_sbus, 43
lammps_interface.MOFFF, 15
lammps_interface.Molecules, 15
lammps_interface.structure_data, 43
lammps_interface.uff, 48
lammps_interface.uff4mof, 48
lammps_interface.uff_nonbonded, 49
lammps_interface.water_models, 49

INDEX

Symbols

_Cell__mkcell() (*lammps_interface.structure_data.Cell method*), 43
_Cell__mklammps() (*lammps_interface.structure_data.Cell method*), 43
_Cell__mkparam() (*lammps_interface.structure_data.Cell method*), 43
__dict__ (*lammps_interface.CIFIO.CIF attribute*), 3
__dict__ (*lammps_interface.ForceFields.ForceField attribute*), 8
__dict__ (*lammps_interface.InputHandler.Options attribute*), 15
__dict__ (*lammps_interface.create_cluster_v2.Cluster attribute*), 17
__dict__ (*lammps_interface.create_cluster_v2.LammpsSimulation attribute*), 20
__dict__ (*lammps_interface.lammps_main.LammpsSimulation attribute*), 21
__dict__ (*lammps_interface.lammps_potentials.AnglePotential attribute*), 28
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Charmm attribute*), 23
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Class2 attribute*), 24
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Class2.BondAngle attribute*), 23
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Class2.BondBond attribute*), 23
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Cosine attribute*), 24
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.CosineDelta attribute*), 24
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.CosinePeriodic attribute*), 25
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.CosineShift attribute*), 25
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.CosineShiftExp attribute*), 25
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.CosineSquared attribute*), 26
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Dipole attribute*), 26
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Fourier attribute*), 26
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.FourierS attribute*), 27
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Harmoni attribute*), 27
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Quartic attribute*), 27
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Sdk attribute*), 28
__dict__ (*lammps_interface.lammps_potentials.AnglePotential.Table attribute*), 28
__dict__ (*lammps_interface.lammps_potentials.BondPotential attribute*), 32
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Class2 attribute*), 28
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Fene attribute*), 29
__dict__ (*lammps_interface.lammps_potentials.BondPotential.FeneExpa attribute*), 29
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Harmonic attribute*), 29
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Harmonic attribute*), 30
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Harmonic attribute*), 30
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Morse attribute*), 30
__dict__ (*lammps_interface.lammps_potentials.BondPotential.NonLined attribute*), 31
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Quartic attribute*), 31
__dict__ (*lammps_interface.lammps_potentials.BondPotential.Table attribute*), 31
__dict__ (*lammps_interface.lammps_potentials.DihedralPotential attribute*), 36
__dict__ (*lammps_interface.lammps_potentials.DihedralPotential.Charm attribute*), 32
__dict__ (*lammps_interface.lammps_potentials.DihedralPotential.Class attribute*), 34
__dict__ (*lammps_interface.lammps_potentials.DihedralPotential.Class

attribute), 32
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Class2AngleTorsion attribute), 33
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Class2AngleTorsion13 attribute), 33
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Class2EndBondTorsion attribute), 33
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Class2MiddleBondTorsion attribute), 33
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.CosineShiftExp attribute), 34
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Fourier attribute), 34
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Harmonic attribute), 35
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Helix attribute), 35
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.MultiHarmonic attribute), 35
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Opls attribute), 36
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Quadratic attribute), 36
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.Table attribute), 36
__dict__ (lammps_interface.lammps_potentials.DihedralPotential.nharmonic attribute), 37
__dict__ (lammps_interface.lammps_potentials.ImproperPotential attribute), 40
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Class2 attribute), 37
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Class2Angle attribute), 37
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Cossq attribute), 38
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Cvff attribute), 38
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Harmonic attribute), 38
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Umbrella attribute), 39
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.Wilson attribute), 39
__dict__ (lammps_interface.lammps_potentials.ImproperPotential.ncvff attribute), 40
__dict__ (lammps_interface.lammps_potentials.PairPotential attribute), 43
__dict__ (lammps_interface.lammps_potentials.PairPotential.Buck attribute), 40
__dict__ (lammps_interface.lammps_potentials.PairPotential.Buck6d attribute), 40
__dict__ (lammps_interface.lammps_potentials.PairPotential.Hbond attribute), 41
__dict__ (lammps_interface.lammps_potentials.PairPotential.LjCharmmCoulLong attribute), 41
__dict__ (lammps_interface.lammps_potentials.PairPotential.LjCut attribute), 41
__dict__ (lammps_interface.lammps_potentials.PairPotential.LjCutCoulLong attribute), 42
__dict__ (lammps_interface.lammps_potentials.PairPotential.LjCutTip4pLong attribute), 42
__dict__ (lammps_interface.lammps_potentials.PairPotential.Table attribute), 42
__dict__ (lammps_interface.structure_data.Cell attribute), 43
__dict__ (lammps_interface.structure_data.MDMC_config attribute), 45
__dict__ (lammps_interface.structure_data.MolecularGraph method), 45
__init__() (lammps_interface.CIFIO.CIF method), 3
__init__() (lammps_interface.ForceFields.BTW_FF method), 4
__init__() (lammps_interface.ForceFields.Dreiding method), 5
__init__() (lammps_interface.ForceFields.Dubbeldam method), 6
__init__() (lammps_interface.ForceFields.EPM2_CO2 method), 7
__init__() (lammps_interface.ForceFields.FMOFCu method), 7
__init__() (lammps_interface.ForceFields.MOF_FF method), 8
__init__() (lammps_interface.ForceFields.OverwriteFF method), 9
__init__() (lammps_interface.ForceFields.SPC_E method), 10
__init__() (lammps_interface.ForceFields.TIP3P method), 10
__init__() (lammps_interface.ForceFields.TIP4P method), 11
__init__() (lammps_interface.ForceFields.TIP5P method), 12
__init__() (lammps_interface.ForceFields.UFF method), 12
__init__() (lammps_interface.ForceFields.UFF4MOF method), 13
__init__() (lammps_interface.ForceFields.UserFF method), 14
__init__() (lammps_interface.InputHandler.Options method), 15
__init__() (lammps_interface.Molecules.EPM2_CO2 method), 15
__init__() (lammps_interface.Molecules.TIP4P_Water method), 16
__init__() (lammps_interface.Molecules.TIP5P_Water method), 16
__init__() (lammps_interface.create_cluster_v2.Cluster method), 17

__init__() (lammps_interface.create_cluster_v2.LammpsSimulation method), 20
__init__() (lammps_interface.lammps_main.LammpsSimulation method), 21
__init__() (lammps_interface.lammps_potentials.AnglePotential.Charmm method), 23
__init__() (lammps_interface.lammps_potentials.AnglePotential.Class2 method), 24
__init__() (lammps_interface.lammps_potentials.AnglePotential.Class2BondAngle method), 23
__init__() (lammps_interface.lammps_potentials.AnglePotential.Class2BondBond method), 24
__init__() (lammps_interface.lammps_potentials.AnglePotential.Cosine method), 24
__init__() (lammps_interface.lammps_potentials.AnglePotential.CosineDelta method), 24
__init__() (lammps_interface.lammps_potentials.AnglePotential.CosinePeriodic method), 25
__init__() (lammps_interface.lammps_potentials.AnglePotential.CosineShift method), 25
__init__() (lammps_interface.lammps_potentials.AnglePotential.CosineShiftExp method), 25
__init__() (lammps_interface.lammps_potentials.AnglePotential.CosineSquared method), 26
__init__() (lammps_interface.lammps_potentials.AnglePotential.Dipole method), 26
__init__() (lammps_interface.lammps_potentials.AnglePotential.Fourier method), 26
__init__() (lammps_interface.lammps_potentials.AnglePotential.FourierSimple method), 27
__init__() (lammps_interface.lammps_potentials.AnglePotential.Harmonic method), 27
__init__() (lammps_interface.lammps_potentials.AnglePotential.Quartic method), 27
__init__() (lammps_interface.lammps_potentials.AnglePotential.Sdk method), 28
__init__() (lammps_interface.lammps_potentials.AnglePotential.Table method), 28
__init__() (lammps_interface.lammps_potentials.BondPotential.Class2 method), 28
__init__() (lammps_interface.lammps_potentials.BondPotential.Fene method), 29
__init__() (lammps_interface.lammps_potentials.BondPotential.FeneExpand method), 29
__init__() (lammps_interface.lammps_potentials.BondPotential.Harmonic method), 29
__init__() (lammps_interface.lammps_potentials.BondPotential.HarmonicShift method), 30
__init__() (lammps_interface.lammps_potentials.BondPotential.HarmonicShiftCut method), 30
__init__() (lammps_interface.lammps_potentials.BondPotential.Morse method), 30
__init__() (lammps_interface.lammps_potentials.BondPotential.NonLinear method), 31
__init__() (lammps_interface.lammps_potentials.BondPotential.Quartic method), 31
__init__() (lammps_interface.lammps_potentials.BondPotential.Table method), 31
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Charmm method), 32
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Class2 method), 34
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Class2BondAngle method), 32
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Class2BondBond method), 33
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Cosine method), 33
__init__() (lammps_interface.lammps_potentials.DihedralPotential.CosineDelta method), 33
__init__() (lammps_interface.lammps_potentials.DihedralPotential.CosinePeriodic method), 33
__init__() (lammps_interface.lammps_potentials.DihedralPotential.CosineShift method), 34
__init__() (lammps_interface.lammps_potentials.DihedralPotential.CosineShiftExp method), 34
__init__() (lammps_interface.lammps_potentials.DihedralPotential.CosineSquared method), 35
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Dipole method), 35
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Fourier method), 35
__init__() (lammps_interface.lammps_potentials.DihedralPotential.OPLS method), 36
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Quadratic method), 36
__init__() (lammps_interface.lammps_potentials.DihedralPotential.Table method), 36
__init__() (lammps_interface.lammps_potentials.DihedralPotential.nHarmonic method), 37
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Class2 method), 37
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Class2Diff method), 37
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Cossq method), 38
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Cvff method), 38
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Fourier method), 39
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Harmonic method), 39
__init__() (lammps_interface.lammps_potentials.ImproperPotential.HarmonicShift method), 39
__init__() (lammps_interface.lammps_potentials.ImproperPotential.HarmonicShiftCut method), 39
__init__() (lammps_interface.lammps_potentials.ImproperPotential.Umbrella method), 40
__init__() (lammps_interface.lammps_potentials.PairPotential.Buck method), 40

__init__() (lammps_interface.lammps_potentials.PairPotential.BuckCoulLong method), 40
__init__() (lammps_interface.lammps_potentials.PairPotential.HbondDreidingMorse method), 41
__init__() (lammps_interface.lammps_potentials.PairPotential.LjCharmmCoulLong method), 41
__init__() (lammps_interface.lammps_potentials.PairPotential.LjCut method), 41
__init__() (lammps_interface.lammps_potentials.PairPotential.LjCutCoulLong method), 42
__init__() (lammps_interface.lammps_potentials.PairPotential.LjCutTip4pLong method), 42
__init__() (lammps_interface.lammps_potentials.PairPotential.Table method), 43
__init__() (lammps_interface.structure_data.Cell method), 43
__init__() (lammps_interface.structure_data.MDMC_config method), 45
__init__() (lammps_interface.structure_data.MolecularGraph method), 45
__metaclass__ (lammps_interface.ForceFields.ForceField attribute), 8
__module__ (lammps_interface.CIFIO.CIF attribute), 3
__module__ (lammps_interface.ForceFields.BTW_FF attribute), 4
__module__ (lammps_interface.ForceFields.Dreiding attribute), 5
__module__ (lammps_interface.ForceFields.Dubbeldam attribute), 6
__module__ (lammps_interface.ForceFields.EPM2_CO2 attribute), 7
__module__ (lammps_interface.ForceFields.FMOFCu attribute), 7
__module__ (lammps_interface.ForceFields.ForceField attribute), 8
__module__ (lammps_interface.ForceFields.MOF_FF attribute), 8
__module__ (lammps_interface.ForceFields.OverwriteFF attribute), 9
__module__ (lammps_interface.ForceFields.SPC_E attribute), 10
__module__ (lammps_interface.ForceFields.TIP3P attribute), 10
__module__ (lammps_interface.ForceFields.TIP4P attribute), 11
__module__ (lammps_interface.ForceFields.TIP5P attribute), 12
__module__ (lammps_interface.ForceFields.UFF attribute), 12
__module__ (lammps_interface.ForceFields.UFF4MOF attribute), 13
__module__ (lammps_interface.ForceFields.UserFF attribute), 14
__module__ (lammps_interface.InputHandler.Options attribute), 15
__module__ (lammps_interface.Molecules.CO2 attribute), 15
__module__ (lammps_interface.Molecules.EPM2_CO2 attribute), 16
__module__ (lammps_interface.Molecules.Molecule attribute), 16
__module__ (lammps_interface.Molecules.TIP4P_Water attribute), 16
__module__ (lammps_interface.Molecules.TIP5P_Water attribute), 16
__module__ (lammps_interface.Molecules.Water attribute), 17
__module__ (lammps_interface.create_cluster_v2.Cluster attribute), 18
__module__ (lammps_interface.create_cluster_v2.LammpsSimulation attribute), 20
__module__ (lammps_interface.lammps_main.LammpsSimulation attribute), 21
__module__ (lammps_interface.lammps_potentials.AnglePotential attribute), 28
__module__ (lammps_interface.lammps_potentials.AnglePotential.Charm attribute), 23
__module__ (lammps_interface.lammps_potentials.AnglePotential.Class attribute), 24
__module__ (lammps_interface.lammps_potentials.AnglePotential.Class attribute), 23
__module__ (lammps_interface.lammps_potentials.AnglePotential.Class attribute), 24
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 24
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 24
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 25
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 25
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 25
__module__ (lammps_interface.lammps_potentials.AnglePotential.Cosin attribute), 26
__module__ (lammps_interface.lammps_potentials.AnglePotential.Dipol attribute), 26
__module__ (lammps_interface.lammps_potentials.AnglePotential.Fourie attribute), 26
__module__ (lammps_interface.lammps_potentials.AnglePotential.Fourie attribute), 27
__module__ (lammps_interface.lammps_potentials.AnglePotential.Harm attribute), 27
__module__ (lammps_interface.lammps_potentials.AnglePotential.Quart attribute), 27
__module__ (lammps_interface.lammps_potentials.AnglePotential.Sdk attribute), 28

__module__ (lammps_interface.lammps_potentials.AnglePotential.Table attribute), 28
__module__ (lammps_interface.lammps_potentials.BondPotential.Class2 attribute), 32
__module__ (lammps_interface.lammps_potentials.BondPotential.Fene attribute), 28
__module__ (lammps_interface.lammps_potentials.BondPotential.FeneExpand attribute), 29
__module__ (lammps_interface.lammps_potentials.BondPotential.Harmonic attribute), 29
__module__ (lammps_interface.lammps_potentials.BondPotential.HarmonicShift attribute), 30
__module__ (lammps_interface.lammps_potentials.BondPotential.HarmonicShiftCut attribute), 30
__module__ (lammps_interface.lammps_potentials.BondPotential.Morse attribute), 30
__module__ (lammps_interface.lammps_potentials.BondPotential.NonLinear attribute), 31
__module__ (lammps_interface.lammps_potentials.BondPotential.Quartic attribute), 31
__module__ (lammps_interface.lammps_potentials.BondPotential.Table attribute), 31
__module__ (lammps_interface.lammps_potentials.DihedralPotential attribute), 31
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Charmm attribute), 36
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2 attribute), 32
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2AngleAngle attribute), 34
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2AngleTorsion attribute), 32
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2BondBond attribute), 33
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2BondBond13 attribute), 33
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2EndBondTorsion attribute), 33
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Class2MiddleBondTorsion attribute), 33
__module__ (lammps_interface.lammps_potentials.DihedralPotential.CosineShiftExp attribute), 34
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Fourier attribute), 34
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Harmonic attribute), 35
__module__ (lammps_interface.lammps_potentials.DihedralPotential.OPLS attribute), 35
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Multi/Harmonic attribute), 35
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Quadratic attribute), 36
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Table attribute), 36
__module__ (lammps_interface.lammps_potentials.DihedralPotential.Table attribute), 36
__module__ (lammps_interface.lammps_potentials.DihedralPotential.nH attribute), 37
__module__ (lammps_interface.lammps_potentials.ImproperPotential attribute), 40
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Cl attribute), 38
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Cl attribute), 37
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Co attribute), 38
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Cv attribute), 38
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Fo attribute), 39
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Ha attribute), 39
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Ri attribute), 39
__module__ (lammps_interface.lammps_potentials.ImproperPotential.Un attribute), 40
__module__ (lammps_interface.lammps_potentials.PairPotential attribute), 43
__module__ (lammps_interface.lammps_potentials.PairPotential.Buck attribute), 40
__module__ (lammps_interface.lammps_potentials.PairPotential.BuckCo attribute), 40
__module__ (lammps_interface.lammps_potentials.PairPotential.Hbondl attribute), 41
__module__ (lammps_interface.lammps_potentials.PairPotential.LjChar attribute), 41
__module__ (lammps_interface.lammps_potentials.PairPotential.LjCut attribute), 41
__module__ (lammps_interface.lammps_potentials.PairPotential.LjCutC attribute), 42
__module__ (lammps_interface.lammps_potentials.PairPotential.LjCutTi attribute), 42
__module__ (lammps_interface.lammps_potentials.PairPotential.Table attribute), 43
__module__ (lammps_interface.structure_data.Cell attribute), 43
__module__ (lammps_interface.structure_data.MDMC_config attribute), 45
__module__ (lammps_interface.structure_data.MolecularGraph attribute), 45
__module__ (lammps_interface.structure_data.MolecularGraph method), 45
__module__ (lammps_interface.lammps_potentials.PairPotential.Buck method), 40
__module__ (lammps_interface.lammps_potentials.PairPotential.BuckCo method), 40
__module__ (lammps_interface.lammps_potentials.PairPotential.Hbondl method), 41

__repr__() (lammps_interface.lammps_potentials.PairPotential.LjCutCoulLong method), 41
__repr__() (lammps_interface.lammps_potentials.PairPotential.LjCutCoulNm method), 41
__repr__() (lammps_interface.lammps_potentials.PairPotential.LjCutCoulTable method), 42
__repr__() (lammps_interface.lammps_potentials.PairPotential.LjCutTip4pLong method), 42
__repr__() (lammps_interface.lammps_potentials.PairPotential.Table method), 43
__str__() (lammps_interface.CIFIO.CIF method), 3
__str__() (lammps_interface.lammps_potentials.AnglePotential.Charmm method), 23
__str__() (lammps_interface.lammps_potentials.AnglePotential.Class2 method), 24
__str__() (lammps_interface.lammps_potentials.AnglePotential.Class2BondAngle method), 23
__str__() (lammps_interface.lammps_potentials.AnglePotential.Class2BondBond method), 24
__str__() (lammps_interface.lammps_potentials.AnglePotential.Cosine method), 24
__str__() (lammps_interface.lammps_potentials.AnglePotential.CosineDelta method), 24
__str__() (lammps_interface.lammps_potentials.AnglePotential.CosinePeriodic method), 25
__str__() (lammps_interface.lammps_potentials.AnglePotential.CosineShift method), 25
__str__() (lammps_interface.lammps_potentials.AnglePotential.CosineShiftExp method), 25
__str__() (lammps_interface.lammps_potentials.AnglePotential.CosineSquared method), 26
__str__() (lammps_interface.lammps_potentials.AnglePotential.Dipole method), 26
__str__() (lammps_interface.lammps_potentials.AnglePotential.Fourier method), 26
__str__() (lammps_interface.lammps_potentials.AnglePotential.FourierSimple method), 27
__str__() (lammps_interface.lammps_potentials.AnglePotential.Harmonic method), 27
__str__() (lammps_interface.lammps_potentials.AnglePotential.Quartic method), 27
__str__() (lammps_interface.lammps_potentials.AnglePotential.Sdk method), 28
__str__() (lammps_interface.lammps_potentials.BondPotential.Class2 method), 29
__str__() (lammps_interface.lammps_potentials.BondPotential.Fene method), 29
__str__() (lammps_interface.lammps_potentials.BondPotential.FeneExpand method), 29
__str__() (lammps_interface.lammps_potentials.BondPotential.Harmonic method), 30
__str__() (lammps_interface.lammps_potentials.BondPotential.HarmonicShift method), 30
__str__() (lammps_interface.lammps_potentials.BondPotential.HarmonicShiftCut method), 30
__str__() (lammps_interface.lammps_potentials.BondPotential.LjCutCoulLong method), 31
__str__() (lammps_interface.lammps_potentials.BondPotential.Morse method), 31
__str__() (lammps_interface.lammps_potentials.BondPotential.NonLinear method), 31
__str__() (lammps_interface.lammps_potentials.BondPotential.Quartic method), 31
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Charmm method), 32
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Class2 method), 32
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Class2MBT method), 33
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Class2MBT2 method), 33
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Class2MBT2Opt method), 33
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Class2Opt method), 34
__str__() (lammps_interface.lammps_potentials.DihedralPotential.CosineDelta method), 34
__str__() (lammps_interface.lammps_potentials.DihedralPotential.CosinePeriodic method), 34
__str__() (lammps_interface.lammps_potentials.DihedralPotential.CosineShift method), 35
__str__() (lammps_interface.lammps_potentials.DihedralPotential.CosineShiftExp method), 35
__str__() (lammps_interface.lammps_potentials.DihedralPotential.CosineSquared method), 35
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Opls method), 36
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Quadratic method), 36
__str__() (lammps_interface.lammps_potentials.DihedralPotential.Simple method), 37
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Class2 method), 38
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Cossq method), 37
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Cossq2 method), 38
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Cvff method), 38
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Fourier method), 39
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Harmonic method), 39
__str__() (lammps_interface.lammps_potentials.ImproperPotential.Ring method), 39
__str__() (lammps_interface.lammps_potentials.ImproperPotential.UmbrellaAmber method), 40
__str__() (lammps_interface.lammps_potentials.PairPotential.Buck

method), 40
__str__() (*lammps_interface.lammps_potentials.PairPotential.BuckCoulLong method*), 40
__str__() (*lammps_interface.lammps_potentials.PairPotential.BuckLong method*), 41
__str__() (*lammps_interface.lammps_potentials.PairPotential.LjCharmmCoulLong method*), 41
__str__() (*lammps_interface.lammps_potentials.PairPotential.LjCut method*), 42
__str__() (*lammps_interface.lammps_potentials.PairPotential.LjCutCoulLong method*), 42
__str__() (*lammps_interface.lammps_potentials.PairPotential.LjCutTip4pLong method*), 42
__str__() (*lammps_interface.lammps_potentials.PairPotential.Table method*), 43
__weakref__ (*lammps_interface.CIFIO.CIF attribute*), 3
__weakref__ (*lammps_interface.ForceFields.ForceField attribute*), 8
__weakref__ (*lammps_interface.InputHandler.Options attribute*), 15
__weakref__ (*lammps_interface.create_cluster_v2.Cluster attribute*), 18
__weakref__ (*lammps_interface.create_cluster_v2.LammpsSimulation attribute*), 20
__weakref__ (*lammps_interface.lammps_main.LammpsSimulation attribute*), 21
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential attribute*), 28
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Charmm attribute*), 23
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Class2 attribute*), 24
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Class2AngleAngle attribute*), 23
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Class2AngleAngleMatrix attribute*), 24
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Cosine attribute*), 24
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.CosineDelta attribute*), 24
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.CosinePeriodic attribute*), 25
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.CosineShift attribute*), 25
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.CosineShiftExp attribute*), 25
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.CosineSquared attribute*), 26
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Dipole attribute*), 26
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Fourier attribute*), 26
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.FourierSimple attribute*), 27
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Harmonic attribute*), 27
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.MorseAngle attribute*), 28
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Quartic attribute*), 28
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Sdk attribute*), 28
__weakref__ (*lammps_interface.lammps_potentials.AnglePotential.Table attribute*), 28
__weakref__ (*lammps_interface.lammps_potentials.BondPotential attribute*), 32
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Class2 attribute*), 29
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Fene attribute*), 29
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.FeneExpand attribute*), 29
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Harmonic attribute*), 30
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.HarmonicShift attribute*), 30
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.HarmonicShiftCut attribute*), 30
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Morse attribute*), 31
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.NonLinear attribute*), 31
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Quartic attribute*), 31
__weakref__ (*lammps_interface.lammps_potentials.BondPotential.Table attribute*), 31
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential attribute*), 36
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Class2 attribute*), 32
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Class2AngleAngle attribute*), 34
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Class2AngleAngleMatrix attribute*), 32
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Class2BondAngle attribute*), 33
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Class2BondBond attribute*), 33
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Cosine attribute*), 34
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.CosineShiftExp attribute*), 34
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.CosineSquared attribute*), 34
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Fourier attribute*), 35
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Harmonic*

attribute), 35
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.MultiHarmonic attribute*), 35
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Opls attribute*), 36
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Quadratic attribute*), 36
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.Table attribute*), 36
__weakref__ (*lammps_interface.lammps_potentials.DihedralPotential.nHarmonic attribute*), 37
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential attribute*), 40
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Class2 attribute*), 38
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Class2.AngleAngle attribute*), 37
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Cossq attribute*), 38
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Cvff attribute*), 38
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Fourier attribute*), 39
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Harmonic attribute*), 39
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Ring attribute*), 39
__weakref__ (*lammps_interface.lammps_potentials.ImproperPotential.Umbrella attribute*), 40
__weakref__ (*lammps_interface.lammps_potentials.PairPotential attribute*), 43
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.Buck attribute*), 40
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.BuckCoulLong attribute*), 41
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.HbondDreidingMorse attribute*), 41
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.LiCharmmCoulLong attribute*), 41
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.LiCut attribute*), 42
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.LiCutCoulLong attribute*), 42
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.LiCutTip4pLong attribute*), 42
__weakref__ (*lammps_interface.lammps_potentials.PairPotential.Table attribute*), 43
__weakref__ (*lammps_interface.structure_data.Cell attribute*), 43
__weakref__ (*lammps_interface.structure_data.MDMC_config attribute*), 45
_set_attr() (*lammps_interface.InputHandler.Options method*), 15
type() (*lammps_interface.Molecules.Molecule property*), 16

A

a() (*lammps_interface.structure_data.Cell property*), 43
add_atomic_node() (*lammps_interface.structure_data.MolecularGraph method*), 45
add_bond_edge() (*lammps_interface.structure_data.MolecularGraph method*), 45
add_co2_model() (*lammps_interface.lammps_main.LammpsSimulation method*), 21
add_data() (*lammps_interface.CIFIO.CIF method*), 3
add_distance_matrix() (*in module lammps_interface.mof_sbus*), 43
add_water_model() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
add_water_model() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
alpha() (*lammps_interface.structure_data.Cell property*), 43
angle_term() (*lammps_interface.ForceFields.BTW_FF method*), 4
angle_term() (*lammps_interface.ForceFields.Dreiding method*), 5
angle_term() (*lammps_interface.ForceFields.Dubbeldam method*), 6
angle_term() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
angle_term() (*lammps_interface.ForceFields.FMOFCu method*), 8
angle_term() (*lammps_interface.ForceFields.ForceField method*), 8
angle_term() (*lammps_interface.ForceFields.MOF_FF method*), 8
angle_term() (*lammps_interface.ForceFields.SPC_E method*), 10
angle_term() (*lammps_interface.ForceFields.TIP3P method*), 10
angle_term() (*lammps_interface.ForceFields.TIP4P method*), 11
angle_term() (*lammps_interface.ForceFields.TIP5P method*), 12
angle_term() (*lammps_interface.ForceFields.UFF method*), 12
angle_term() (*lammps_interface.ForceFields.UFF4MOF method*), 13
angle_term() (*lammps_interface.ForceFields.UserFF method*), 14
AnglePotential (*class in lammps_interface.lammps_potentials*), 23
AnglePotential.Charmm (*class in lammps_interface.lammps_potentials*), 23

AnglePotential.Class2 (*class in lammps_interface.lammps_potentials*), 23
AnglePotential.Class2.BondAngle (*class in lammps_interface.lammps_potentials*), 23
AnglePotential.Class2.BondBond (*class in lammps_interface.lammps_potentials*), 23
AnglePotential.Cosine (*class in lammps_interface.lammps_potentials*), 24
AnglePotential.CosineDelta (*class in lammps_interface.lammps_potentials*), 24
AnglePotential.CosinePeriodic (*class in lammps_interface.lammps_potentials*), 25
AnglePotential.CosineShift (*class in lammps_interface.lammps_potentials*), 25
AnglePotential.CosineShiftExp (*class in lammps_interface.lammps_potentials*), 25
AnglePotential.CosineSquared (*class in lammps_interface.lammps_potentials*), 26
AnglePotential.Dipole (*class in lammps_interface.lammps_potentials*), 26
AnglePotential.Fourier (*class in lammps_interface.lammps_potentials*), 26
AnglePotential.FourierSimple (*class in lammps_interface.lammps_potentials*), 27
AnglePotential.Harmonic (*class in lammps_interface.lammps_potentials*), 27
AnglePotential.Quartic (*class in lammps_interface.lammps_potentials*), 27
AnglePotential.Sdk (*class in lammps_interface.lammps_potentials*), 28
AnglePotential.Table (*class in lammps_interface.lammps_potentials*), 28
approximate_positions() (*lammps_interface.Molecules.CO2 method*), 15
approximate_positions() (*lammps_interface.Molecules.Water method*), 17
assign_force_fields() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
assign_force_fields() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
assign_molecule_ids() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
assign_molecule_ids() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
atom_site_constraints() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_description() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_fract_x() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_fract_y() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_fract_z() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_fragment() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_label() (*lammps_interface.CIFIO.CIF static method*), 3
atom_site_type_symbol() (*lammps_interface.CIFIO.CIF static method*), 3
atom_type_partial_charge() (*lammps_interface.CIFIO.CIF static method*), 3
atomic_node_sanity_check() (*lammps_interface.structure_data.MolecularGraph method*), 45

B

b() (*lammps_interface.structure_data.Cell property*), 43
beta() (*lammps_interface.structure_data.Cell property*), 44
bond_term() (*lammps_interface.ForceFields.BTW_FF method*), 4
bond_term() (*lammps_interface.ForceFields.Dreiding method*), 5
bond_term() (*lammps_interface.ForceFields.Dubbeldam method*), 6
bond_term() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
bond_term() (*lammps_interface.ForceFields.FMOFCu method*), 8
bond_term() (*lammps_interface.ForceFields.ForceField method*), 8
bond_term() (*lammps_interface.ForceFields.MOF_FF method*), 9
bond_term() (*lammps_interface.ForceFields.SPC_E method*), 10
bond_term() (*lammps_interface.ForceFields.TIP3P method*), 10
bond_term() (*lammps_interface.ForceFields.TIP4P method*), 11
bond_term() (*lammps_interface.ForceFields.TIP5P method*), 12
bond_term() (*lammps_interface.ForceFields.UFF method*), 13

bond_term() (*lammps_interface.ForceFields.UFF4MOF method*), 13
bond_term() (*lammps_interface.ForceFields.UserFF method*), 14
BondPotential (*class in lammps_interface.lammps_potentials*), 28
BondPotential.Class2 (*class in lammps_interface.lammps_potentials*), 28
BondPotential.Fene (*class in lammps_interface.lammps_potentials*), 29
BondPotential.FeneExpand (*class in lammps_interface.lammps_potentials*), 29
BondPotential.Harmonic (*class in lammps_interface.lammps_potentials*), 29
BondPotential.HarmonicShift (*class in lammps_interface.lammps_potentials*), 30
BondPotential.HarmonicShiftCut (*class in lammps_interface.lammps_potentials*), 30
BondPotential.Morse (*class in lammps_interface.lammps_potentials*), 30
BondPotential.NonLinear (*class in lammps_interface.lammps_potentials*), 31
BondPotential.Quartic (*class in lammps_interface.lammps_potentials*), 31
BondPotential.Table (*class in lammps_interface.lammps_potentials*), 31
BTW_FF (*class in lammps_interface.ForceFields*), 4
build_supercell() (*lammps_interface.structure_data.MolecularGraph method*), 46

C

c() (*lammps_interface.structure_data.Cell property*), 44
cap_1D_organic() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cap_3D_organic() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cap_by_material() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cap_primary_cluster() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cap_zeolite() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cap_zeolite_v2() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cart_dist() (*lammps_interface.create_cluster_v2.Cluster method*), 18
ccdc_geom_bond_type() (*lammps_interface.CIFIO.CIF static method*), 3
Cell (*class in lammps_interface.structure_data*), 43
cell() (*lammps_interface.structure_data.Cell property*), 44
cell_angle_alpha() (*lammps_interface.CIFIO.CIF static method*), 4
cell_angle_beta() (*lammps_interface.CIFIO.CIF static method*), 4
cell_angle_gamma() (*lammps_interface.CIFIO.CIF static method*), 4
cell_length_a() (*lammps_interface.CIFIO.CIF static method*), 4
cell_length_b() (*lammps_interface.CIFIO.CIF static method*), 4
cell_length_c() (*lammps_interface.CIFIO.CIF static method*), 4
CIF (*class in lammps_interface.CIFIO*), 3
clean() (*in module lammps_interface.structure_data*), 48
Cluster (*class in lammps_interface.create_cluster_v2*), 17
CO2 (*class in lammps_interface.Molecules*), 15
compute_all_angles() (*lammps_interface.Molecules.Molecule method*), 16
compute_angle_between() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_angle_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_angles() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_atomic_pair_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_bond_image_flag() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_bond_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_bond_typing() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_bonding() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_cartesian_coordinates() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_cluster_box_size() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20

compute_cluster_in_disgraph() (*lammps_interface.create_cluster_v2.Cluster method*), 18
compute_cluster_in_tree() (*lammps_interface.create_cluster_v2.Cluster method*), 18
compute_dihedral_between() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_dihedral_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_dihedrals() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_force_field_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_force_field_terms() (*lammps_interface.ForceFields.UserFF method*), 14
compute_improper_dihedrals() (*lammps_interface.structure_data.MolecularGraph method*), 46
compute_improper_terms() (*lammps_interface.ForceFields.ForceField method*), 8
compute_init_typing() (*lammps_interface.structure_data.MolecularGraph method*), 47
compute_midpoint_vector() (*lammps_interface.Molecules.Water method*), 17
compute_min_img_distances() (*lammps_interface.structure_data.MolecularGraph method*), 47
compute_molecules() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
compute_molecules() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
compute_orthogonal_vector() (*lammps_interface.Molecules.Water method*), 17
compute_primary_cluster() (*lammps_interface.create_cluster_v2.Cluster method*), 18
compute_required_caps() (*lammps_interface.create_cluster_v2.Cluster method*), 18
compute_simulation_size() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
compute_simulation_size() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
compute_topology_information() (*lammps_interface.structure_data.MolecularGraph method*), 47
construct_data_file() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
construct_data_file() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
construct_input_file() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
construct_input_file() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
coplanar() (*lammps_interface.structure_data.MolecularGraph method*), 47
correspondence_graph() (*lammps_interface.structure_data.MolecularGraph method*), 47
count_angles() (*lammps_interface.create_cluster_v2.LammpsSimulat method*), 20
count_angles() (*lammps_interface.structure_data.MolecularGraph method*), 47
count_dihedrals() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
count_dihedrals() (*lammps_interface.structure_data.MolecularGraph method*), 47
count_impropers() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
count_impropers() (*lammps_interface.structure_data.MolecularGraph method*), 47
create_cluster_around_point() (*lammps_interface.create_cluster_v2.Cluster method*), 18
create_cluster_around_point_v2() (*lammps_interface.create_cluster_v2.Cluster method*), 18
create_cluster_around_point_v3() (*lammps_interface.create_cluster_v2.Cluster method*), 18
crystal_system() (*lammps_interface.structure_data.Cell property*), 44
cut_cappable_bonds() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cut_molecule() (*lammps_interface.create_cluster_v2.LammpsSimulat*

method), 20
cut_molecule() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
cxtd_comp_cap() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_continuous() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_convert_to_orig() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_from_undirected() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_num_keep() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_secondary_graph() (*lammps_interface.create_cluster_v2.Cluster method*), 18
cxtd_comp_to_keep() (*lammps_interface.create_cluster_v2.Cluster method*), 18

D

debug_edges_to_cut() (*lammps_interface.create_cluster_v2.Cluster method*), 19
define_styles() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
define_styles() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
del_parenth() (*in module lammps_interface.structure_data*), 48
detect_clusters() (*lammps_interface.structure_data.MolecularGraph method*), 47
detect_ff_terms() (*lammps_interface.ForceFields.BTW_FF method*), 5
detect_ff_terms() (*lammps_interface.ForceFields.Dreiding method*), 5
detect_ff_terms() (*lammps_interface.ForceFields.Dubbeldam method*), 6
detect_ff_terms() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
detect_ff_terms() (*lammps_interface.ForceFields.FMOFCu method*), 8
detect_ff_terms() (*lammps_interface.ForceFields.MOF_FF method*), 9
detect_ff_terms() (*lammps_interface.ForceFields.SPC_E method*), 10
detect_ff_terms() (*lammps_interface.ForceFields.TIP3P method*), 11
detect_ff_terms() (*lammps_interface.ForceFields.TIP4P method*), 11
detect_ff_terms() (*lammps_interface.ForceFields.TIP5P method*), 12
detect_ff_terms() (*lammps_interface.ForceFields.UFF method*), 13
detect_ff_terms() (*lammps_interface.ForceFields.UFF4MOF method*), 13
dihedral_term() (*lammps_interface.ForceFields.BTW_FF method*), 5
dihedral_term() (*lammps_interface.ForceFields.Dreiding method*), 5
dihedral_term() (*lammps_interface.ForceFields.Dubbeldam method*), 6
dihedral_term() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
dihedral_term() (*lammps_interface.ForceFields.FMOFCu method*), 8
dihedral_term() (*lammps_interface.ForceFields.ForceField method*), 8
dihedral_term() (*lammps_interface.ForceFields.MOF_FF method*), 9
dihedral_term() (*lammps_interface.ForceFields.SPC_E method*), 10
dihedral_term() (*lammps_interface.ForceFields.TIP3P method*), 11
dihedral_term() (*lammps_interface.ForceFields.TIP4P method*), 11
dihedral_term() (*lammps_interface.ForceFields.TIP5P method*), 12
dihedral_term() (*lammps_interface.ForceFields.UFF method*), 13
dihedral_term() (*lammps_interface.ForceFields.UFF4MOF method*), 14
dihedral_term() (*lammps_interface.ForceFields.UserFF method*), 14
DihedralPotential (*class in lammps_interface.lammps_potentials*), 32
DihedralPotential.Charmm (*class in lammps_interface.lammps_potentials*), 32
DihedralPotential.Class2 (*class in lammps_interface.lammps_potentials*), 32
DihedralPotential.Class2.AngleAngleTorsion

(*class in lammps_interface.lammps_potentials*), 32
DihedralPotential.Class2.AngleTorsion (*class in lammps_interface.lammps_potentials*), 32
DihedralPotential.Class2.BondBond13 (*class in lammps_interface.lammps_potentials*), 33
DihedralPotential.Class2.EndBondTorsion (*class in lammps_interface.lammps_potentials*), 33
DihedralPotential.Class2.MiddleBondTorsion (*class in lammps_interface.lammps_potentials*), 33
DihedralPotential.CosineShiftExp (*class in lammps_interface.lammps_potentials*), 34
DihedralPotential.Fourier (*class in lammps_interface.lammps_potentials*), 34
DihedralPotential.Harmonic (*class in lammps_interface.lammps_potentials*), 35
DihedralPotential.Helix (*class in lammps_interface.lammps_potentials*), 35
DihedralPotential.MultiHarmonic (*class in lammps_interface.lammps_potentials*), 35
DihedralPotential.nHarmonic (*class in lammps_interface.lammps_potentials*), 37
DihedralPotential.Opls (*class in lammps_interface.lammps_potentials*), 36
DihedralPotential.Quadratic (*class in lammps_interface.lammps_potentials*), 36
DihedralPotential.Table (*class in lammps_interface.lammps_potentials*), 36
disconnect_1D_building_blocks() (*lammps_interface.create_cluster_v2.Cluster method*), 19
disconnect_external_building_blocks() (*lammps_interface.create_cluster_v2.Cluster method*), 19
DOD (*lammps_interface.Molecules.TIP5P_Water attribute*), 16
Dreiding (*class in lammps_interface.ForceFields*), 5
Dubbeldam (*class in lammps_interface.ForceFields*), 6
dummy() (*lammps_interface.Molecules.TIP4P_Water property*), 16
dummy() (*lammps_interface.Molecules.TIP5P_Water property*), 16

E

edges_iter2() (*lammps_interface.structure_data.MolecularGraph method*), 47
EPM2_CO2 (*class in lammps_interface.ForceFields*), 7
EPM2_CO2 (*class in lammps_interface.Molecules*), 15

F

fixcount() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
fixcount() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
FMOFCu (*class in lammps_interface.ForceFields*), 7
ForceField (*class in lammps_interface.ForceFields*), 8
fractional() (*lammps_interface.structure_data.MolecularGraph method*), 47
from_CIF() (*in module lammps_interface.structure_data*), 48

G

gamma() (*lammps_interface.structure_data.Cell property*), 44
general_label() (*lammps_interface.CIFIO.CIF static method*), 4
geom_bond_atom_site_label_1() (*lammps_interface.CIFIO.CIF static method*), 4
geom_bond_atom_site_label_2() (*lammps_interface.CIFIO.CIF static method*), 4
geom_bond_distance() (*lammps_interface.CIFIO.CIF static method*), 4
geom_bond_site_symmetry_2() (*lammps_interface.CIFIO.CIF static method*), 4
get_BFS_tree() (*lammps_interface.create_cluster_v2.Cluster method*), 19
get_cell() (*lammps_interface.structure_data.Cell method*), 44
get_cell_inverse() (*lammps_interface.structure_data.Cell method*), 44
get_element_label() (*lammps_interface.CIFIO.CIF method*), 4
get_non_loop_block() (*lammps_interface.CIFIO.CIF method*), 4
get_nth_heirarcy_BFS_tree() (*lammps_interface.create_cluster_v2.Cluster method*), 19
get_params() (*lammps_interface.structure_data.Cell method*), 44
get_start_and_kept_nodes() (*lammps_interface.create_cluster_v2.Cluster method*), 19
get_time() (*in module lammps_interface.CIFIO*), 4
get_time() (*lammps_interface.CIFIO.CIF method*), 4

git_revision_hash() (*in module lammps_interface.InputHandler*), 15
groups() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
groups() (*lammps_interface.lammps_main.LammpsSimulation method*), 22

H

H_coord() (*lammps_interface.Molecules.Water property*), 17
hbond_pot() (*lammps_interface.ForceFields.Dreiding method*), 5
HOH (*lammps_interface.Molecules.TIP4P_Water attribute*), 16
HOH (*lammps_interface.Molecules.TIP5P_Water attribute*), 16

I

identify_1D_building_blocks() (*lammps_interface.create_cluster_v2.Cluster method*), 19
identify_all_truncations() (*lammps_interface.create_cluster_v2.Cluster method*), 19
identify_mat_type() (*lammps_interface.create_cluster_v2.Cluster method*), 19
img_offset() (*lammps_interface.structure_data.MolecularGraph method*), 47
improper_term() (*lammps_interface.ForceFields.BTW_FF method*), 5
improper_term() (*lammps_interface.ForceFields.Dreiding method*), 6
improper_term() (*lammps_interface.ForceFields.Dubbeldam method*), 7
improper_term() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
improper_term() (*lammps_interface.ForceFields.FMOFCu method*), 8
improper_term() (*lammps_interface.ForceFields.ForceField method*), 8
improper_term() (*lammps_interface.ForceFields.MOF_FF method*), 9
improper_term() (*lammps_interface.ForceFields.SPC_E method*), 10
improper_term() (*lammps_interface.ForceFields.TIP3P method*), 11
improper_term() (*lammps_interface.ForceFields.TIP4P method*), 11
improper_term() (*lammps_interface.ForceFields.TIP5P method*), 12
improper_term() (*lammps_interface.ForceFields.UFF method*), 13
improper_term() (*lammps_interface.ForceFields.UFF4MOF method*), 14
improper_term() (*lammps_interface.ForceFields.UserFF method*), 14
ImproperPotential (*class in lammps_interface.lammps_potentials*), 37
ImproperPotential.Class2 (*class in lammps_interface.lammps_potentials*), 37
ImproperPotential.Class2.AngleAngle (*class in lammps_interface.lammps_potentials*), 37
ImproperPotential.Cossq (*class in lammps_interface.lammps_potentials*), 38
ImproperPotential.Cvff (*class in lammps_interface.lammps_potentials*), 38
ImproperPotential.Fourier (*class in lammps_interface.lammps_potentials*), 38
ImproperPotential.Harmonic (*class in lammps_interface.lammps_potentials*), 39
ImproperPotential.Ring (*class in lammps_interface.lammps_potentials*), 39
ImproperPotential.Umbrella (*class in lammps_interface.lammps_potentials*), 39
in_cell() (*lammps_interface.structure_data.MolecularGraph method*), 47
increment_graph_sizes() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
increment_graph_sizes() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
insert_block_order() (*lammps_interface.CIFIO.CIF method*), 4
insert_graph() (*lammps_interface.ForceFields.FMOFCu method*), 8
inverse() (*lammps_interface.structure_data.Cell property*), 44
iterative_BFS_tree_structure() (*lammps_interface.create_cluster_v2.Cluster method*), 19

L

label() (*lammps_interface.CIFIO.CIF static method*), 4
lammps_interface (*module*), 49
lammps_interface.atomic (*module*), 17
lammps_interface.BTW (*module*), 3
lammps_interface.ccdc (*module*), 17
lammps_interface.CIFIO (*module*), 3
lammps_interface.create_cluster_v2 (*module*), 17
lammps_interface.dreiding (*module*), 21
lammps_interface.Dubbeldam (*module*), 4

lammps_interface.ForceFields (*module*), 4
lammps_interface.gas_models (*module*), 21
lammps_interface.generic_raspa (*module*), 21
lammps_interface.InputHandler (*module*), 15
lammps_interface.lammps_main (*module*), 21
lammps_interface.lammps_potentials (*module*), 23
lammps_interface.mof_sbus (*module*), 43
lammps_interface.MOFFF (*module*), 15
lammps_interface.Molecules (*module*), 15
lammps_interface.structure_data (*module*), 43
lammps_interface.uff (*module*), 48
lammps_interface.uff4mof (*module*), 48
lammps_interface.uff_nonbonded (*module*), 49
lammps_interface.water_models (*module*), 49
LammpsSimulation (*class in lammps_interface.create_cluster_v2*), 20
LammpsSimulation (*class in lammps_interface.lammps_main*), 21
lx() (*lammps_interface.structure_data.Cell property*), 44
ly() (*lammps_interface.structure_data.Cell property*), 44
lz() (*lammps_interface.structure_data.Cell property*), 44

M

main() (*in module lammps_interface.create_cluster_v2*), 21
main() (*in module lammps_interface.lammps_main*), 22
map_pair_unique_bond() (*lammps_interface.ForceFields.OverwriteFF method*), 9
map_pair_unique_bond() (*lammps_interface.ForceFields.UserFF method*), 14
map_quadruplet_unique_dihedral() (*lammps_interface.ForceFields.OverwriteFF method*), 9
map_quadruplet_unique_dihedral() (*lammps_interface.ForceFields.UserFF method*), 14
map_quadruplet_unique_improper() (*lammps_interface.ForceFields.OverwriteFF method*), 9
map_quadruplet_unique_improper() (*lammps_interface.ForceFields.UserFF method*), 14
map_triplet_unique_angle() (*lammps_interface.ForceFields.OverwriteFF method*), 9
map_triplet_unique_angle() (*lammps_interface.ForceFields.UserFF method*), 14
map_user_to_unique_atom() (*lammps_interface.ForceFields.OverwriteFF method*), 9
map_user_to_unique_atom() (*lammps_interface.ForceFields.UserFF method*), 14
MDMC_config (*class in lammps_interface.structure_data*), 45
merge_graphs() (*lammps_interface.create_cluster_v2.LammpsSimulat method*), 20
merge_graphs() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
min_img() (*lammps_interface.structure_data.MolecularGraph method*), 47
min_img_distance() (*lammps_interface.structure_data.MolecularGraph method*), 47
minimum_supercell() (*lammps_interface.structure_data.Cell method*), 44
minimum_width() (*lammps_interface.structure_data.Cell property*), 44
mod_to_UC() (*lammps_interface.structure_data.Cell method*), 44
modify_structure_w_hydrogens() (*lammps_interface.create_cluster_v2.Cluster method*), 19
MOF_FF (*class in lammps_interface.ForceFields*), 8
MolecularGraph (*class in lammps_interface.structure_data*), 45
Molecule (*class in lammps_interface.Molecules*), 16
molecule_template() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
molecule_template() (*lammps_interface.lammps_main.LammpsSimulation method*), 22

N

node_dict_factory (*lammps_interface.structure_data.MolecularGraph attribute*), 47
nodes_iter2() (*lammps_interface.structure_data.MolecularGraph method*), 47
nodes_that_DNE_in_origraph() (*lammps_interface.create_cluster_v2.Cluster method*), 19
nodes_w_2plus_parents() (*lammps_interface.create_cluster_v2.Cluster method*), 19

O

O_coord() (*lammps_interface.Molecules.CO2 property*), 15
Options (*class in lammps_interface.InputHandler*), 15
orthogonal_transformation() (*lammps_interface.structure_data.Cell method*), 44
overwrite_force_field_terms() (*lammps_interface.ForceFields.UserFF method*), 14
OverwriteFF (*class in lammps_interface.ForceFields*), 9

P

pair_coul_term() (*lammps_interface.ForceFields.MOF_FF method*), 9
pair_terms() (*lammps_interface.ForceFields.BTW_FF method*), 5
pair_terms() (*lammps_interface.ForceFields.Dreiding method*), 6
pair_terms() (*lammps_interface.ForceFields.Dubbeldam method*), 7
pair_terms() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
pair_terms() (*lammps_interface.ForceFields.FMOFCu method*), 8
pair_terms() (*lammps_interface.ForceFields.MOF_FF method*), 9
pair_terms() (*lammps_interface.ForceFields.SPC_E method*), 10
pair_terms() (*lammps_interface.ForceFields.TIP3P method*), 11
pair_terms() (*lammps_interface.ForceFields.TIP4P method*), 11
pair_terms() (*lammps_interface.ForceFields.TIP5P method*), 12
pair_terms() (*lammps_interface.ForceFields.UFF method*), 13
pair_terms() (*lammps_interface.ForceFields.UFF4MOF method*), 14
PairPotential (*class in lammps_interface.lammps_potentials*), 40
PairPotential.Buck (*class in lammps_interface.lammps_potentials*), 40
PairPotential.BuckCoulLong (*class in lammps_interface.lammps_potentials*), 40
PairPotential.HbondDreidingMorse (*class in lammps_interface.lammps_potentials*), 41
PairPotential.LjCharmmCoulLong (*class in lammps_interface.lammps_potentials*), 41
PairPotential.LjCut (*class in lammps_interface.lammps_potentials*), 41
PairPotential.LjCutCoulLong (*class in lammps_interface.lammps_potentials*), 42
PairPotential.LjCutTip4pLong (*class in lammps_interface.lammps_potentials*), 42
PairPotential.Table (*class in lammps_interface.lammps_potentials*), 42
params() (*lammps_interface.structure_data.Cell property*), 44
parse_sym_flag_for_directionality() (*lammps_interface.create_cluster_v2.Cluster method*), 19
parse_user_input() (*lammps_interface.ForceFields.OverwriteFF method*), 10
parse_user_input() (*lammps_interface.ForceFields.UserFF method*), 14
preliminary_truncate_BFS_tree() (*lammps_interface.create_cluster_v2.Cluster method*), 19

R

RCO (*lammps_interface.Molecules.EPM2_CO2 attribute*), 15
Rdum (*lammps_interface.Molecules.TIP4P_Water attribute*), 16
Rdum (*lammps_interface.Molecules.TIP5P_Water attribute*), 16
read() (*lammps_interface.CIFIO.CIF method*), 4
read_xyz_center() (*in module lammps_interface.create_cluster_v2*), 21
recurse_bonds_to_end() (*lammps_interface.structure_data.MolecularGraph method*), 47
recurse_linear_chains() (*lammps_interface.structure_data.MolecularGraph method*), 47
redefine_lattice() (*lammps_interface.structure_data.MolecularGraph method*), 47
reorder_labels() (*lammps_interface.structure_data.MolecularGraph method*), 47
ROH (*lammps_interface.Molecules.TIP4P_Water attribute*), 16
ROH (*lammps_interface.Molecules.TIP5P_Water attribute*), 16
rotation_from_vectors() (*lammps_interface.Molecules.Molecule method*), 16
rotation_matrix() (*lammps_interface.Molecules.Molecule method*), 16
run_command_line_options() (*lammps_interface.InputHandler.Options method*), 15

S

set_cell() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
set_cell() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
set_cell() (*lammps_interface.structure_data.Cell method*), 44
set_graph() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
set_graph() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
set_MDMC_config() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 20
set_MDMC_config() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
set_params() (*lammps_interface.structure_data.Cell method*), 44
show() (*lammps_interface.structure_data.MolecularGraph method*), 48
sorted_edge_list() (*lammps_interface.structure_data.MolecularGraph method*), 48
sorted_node_list() (*lammps_interface.structure_data.MolecularGraph method*), 48
SPC_E (*class in lammps_interface.ForceFields*), 10
special_commands() (*lammps_interface.ForceFields.BTW_FF method*), 5
special_commands() (*lammps_interface.ForceFields.Dreiding method*), 6
special_commands() (*lammps_interface.ForceFields.Dubbeldam method*), 7
special_commands() (*lammps_interface.ForceFields.EPM2_CO2 method*), 7
special_commands() (*lammps_interface.ForceFields.FMOFCu method*), 8
special_commands() (*lammps_interface.ForceFields.MOF_FF method*), 9
special_commands() (*lammps_interface.ForceFields.SPC_E method*), 10
special_commands() (*lammps_interface.ForceFields.TIP3P method*), 11
special_commands() (*lammps_interface.ForceFields.TIP4P method*), 11
special_commands() (*lammps_interface.ForceFields.TIP5P method*), 12
special_commands() (*lammps_interface.ForceFields.UFF method*), 13
special_commands() (*lammps_interface.ForceFields.UFF4MOF method*), 14
split_graph() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
split_graph() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
store_original_size() (*lammps_interface.structure_data.MolecularGraph method*), 48
str() (*lammps_interface.Molecules.Molecule method*), 16
subgraph() (*lammps_interface.structure_data.MolecularGraph method*), 48

T

TIP3P (*class in lammps_interface.ForceFields*), 10
TIP4P (*class in lammps_interface.ForceFields*), 11
TIP4P_Water (*class in lammps_interface.Molecules*), 16
TIP5P (*class in lammps_interface.ForceFields*), 11
TIP5P_Water (*class in lammps_interface.Molecules*), 16
truncate_all() (*lammps_interface.create_cluster_v2.Cluster method*), 19
truncation_criteria() (*lammps_interface.create_cluster_v2.Cluster method*), 19

U

UFF (*class in lammps_interface.ForceFields*), 12
UFF4MOF (*class in lammps_interface.ForceFields*), 13
uff_angle_type() (*lammps_interface.ForceFields.UFF method*), 13
uff_angle_type() (*lammps_interface.ForceFields.UFF4MOF method*), 14
unique_angles() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_angles() (*lammps_interface.ForceFields.UserFF method*), 14
unique_angles() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unique_atoms() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_atoms() (*lammps_interface.ForceFields.UserFF method*), 14

unique_atoms() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unique_bonds() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_bonds() (*lammps_interface.ForceFields.UserFF method*), 14
unique_bonds() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unique_dihedrals() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_dihedrals() (*lammps_interface.ForceFields.UserFF method*), 14
unique_dihedrals() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unique_impropers() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_impropers() (*lammps_interface.ForceFields.UserFF method*), 14
unique_impropers() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unique_pair_terms() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
unique_pair_terms() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
unwrap_node_coordinates() (*lammps_interface.structure_data.MolecularGraph method*), 48
update_num_keep() (*lammps_interface.create_cluster_v2.Cluster method*), 19
update_supercell() (*lammps_interface.structure_data.Cell method*), 44
update_symflag() (*lammps_interface.structure_data.MolecularGraph method*), 48
UserFF (*class in lammps_interface.ForceFields*), 14

V

van_der_waals_pairs() (*lammps_interface.ForceFields.UserFF method*), 15
visualize_n_levels_of_tree() (*lammps_interface.create_cluster_v2.Cluster method*), 19
volume() (*lammps_interface.structure_data.Cell property*), 44

W

Water (*class in lammps_interface.Molecules*), 17
write_cif() (*in module lammps_interface.structure_data*), 48
write_cluster_to_host_xyz_v2() (*lammps_interface.create_cluster_v2.Cluster method*), 19
write_cluster_to_xyz() (*lammps_interface.create_cluster_v2.Cluster method*), 19
write_cluster_to_xyz_host_guest_v2() (*lammps_interface.create_cluster_v2.Cluster method*), 20
write_cutoff() (*lammps_interface.create_cluster_v2.Cluster method*), 20
write_lammps_files() (*lammps_interface.create_cluster_v2.LammpsSimulation method*), 21
write_lammps_files() (*lammps_interface.lammps_main.LammpsSimulation method*), 22
write_LSDALTON() (*lammps_interface.create_cluster_v2.Cluster method*), 19
write_map_from_cluster_xyz_to_unique_types() (*lammps_interface.create_cluster_v2.Cluster method*), 20
write_map_from_MOLECULE_to_unique_types() (*lammps_interface.create_cluster_v2.Cluster method*), 20
write_missing_uniques() (*lammps_interface.ForceFields.OverwriteFF method*), 10
write_missing_uniques() (*lammps_interface.ForceFields.UserFF method*), 15
write_PDB() (*in module lammps_interface.structure_data*), 48
write_RASPA_CIF() (*in module lammps_interface.structure_data*), 48
write_RASPA_sim_files() (*in module lammps_interface.structure_data*), 48

X

xy() (*lammps_interface.structure_data.Cell property*), 44
xz() (*lammps_interface.structure_data.Cell property*), 45

Y

yz() (*lammps_interface.structure_data.Cell property*), 45