Squid Documentation

Release 2.0.0

Henry Herbol

Jul 25, 2019

CONTENTS

1 Squid **3**
- 1.1 Installing . . . 3
- 1.2 Contributing . . . 3
- 1.3 Documentation . . . 3

2 Overview **5**
- 2.1 calcs . . . 5
- 2.2 files . . . 5
- 2.3 forcefields . . . 6
- 2.4 g09 . . . 7
- 2.5 geometry . . . 7
- 2.6 jdftx . . . 8
- 2.7 jobs . . . 8
- 2.8 lammps . . . 9
- 2.9 maths . . . 10
- 2.10 optimizers . . . 10
- 2.11 orca . . . 11
- 2.12 post_process . . . 11
- 2.13 qe . . . 12
- 2.14 structures . . . 12
- 2.15 utils . . . 13

3 Codebase **15**
- 3.1 calcs . . . 15
  - 3.1.1 ANEB . . . 15
  - 3.1.2 NEB . . . 19
- 3.2 files . . . 23
  - 3.2.1 cml_io . . . 23
  - 3.2.2 misc . . . 24
  - 3.2.3 xyz_io . . . 24
- 3.3 forcefields . . . 25
  - 3.3.1 connectors . . . 25
  - 3.3.2 coulomb . . . 29
  - 3.3.3 helper . . . 31
  - 3.3.4 lj . . . 32
  - 3.3.5 morse . . . 35
  - 3.3.6 opls . . . 38
  - 3.3.7 parameters . . . 38
  - 3.3.8 smrff . . . 44
  - 3.3.9 tersoff . . . 45

3.4 g09 . . . 50
3.5 geometry . . . 50
3.5.1 misc . . . 50
3.5.2 packmol . . . 51
3.5.3 spatial . . . 52
3.5.4 transform . . . 54
3.6 jdftx . . . 56
3.7 jobs . . . 56
3.7.1 container . . . 56
3.7.2 nbs . . . 56
3.7.3 queue_manager . . . 58
3.7.4 slurm . . . 60
3.7.5 submission . . . 61
3.8 lammps . . . 62
3.8.1 io.data . . . 62
3.8.2 io.dump . . . 63
3.8.3 io.thermo . . . 63
3.8.4 job . . . 63
3.8.5 parser . . . 64
3.9 maths . . . 66
3.9.1 lhs . . . 66
3.10 optimizers . . . 66
3.10.1 bfgs . . . 66
3.10.2 conjugate_gradient . . . 68
3.10.3 fire . . . 69
3.10.4 lbfgs . . . 69
3.10.5 quick_min . . . 71
3.10.6 steepest_descent . . . 71
3.11 orca . . . 72
3.11.1 io . . . 72
3.11.2 job . . . 73
3.11.3 mep . . . 75
3.11.4 post_process . . . 76
3.11.5 utils . . . 77
3.12 post_process . . . 77
3.12.1 debyer . . . 77
3.12.2 ovito . . . 78
3.12.3 vmd . . . 79
3.13 qe . . . 80
3.14 structures . . . 80
3.14.1 atom . . . 80
3.14.2 molecule . . . 82
3.14.3 results . . . 84
3.14.4 system . . . 85
3.14.5 topology . . . 88
3.15 utils . . . 89
3.15.1 cast . . . 89
3.15.2 print_helper . . . 90
3.15.3 units . . . 92

4 Console Scripts **95**
4.1 chkDFT . . . 95
4.2 scanDFT . . . 95
4.3 procrustes . . . 97

4.4 pysub . . . 97

5 Examples **99**
5.1 Nudged Elastic Band Demo . . . 99
5.2 Molecular Orbital Visualization Demo . . . 103
5.3 DFT - Electrostatic Potential Mapped on Electron Density Post Processing . . . 105
5.4 Geometry - Smoothing out a Reaction Coordinate . . . 106
5.5 Molecular Dynamics Sovlent Box Equilibration . . . 110

6 Indices and tables **113**

Python Module Index **115**

Index **117**

Contents:

CHAPTER

ONE

SQUID

Squid is an open-source molecular simulation codebase developed by the Clancy Lab at the Johns Hopkins University. The codebase includes simplified Molecular Dynamics (MD) and Density Functional Theory (DFT) simulation submission, as well as other utilities such as file I/O and post-processing.

1.1 Installing

For most, the easiest way to install squid is to use pip install:

```
[user@local]~% pip install clancylab-squid
```

If you wish, you may also clone the repository though:

```
[user@local]~% cd ~; git clone https://github.com/ClancyLab/squid.git
```

1.2 Contributing

If you would like to be an active developer within the Clancy Group, please contact the project maintainer to be added as a collaborator on the project. Otherwise, you are welcome to submit pull requests as you see fit, and they will be addressed.

1.3 Documentation

Documentation is necessary, and the following steps MUST be followed during contribution of new code:

Setup

1. Download Sphinx. This can be done simply if you have pip installed via *pip install -U Sphinx*
2. Wherever you have *squid* installed, you want another folder called *squid-docs* (NOT as a subfolder of squid).

```
[user@local]~% cd ~; mkdir squid-docs; cd squid-docs; git clone -b gh-pages␣
↪git@github.com:clancylab/squid.git html
```

3. Forever more just ignore that directory (don't delete it though)

Adding Documentation

Documentation is done using ReStructuredText format docstrings, the Sphinx python package, and indices with autodoc extensions. To add more documentation, first add the file to be included in *docs/source/conf.py* under

os.path.abspath('example/dir/to/script.py'). Secondly, ensure that you have proper docstrings in the python file, and finally run *make full* to re-generate the documentation and commit it to your local branch, as well as the git *gh-pages* branch.

For anymore information on documentation, the tutorial follwed can be found here.

CHAPTER

TWO

OVERVIEW

2.1 calcs

The calcs module contains various calculations that can be seen as an automated task. Primarily, it currently holds two NEB class objects that handle running Nudged Elastic Band.

The first object, `squid.calcs.neb.NEB`, will run a standard NEB optimization. It allows for fixes such as the procrustes superimposition method and climbing image. The second object, `squid.calcs.aneb.ANEB`, handles the automated NEB approach, which will dynamically add in frames during the optimization. The idea of ANEB is that, in the end it should require less DFT calculations to complete.

Module Files:

- neb
- aneb

2.2 files

The files module handles file input and output. Currently, the following is supported:

- `squid.files.xyz_io.read_xyz()`
- `squid.files.xyz_io.write_xyz()`
- `squid.files.cml_io.read_cml()`
- `squid.files.cml_io.write_cml()`

Note - you can import any of these function directly from the files module as:

```
from squid import files

frames = files.read_xyz("demo.xyz")
```

Alternatively, some generators have been made to speed up the reading in of larger files:

- `squid.files.xyz_io.read_xyz_gen()`

When reading in xyz files of many frames, a list of lists holding `structures.atom.Atom` objects is returned. Otherwise, a single list of `structures.atom.Atom` objects is returned.

When reading in cml files, a list of `structures.molecule.Molecule` objects is returned.

Finally, additional functionality exists within the misc module:

- *squid.files.misc.is_exe()* - Determine if a file is an executable.
- *squid.files.misc.last_modified()* - Determine when a file was last modified.
- *squid.files.misc.which()* - Determine where a file is on a system.

Module Files:

- xyz_io
- cml_io
- misc

2.3 forcefields

To handle forcefields in Molecular Dynamics, the various components are subdivided into objects. These are then stored in an overarching `squid.forcefields.parameters.Parameters` object, which is the main interface a user should use.

Main user interface:

- `squid.forcefields.parameters.Parameters`

Subdivided objects:

- `squid.forcefields.connectors.HarmonicConnector` - A generic connector object.
- `squid.forcefields.connectors.Bond` - Derived from the HarmonicConnector, this handles Bonds.
- `squid.forcefields.connectors.Angle` - Derived from the HarmonicConnector, this handles Angles.
- `squid.forcefields.connectors.Dihedral` - Derived from the HarmonicConnector, this handles Dihedrals.

Supported Potentials:

- `squid.forcefields.coulomb.Coul` - An object to handle Coulombic information. This also holds other pertinent atomic information (element, mass, etc).
- `squid.forcefields.lj.LJ` - An object to handle the Lennard-Jones information.
- `squid.forcefields.morse.Morse` - An object to handle Morse information.
- `squid.forcefields.tersoff.Tersoff` - An object to handle Tersoff information.

Helper Code:

- `squid.forcefields.opls.parse_pfile()` - A function to parse the OPLS parameter file.
- *squid.forcefields.smrff.parse_pfile()* - A function to parse the SMRFF parameter file.

Module Files:

- coulomb
- lj
- morse
- tersoff
- opls

- smrff
- connectors
- helper
- parameters

2.4 g09

TODO

Module Files:

- TODO

2.5 geometry

The geometry module is broken down into different sections to handle atomic/molecular/system transformations/calculations.

The transform module holds functions that handle molecular transformations.

- `squid.geometry.transform.align_centroid()` - Align list of atoms to an ellipse along the x-axis.
- `squid.geometry.transform.interpolate()` - Linearly interpolate N frames between a given two frames.
- `squid.geometry.transform.perturbate()` - Perturbate atomic coordinates of a list of atoms.
- `squid.geometry.transform.procrustes()` - Propogate rotations along a list of atoms to minimize rigid rotation, and return the rotation matrices used.
- `squid.geometry.transform.smooth_xyz()` - Iteratively use procrustes and linear interpolation to smooth out a list of atomic coordinates.

Note, when using `squid.geometry.transform.procrustes()` the input frames are being changed! If this is not desired behaviour, and you solely wish for the rotation matrix, then pass in a copy of the frames.

The spatial module holds functions that handle understanding the spatial relationship between atoms/molecules.

- `squid.geometry.spatial.motion_per_frame()` - Get the inter-frame RMS motion per frame.
- `squid.geometry.spatial.mvee()` - Fit a volume to a list of atomic coordinates.
- `squid.geometry.spatial.orthogonal_procrustes()` - Find the rotation matrix that best fits one list of atomic coordinates onto another.
- `squid.geometry.spatial.random_rotation_matrix()` - Generate a random rotation matrix.
- `squid.geometry.spatial.rotation_matrix()` - Generate a rotation matrix based on angle and axis.

The packmol module handles the interface between Squid and packmol (http://m3g.iqm.unicamp.br/packmol/home.shtml). The main functionality here is simply calling `squid.geometry.packmol.packmol()` on a system object with a set of molecules.

The misc module holds functions that are not dependent on other squid modules, but can return useful information and simplify coding.

- *squid.geometry.misc.get_center_of_geometry()*
- *squid.geometry.misc.get_center_of_mass()*
- *squid.geometry.misc.rotate_atoms()*

Once again, all the above can be accessed directly from the geometry module, as shown in the following pseudo-code example here:

```
# NOTE THIS IS PSEUDO CODE AND WILL NOT WORK AS IS

from squid import geometry

mol1 = None
system_obj = None

geometry.packmol(system_obj, [mol1], density=1.0)
geometry.get_center_of_geometry(system_obj.atoms)
```

Module Files:

- misc
- packmol
- spatial
- transform

2.6 jdftx

TODO

Module Files:

- TODO

2.7 jobs

The jobs module handles submitting simulations/calculations to either a queueing system (ex. SLURM/NBS), or locally on a machine. This is done by storing a job into a job container, which will monitor it and allow the user to assess if simulations are still running or not. The job object is mainly used within squid, and is not normally required for the user to generate on their own.

The main interface with the job module is through the queue_manager module and the submission module; however, lower level access can be obtained through the container, nbs, slurm, and misc modules.

The queue_manager module holds the following:

- `squid.jobs.queue_manager.get_all_jobs()` - Get a list of all jobs submitted that are currently running or pending.
- `squid.jobs.queue_manager.get_available_queues()` - Get a list of the avaiable queue/partition names.
- `squid.jobs.queue_manager.get_pending_jobs()` - Get a list of all jobs submitted that are currently pending.
- `squid.jobs.queue_manager.get_queue_manager()` - Get the queue manager available on the system.
- `squid.jobs.queue_manager.get_running_jobs()` - Get a list of all jobs submitted that are currently running.
- `squid.jobs.queue_manager.Job()` - Get a Job object container depending on the queueing system used.

The submission module holds two function that handle submitting a job:

- `squid.jobs.submission.submit_job()` - Submit a script as a job.
- `squid.jobs.submission.pysub()` - Submit a python script as a job.

Module Files:

- container
- nbs
- queue_manager
- slurm
- submission

2.8 lammps

The lammps module allows squid to interface with the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) code. Due to the inherent flexibility of LAMMPS, the user is still required to write-up their own lammps input script so as to not obfuscate the science; however, tedious additional tasks can be done away with using squid.

Two main abilities exist within the lammps module: submitting simulations and parsing output. This is divided into the following:

- `squid.lammps.job.job()` - The main function that allows a user to submit a LAMMPS simulation.
- `squid.lammps.io.dump.read_dump()` - The main function that allows a user to robustly read in a LAMMPS dump file.
- `squid.lammps.io.dump.read_dump_gen()` - A generator for reading in a LAMMPS dump file, so as to improve speeds.
- `squid.lammps.io.data.write_lammps_data()` - A function to automate the writing of a LAMMPS data file.

Module Files:

- io.dump
- io.data

- io.thermo
- job
- parser

References:

- https://lammps.sandia.gov/
- www.cs.sandia.gov/~sjplimp/pizza.html

2.9 maths

The maths module handles additional mathematical calculations that do not pertain to atomic coordinates.

- *squid.maths.lhs.create_lhs()* - A function to generate Latin Hypercube Sampled values.

Module Files:

- lhs

References:

- https://pythonhosted.org/pyDOE/

2.10 optimizers

The optimizers module contains various functions aiding in optimization. Several of these approaches are founded on methods within scipy with minor alterations made here to aid in the internal use of NEB optimization. If you need to use an optimizer, we recommend going straight to Scipy and using their optimizers, as they will remain more up-to-date. These have injected features allowing for use with the internal squid NEB and ANEB calculations.

- *squid.optimizers.steepest_descent.steepest_descent()*
- *squid.optimizers.bfgs.bfgs()*
- *squid.optimizers.lbfgs.lbfgs()*
- *squid.optimizers.quick_min.quick_min()*
- *squid.optimizers.fire.fire()*
- *squid.optimizers.conjugate_gradient.conjugate_gradient()*

Module Files:

- steepest_descent
- bfgs
- lbfgs
- quick_min
- fire
- conjugate_gradient

2.11 orca

The orca module allows squid to interface with the orca DFT code.

- `squid.orca.job.job()` - Submit a simulation.
- `squid.orca.io.read()` - Read in all relevant information from an orca output simulation.
- `squid.orca.post_process.gbw_to_cube()` - Convert the output orca gbw file to a cube file for further processing.
- `squid.orca.post_process.mo_analysis()` - Automate the generation of molecular orbitals from an orca simulation, which will then be visualized using VMD.
- `squid.orca.post_process.pot_analysis()` - Automate the generation of an electrostatic potential mapped to the electron density surface from an orca simulation, which will then be visualized using VMD.

Module Files:

- io
- job
- mep
- post_process
- utils

References:

- https://sites.google.com/site/orcainputlibrary/dft

2.12 post_process

The post_process module holds functions that will aid in common post processing procedures. They further interface with external programs to visualize or simplify the process.

- `squid.post_process.debyer.get_pdf()` - Get a Pair Distribution Function (PDF) of a list of atomic coordinates. This is done using the debyer software (requires that debyer is installed).
- *`squid.post_process.vmd.plot_MO_from_cube()`* - Visualize molecular orbitals in VMD from a cube file.
- *`squid.post_process.vmd.plot_electrostatic_from_cube()`* - Visualize the electrostatic potential in VMD from a cube file.
- `squid.post_process.ovito.ovito_xyz_to_image()` - Automate the generation of an image of atomic coordinates using ovito.
- `squid.post_process.ovito.ovito_xyz_to_gif()` - Automate the generation of a gif of a sequence of atomic coordinates using ovito.

Module Files:

- debyer
- ovito
- vmd

References:

- https://debyer.readthedocs.io/en/latest/
- https://ovito.org/
- https://www.ks.uiuc.edu/Research/vmd/

2.13 qe

TODO

Module Files:

- TODO

2.14 structures

To handle atomic manipulation in python, we break down systems into the following components:

- `squid.structures.atom.Atom` - A single atom object.
- `squid.structures.topology.Connector` - A generic object to handle bonds, angles, and dihedrals.
- `squid.structures.molecule.Molecule` - A molecule object that stores atoms and all inter-atomic connections.
- `squid.structures.system.System` - A system object that holds a simulation environment. Consider this many molecules, and system dimensions for Molecular Dynamics.

For simplicity sake, when we generate a Molecule object based on atoms and bonds, all relevant angles and dihedrals are also generated and stored.

We also store objects to hold output simulation data:

- `squid.structures.results.DFT_out` - DFT specific output
- `squid.structures.results.sim_out` - More generic output

Module Files:

- atom
- molecule
- results
- system
- topology

2.15 utils

The utils module holds various utility functions that help squid internally; however, can be used externally as well.

The cast module holds functions to handle variable type assessment:

- *squid.utils.cast.is_array()* - Check if a variable is array like.
- *squid.utils.cast.check_vec()* - Check a vector for certain features.
- *squid.utils.cast.is_numeric()* - Check if a variable is numeric.
- *squid.utils.cast.assert_vec()* - Assert that a variable is array like with certain features.
- *squid.utils.cast.simplify_numerical_array()* - Simplify a sequence of numbers to a comma separated string with values within a range indicated using inclusive i-j.

The print_helper module holds functions to simplify string terminal output on Linux/Unix primarily:

- *squid.utils.print_helper.color_set()*
- *squid.utils.print_helper.strip_color()*
- *squid.utils.print_helper.spaced_print()*
- *squid.utils.print_helper.printProgressBar()*
- *squid.utils.print_helper.bytes2human()*

The units module holds functions to handle SI unit conversion:

- *squid.utils.units.convert_energy()*
- *squid.utils.units.convert_pressure()*
- *squid.utils.units.convert_dist()*
- *squid.utils.units.elem_i2s()*
- *squid.utils.units.elem_s2i()*
- *squid.utils.units.elem_weight()*
- *squid.utils.units.elem_sym_from_weight()*
- *squid.utils.units.convert()*

Module Files:

- cast
- print_helper
- units

CHAPTER

THREE

CODEBASE

3.1 calcs

3.1.1 ANEB

The Auto ANEB module simplifies the submission of Auto Nudged Elastic Band simulations.

NOTE! This module is still in a very rough beta. It has been hacked together from the NEB module and is being tested. Do not use this expecting a miracle.

Squid Auto Nudged Elastic Band package Currently supports g09 and orca Cite ANEB:

> http://aip.scitation.org/doi/full/10.1063/1.4961868 http://scitation.aip.org/content/aip/journal/jcp/113/22/10.1063/1.1323224

BFGS is the best method, cite: http://theory.cm.utexas.edu/henkelman/pubs/sheppard08_134106.pdf

Nudged Elastic Band. k for VASP is 5 eV/Angstrom, ie 0.1837 Hartree/Angstrom. Gtol of 1E-5 from scipy.bfgs package

The following code has been tested out to some moderate success for now:

```
new_opt_params = {'step_size': 1.0,
                  'step_size_adjustment': 0.5,
                  'max_step': 0.04,
                  'maxiter': 100,
                  'linesearch': None,
                  'accelerate': False,
                  'N_reset_hess': 10,
                  'max_steps_remembered': 5,
                  'fit_rigid': True,
                  'g_rms': units.convert("eV/Ang", "Ha/Ang", 0.001),
                  'g_max': units.convert("eV/Ang", "Ha/Ang", 0.03)}

new_auto_opt_params = {'step_size': 1.0,
                       'step_size_adjustment': 0.5,
                       'max_step': 0.04,
                       'maxiter': 20,
                       'linesearch': 'backtrack',
                       'accelerate': True,
                       'reset_step_size': 20,
                       'fit_rigid': True,
                       'g_rms': units.convert("eV/Ang", "Ha/Ang", 10.0),
                       'g_max': units.convert("eV/Ang", "Ha/Ang", 0.03)}
```

(continues on next page)

(continued from previous page)

```
nebs = aneb.ANEB("debug_auto", frames, "!HF-3c", fit_rigid=True,
                 opt='LBFGS',
                 new_opt_params=new_opt_params,
                 new_auto_opt_params=new_auto_opt_params,
                 ci_N=3,
                 ANEB_Nsim=5,
                 ANEB_Nmax=15)
```

- `g09_start_job()`
- `g09_results()`
- `orca_start_job()`
- `orca_results()`
- `ANEB`

---

class squid.calcs.aneb.**ANEB**(*name*, *states*, *theory*, *extra_section=''*, *initial_guess=None*, *spring_atoms=None*, *procs=1*, *queue=None*, *mem=2000*, *priority=None*, *disp=0*, *k=0.00367453*, *charge=0*, *fit_rigid=True*, *DFT='orca'*, *opt='LBFGS'*, *start_job=None*, *get_results=None*, *new_opt_params={}*, *new_auto_opt_params={}*, *callback=None*, *ci_ANEB=False*, *ci_N=5*, *ANEB_Nsim=5*, *ANEB_Nmax=15*, *add_by_energy=False*)

A method for determining the minimum energy pathway of a reaction using DFT. Note, this method was written for atomic orbital DFT codes; however, is potentially generalizable to other programs.

Parameters

name: ***str*** The name of the ANEB simulation to be run.

states: ***list, list,*** **`squid.structures.atom.Atom`** A list of frames, each frame being a list of atom structures. These frames represent your reaction coordinate.

theory: ***str*** The route line for your DFT simulation.

extra_section: ***str, optional*** Additional parameters for your DFT simulation.

initial_guess: ***list, str, optional*** TODO - List of strings specifying a previously run ANEB simulation, allowing restart capabilities.

spring_atoms: ***list, int, optional*** Specify which atoms will be represented by virutal springs in the ANEB calculations. Default includes all.

procs: ***int, optional*** The number of processors for your simulation.

queue: ***str, optional*** Which queue you wish your simulation to run on (queueing system dependent). When None, ANEB is run locally.

mem: ***float, optional*** Specify memory constraints (specific to your X_start_job method).

priority: ***int, optional*** Whether to submit a DFT simulation with some given priority or not.

disp: ***int, optional*** Specify for additional stdout information.

charge: ***int*** Charge of the system.

k: ***float, optional*** The spring constant for your ANEB simulation.

fit_rigid: ***bool, optional*** Whether you want to use procrustes to minimize motion between adjacent frames (thus minimizing error due to excessive virtal spring forces).

DFT: ***str, optional*** Specify if you wish to use the default X_start_job and X_results functions where X is either g09 or orca.

opt: ***str, optional*** Select which optimization method you wish to use from the following: LBFGS.

start_job: ***func, optional*** A function specifying how to submit your ANEB single point calculations. Needed if DFT is neither orca nor g09.

get_results: ***func, optional*** A function specifying how to read your ANEB single point calculations. Needed if DFT is neither orca nor g09.

new_opt_params: ***dict, optional*** Pass any additional parameters to the optimization algorithm. Note, these parameters are for the final calculation after frames have been added in.

new_auto_opt_params: ***dict, optional*** Pass any additional parameters to the optimization algorithm. Note, these parameters are for the iterative calculations, as frames are being added to the band.

callback: ***func, optional*** A function to be run after each each to calculate().

ci_ANEB: ***bool, optional*** Whether to use the climbing image variation of ANEB.

ci_N: ***int, optional*** How many iterations to wait in climbing image ANEB before selecting which image to be used.

ANEB_Nsim: ***int, optional*** The number of frames for an auto ANEB calculation. If an even number is chosen, the expansion happens around floor(ANEB_Nsim/2).

ANEB_Nmax: ***int, optional*** The maximum number of frames to build up to in the auto ANEB.

add_by_energy: ***bool, optional*** If the user wants to add frames by the largest dE instead of dR (motion per frame), then set this flag to True.

Returns

This `ANEB` object.

References

- Henkelman, G.; Jonsson, H. The Journal of Chemical Physics 2000, 113, 9978-9985.
- Jonsson, H.; Mills, G.; Jacobson, K. W. In Classical and Quantum Dynamics in Condensed Phase Simulations;
- Berne, B. J., Ciccotti, G., Coker, D. F., Eds.; World Scientific, 1998; Chapter 16, pp 385-404.
- Armijo, L. Pacific Journal of Mathematics 1966, 16.
- Sheppard, D.; Terrell, R.; Henkelman, G. The Journal of Chemical Physics 2008, 128.
- Henkelman, G.; Uberuaga, B. P.; Jonsson, H. Journal of Chemical Physics 2000, 113.
- Atomic Simulation Environment - https://wiki.fysik.dtu.dk/ase/
- Kolsbjerg, E. L.; Groves, M. N.; Hammer, B. The Journal of Chemical Physics 2016, 145.

align_coordinates(*r*, *B=None*, *H=None*, *return_matrix=False*)

Get a rotation matrix A that will remove rigid rotation from the new coordinates r. Further, if another vector needs rotating by the same matrix A, it should be passed in B and will be rotated. If a matrix also needs rotating, it can be passed as H and also be rotated.

Parameters

r: ***list, float*** 1D array of atomic coordinates to be rotated by procrustes matrix A.

B: ***list, list, float, optional*** A list of vectors that may also be rotated by the same matrix as *r*.

H: ***list, list, float, optional***

A matrix that should also be rotated via: H = R * H * R.T

return_matrix: ***bool, optional*** Whether to also return the rotation matrix used or not.

Returns

rotations: ***dict*** A dictionary holding 'A', the rotation matrix, 'r', the rotated new coordinates, 'B', a list of all other vectors that were rotated, and 'H', a rotated matrix.

`squid.calcs.aneb.`**`g09_results`**(*ANEB*, *step_to_use*, *i*, *state*)

A method for reading in the output of Gaussian09 single point calculations for ANEB calculations. This will both (a) assign forces to the atoms stored in state and (b) return the energy and atoms.

Parameters

ANEB: **`ANEB`** An ANEB container holding the main ANEB simulation

step_to_use: ***int*** Which iteration in the ANEB sequence the output to be read in is on.

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **`squid.structures.atom.Atom`** A list of atoms describing the image on the frame associated with index *i*.

Returns

new_energy: ***float*** The energy of the system in Hartree (Ha).

new_atoms: ***list,*** **`squid.structures.atom.Atom`** A list of atoms with the forces attached in units of Hartree per Angstrom (Ha/Ang).

`squid.calcs.aneb.`**`g09_start_job`**(*ANEB*, *i*, *state*, *charge*, *procs*, *queue*, *initial_guess*, *extra_section*, *mem*, *priority*)

A method for submitting a single point calculation using Gaussian09 for ANEB calculations.

Parameters

ANEB: **`ANEB`** An ANEB container holding the main ANEB simulation

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **`squid.structures.atom.Atom`** A list of atoms describing the image on the frame associated with index *i*.

charge: ***int*** Charge of the system.

procs: ***int*** The number of processors to use during calculations.

queue: ***str*** Which queue to submit the simulation to (this is queueing system dependent).

initial_guess: ***str*** The name of a previous simulation for which we can read in a hessian.

extra_section: ***str*** Extra settings for this DFT method.

mem: ***int*** How many Mega Words (MW) you wish to have as dynamic memory.

priority: ***int*** Whether to submit the job with a given priority (NBS). Not setup for this function yet.

Returns

g09_job: **`squid.jobs.container.JobObject`** A job container holding the g09 simulation.

squid.calcs.aneb.**orca_results**(*ANEB*, *step_to_use*, *i*, *state*)

A method for reading in the output of Orca single point calculations for ANEB calculations. This will both (a) assign forces to the atoms stored in state and (b) return the energy and atoms.

Parameters

ANEB: **ANEB** An ANEB container holding the main ANEB simulation

step_to_use: ***int*** Which iteration in the ANEB sequence the output to be read in is on.

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **squid.structures.atom.Atom** A list of atoms describing the image on the frame associated with index *i*.

Returns

new_energy: ***float*** The energy of the system in Hartree (Ha).

new_atoms: ***list,*** **squid.structures.atom.Atom** A list of atoms with the forces attached in units of Hartree per Angstrom (Ha/Ang).

squid.calcs.aneb.**orca_start_job**(*ANEB*, *i*, *state*, *charge*, *procs*, *queue*, *initial_guess*, *extra_section*, *mem*, *priority*)

A method for submitting a single point calculation using Orca for ANEB calculations.

Parameters

ANEB: **ANEB** An ANEB container holding the main ANEB simulation

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **squid.structures.atom.Atom** A list of atoms describing the image on the frame associated with index *i*.

charge: ***int*** Charge of the system.

procs: ***int*** The number of processors to use during calculations.

queue: ***str*** Which queue to submit the simulation to (this is queueing system dependent).

initial_guess: ***str*** The name of a previous simulation for which we can read in a hessian.

extra_section: ***str*** Extra settings for this DFT method.

mem: ***int*** How many MegaBytes (MB) of memory you have available per core.

priority: ***int*** Whether to submit to NBS with a given priority

Returns

orca_job: **squid.jobs.container.JobObject** A job container holding the orca simulation.

3.1.2 NEB

The NEB module simplifies the submission of Nudged Elastic Band simulations.

- g09_start_job()
- g09_results()
- orca_start_job()
- orca_results()
- NEB

class squid.calcs.neb.**NEB**(*name*, *states*, *theory*, *extra_section=''*, *initial_guess=None*, *spring_atoms=None*, *nprocs=1*, *queue=None*, *mem=2000*, *priority=None*, *disp=0*, *k=0.00367453*, *charge=0*, *multiplicity=1*, *fit_rigid=True*, *DFT='orca'*, *opt='LBFGS'*, *start_job=None*, *get_results=None*, *new_opt_params={}*, *callback=None*, *ci_neb=False*, *ci_N=5*, *no_energy=False*)

A method for determining the minimum energy pathway of a reaction using DFT. Note, this method was written for atomic orbital DFT codes; however, is potentially generalizable to other programs.

Parameters

name: ***str*** The name of the NEB simulation to be run.

states: ***list, list,*** **squid.structures.atom.Atom** A list of frames, each frame being a list of atom structures. These frames represent your reaction coordinate.

theory: ***str*** The route line for your DFT simulation.

extra_section: ***str, optional*** Additional parameters for your DFT simulation.

initial_guess: ***list, str, optional*** TODO - List of strings specifying a previously run NEB simulation, allowing restart capabilities.

spring_atoms: ***list, int, optional*** Specify which atoms will be represented by virutal springs in the NEB calculations. Default includes all.

nprocs: ***int, optional*** The number of processors for your simulation.

queue: ***str, optional*** Which queue you wish your simulation to run on (queueing system dependent). When None, NEB is run locally.

mem: ***float, optional*** Specify memory constraints (specific to your X_start_job method).

priority: ***int, optional*** Whether to submit a DFT simulation with some given priority or not.

disp: ***int, optional*** Specify for additional stdout information.

charge: ***int*** Charge of the system.

multiplicity: ***int*** Multiplicity of the system.

k: ***float, optional*** The spring constant for your NEB simulation.

fit_rigid: ***bool, optional*** Whether you want to use procrustes to minimize motion between adjacent frames (thus minimizing error due to excessive virtal spring forces).

DFT: ***str, optional*** Specify if you wish to use the default X_start_job and X_results functions where X is either g09 or orca.

opt: ***str, optional*** Select which optimization method you wish to use from the following: BFGS, LBFGS, SD, FIRE, QM, CG, scipy_X. Note, if using scipy_X, change X to be a valid scipy minimize method.

start_job: ***func, optional*** A function specifying how to submit your NEB single point calculations. Needed if DFT is neither orca nor g09.

get_results: ***func, optional*** A function specifying how to read your NEB single point calculations. Needed if DFT is neither orca nor g09. Note, this function returns two things: list of energies, list of atoms. Further, the forces are contained within each atom object. It also requires that the forces on the state object be updated within said function (for more info see example codes). Finally, if using no_energy=True, then return None (or an empty list) for the energies.

new_opt_params: ***dict, optional*** Pass any additional parameters to the optimization algorithm.

callback: ***func, optional*** A function to be run after each each to calculate().

ci_neb: ***bool, optional*** Whether to use the climbing image variation of NEB.

ci_N: ***int, optional*** How many iterations to wait in climbing image NEB before selecting which image to be used.

no_energy: ***bool, optional*** A flag to turn on an experimental method, in which our selection of the tangent is based on only the force, and not the energy. Note, the code still expects the get_results function to return two things, so just have it return (None, atoms + forces).

Returns

This `NEB` object.

References

- Henkelman, G.; Jonsson, H. The Journal of Chemical Physics 2000, 113, 9978-9985.
- Jonsson, H.; Mills, G.; Jacobson, K. W. In Classical and Quantum Dynamics in Condensed Phase Simulations;
- Berne, B. J., Ciccotti, G., Coker, D. F., Eds.; World Scientific, 1998; Chapter 16, pp 385-404.
- Armijo, L. Pacific Journal of Mathematics 1966, 16.
- Sheppard, D.; Terrell, R.; Henkelman, G. The Journal of Chemical Physics 2008, 128.
- Henkelman, G.; Uberuaga, B. P.; Jonsson, H. Journal of Chemical Physics 2000, 113.
- Atomic Simulation Environment - https://wiki.fysik.dtu.dk/ase/

align_coordinates(*r*, *B=None*, *H=None*, *return_matrix=False*)

Get a rotation matrix A that will remove rigid rotation from the new coordinates r. Further, if another vector needs rotating by the same matrix A, it should be passed in B and will be rotated. If a matrix also needs rotating, it can be passed as H and also be rotated.

Parameters

r: ***list, float*** 1D array of atomic coordinates to be rotated by procrustes matrix A.

B: ***list, list, float, optional*** A list of vectors that may also be rotated by the same matrix as *r*.

H: ***list, list, float, optional***

A matrix that should also be rotated via: H = R * H * R.T

return_matrix: ***bool, optional*** Whether to also return the rotation matrix used or not.

Returns

rotations: ***dict*** A dictionary holding 'A', the rotation matrix, 'r', the rotated new coordinates, 'B', a list of all other vectors that were rotated, and 'H', a rotated matrix.

squid.calcs.neb.**g09_results**(*NEB*, *step_to_use*, *i*, *state*)

A method for reading in the output of Gaussian09 single point calculations for NEB calculations. This will both (a) assign forces to the atoms stored in state and (b) return the energy and atoms.

Parameters

NEB: **`NEB`** An NEB container holding the main NEB simulation

step_to_use: ***int*** Which iteration in the NEB sequence the output to be read in is on.

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **`squid.structures.atom.Atom`** A list of atoms describing the image on the frame associated with index *i*.

Returns

new_energy: ***float*** The energy of the system in Hartree (Ha).

new_atoms: ***list,*** **squid.structures.atom.Atom** A list of atoms with the forces attached in units of Hartree per Angstrom (Ha/Ang).

squid.calcs.neb.**g09_start_job**(*NEB*, *i*, *state*, *charge*, *multiplicity*, *nprocs*, *queue*, *initial_guess*, *extra_section*, *mem*, *priority*, *extra_keywords={}*)

A method for submitting a single point calculation using Gaussian09 for NEB calculations.

Parameters

NEB: **NEB** An NEB container holding the main NEB simulation

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **squid.structures.atom.Atom** A list of atoms describing the image on the frame associated with index *i*.

charge: ***int*** Charge of the system.

multiplicity: ***int*** Multiplicity of the system.

nprocs: ***int*** The number of processors to use during calculations.

queue: ***str*** Which queue to submit the simulation to (this is queueing system dependent).

initial_guess: ***str*** The name of a previous simulation for which we can read in a hessian.

extra_section: ***str*** Extra settings for this DFT method.

mem: ***int*** How many Mega Words (MW) you wish to have as dynamic memory.

priority: ***int*** Whether to submit the job with a given priority (NBS). Not setup for this function.

extra_keywords: ***dict, optional*** Specify extra keywords beyond the defaults.

Returns

g09_job: **squid.jobs.container.JobObject** A job container holding the g09 simulation.

squid.calcs.neb.**orca_results**(*NEB*, *step_to_use*, *i*, *state*)

A method for reading in the output of Orca single point calculations for NEB calculations. This will both (a) assign forces to the atoms stored in state and (b) return the energy and atoms.

Parameters

NEB: **NEB** An NEB container holding the main NEB simulation

step_to_use: ***int*** Which iteration in the NEB sequence the output to be read in is on.

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **squid.structures.atom.Atom** A list of atoms describing the image on the frame associated with index *i*.

Returns

new_energy: ***float*** The energy of the system in Hartree (Ha).

new_atoms: ***list,*** **squid.structures.atom.Atom** A list of atoms with the forces attached in units of Hartree per Angstrom (Ha/Ang).

squid.calcs.neb.**orca_start_job**(*NEB*, *i*, *state*, *charge*, *multiplicity*, *nprocs*, *queue*, *initial_guess*, *extra_section*, *mem*, *priority*, *extra_keywords={}*)

A method for submitting a single point calculation using Orca for NEB calculations.

Parameters

NEB: **NEB** An NEB container holding the main NEB simulation

i: ***int*** The index corresponding to which image on the frame is to be simulated.

state: ***list,*** **squid.structures.atom.Atom** A list of atoms describing the image on the frame associated with index *i*.

charge: ***int*** Charge of the system.

multiplicity: ***int*** Multiplicity of the system.

nprocs: ***int*** The number of processors to use during calculations.

queue: ***str*** Which queue to submit the simulation to (this is queueing system dependent).

initial_guess: ***str*** The name of a previous simulation for which we can read in a hessian.

extra_section: ***str*** Extra settings for this DFT method.

mem: ***int*** How many MegaBytes (MB) of memory you have available per core.

priority: ***int*** Whether to submit to NBS with a given priority

extra_keywords: ***dict, optional*** Specify extra keywords beyond the defaults.

Returns

orca_job: **squid.jobs.container.JobObject** A job container holding the orca simulation.

3.2 files

3.2.1 cml_io

squid.files.cml_io.**read_cml**(*name*)

Read in a file written in the Chemical Markup Language (CML) format. As cml files may hold more than simple atomic coordinates, we return a list of molecules instead.

Parameters

name: ***str*** File name.

Returns

molecules: ***list,*** **squid.structures.molecule.Molecule** A list of molecules in read in from the CML file.

squid.files.cml_io.**write_cml**(*atoms*, *name=None*, *bonds=None*)

Write atomic coordinates and any other relevant information into a file using the Chemical Markup Language (CML) format.

Parameters

atoms: **...** A list of atomic coordinates to be stored. Note, you may also input a molecule object which stores relevant bonding information. You may further pass a System object that has further information.

name: ***str, optional*** The name of the output file (either ending or not in .cml).

bonds: ***list,*** **squid.structures.topology.Connector,** ***optional*** A list of bonds within the system. This is useful when the input is a list of squid.structures.atom.Atom.

Returns

None

3.2.2 misc

squid.files.misc.**close_pipes**(*p*)
A simple function to close the pipes if they remain open.

squid.files.misc.**is_exe**(*fpath*)
A function to determine if a file is an executable.

Parameters

fpath: ***str*** Path to a file.

Returns

is_executable: ***bool*** Whether the file is an executable or not.

References

- http://stackoverflow.com/a/377028

squid.files.misc.**last_modified**(*name*)
Determine when a file was last modified in seconds.

Parameters

name: ***str*** Name of the file.

Returns

time: ***datetime.datetime*** The last time this file was modified in the standard python datetime format.

squid.files.misc.**which**(*program*)
A function to return the full path of a system executable.

Parameters

program: ***str*** The name of the system executable to find.

Returns

path: ***str or None*** The path to the system executable. If none exists, then None.

References

- http://stackoverflow.com/a/377028

3.2.3 xyz_io

squid.files.xyz_io.**read_xyz**(*name, cols=['element', 'x', 'y', 'z'], cast_elem_to_sym=True, fast=True*)
Read in a file written in the XYZ file format. This is an improved version, accounting for xyz files of varying atom numbers.

Parameters

name: ***str*** File name with or without .xyz file extension.

cols: ***list, str, optional*** The specific columns in this xyz file. Note - we may not support all possibilities, and order matters!

cast_elem_to_sym: ***bool, optional*** Whether to cast the element into the symbol (ex. 2 becomes He).

fast: ***bool, optional*** If specified, you are promising that this xyz file has the columns [element x y z]. Further, if speed truly matters and you do not want to foce cast element into symbols, we recommend setting cast_elem_to_sym=False.

Returns

frames: ***list, list,*** **squid.structures.atom.Atom** A list of atoms read in from the xyz file. If there is only one frame, then only a *list* of squid.structures.atom.Atom is returned.

squid.files.xyz_io.**read_xyz_gen**(*name, cols=['element', 'x', 'y', 'z'], cast_elem_to_sym=True, fast=False*)

This will yield a frame from an xyz file.

Parameters

name: ***str*** File name with or without .xyz file extension.

cols: ***list, str, optional*** The specific columns in this xyz file. Note - we may not support all possibilities, and order matters!

cast_elem_to_sym: ***bool, optional*** Whether to cast the element into the symbol (ex. 2 becomes He).

fast: ***bool, optional*** If specified, you are promising that this xyz file has the columns [element x y z]. Further, if speed truly matters and you do not want to foce cast element into symbols, we recommend setting cast_elem_to_sym=False.

Returns

yield: ***list,*** **squid.structures.atom.Atom** A frame from an xyz file.

squid.files.xyz_io.**write_xyz**(*frames*, *name='out'*, *ID='Atoms'*)

Write frames of atomic conformations to a file written in the XYZ file format.

Parameters

frames_or_system: ***list,*** **squid.structures.atom.Atom** Atoms to be written to an xyz file.

name: ***str, optional*** A filename for the xyz file.

ID: ***str, optional*** What is to be written on the xyz comment line.

Returns

None

3.3 forcefields

3.3.1 connectors

class squid.forcefields.connectors.**Angle**(*indices=None*, *energies=None*, *equilibs=None*, *line=None*)

classmethod load_opls(*angle_types*, *pfile_name=None*, *restrict=None*)

Given a parameter file, inport the Angle parameters if possible.

Parameters

angle_types: ***list, dict*** Angle types from a parsed opls parameter file.

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

angle_objs: ***list, Angle*****, or** ***None*** Returns a list of Angle objects if possible, else None.

class squid.forcefields.connectors.**Bond**(*indices=None*, *energies=None*, *equilibs=None*, *line=None*)

classmethod load_opls(*bond_types*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the Bond parameters if possible.

Parameters

bond_types: ***list, dict*** Bond types from a parsed opls parameter file.

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

bond_objs: ***list, Bond*****, or** ***None*** Returns a list of Bond objects if possible, else None.

class squid.forcefields.connectors.**Dihedral**(*indices=None*, *energies=None*, *equilibs=None*, *line=None*)

classmethod load_opls(*dihedral_types*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the Dihedral parameters if possible.

Parameters

dihedral_types: ***list, dict*** Dihedral types from a parsed opls parameter file.

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

dihedral_objs: ***list, Dihedral*****, or** ***None*** Returns a list of Dihedral objects if possible, else None.

class squid.forcefields.connectors.**HarmonicConnector**(*indices=None*, *energies=None*, *equilibs=None*, *line=None*)
Initialize a general connector object. Either pass indices, energies, and equilibs, or pass a line to be parsed.

This object contains the following:

- `assign_line()`
- `fix()`
- `load_opls()`
- `load_smrff()`
- `pack()`
- `parse_line()`
- `printer()`
- `unpack()`

- `validate()`

Parameters

indices: ***list or tuple, str or int*** The indices of the atom types in this connection.

energies: ***list, float*** A list of energies associated with this connection.

equilibs: ***list, float*** A list of equilibrium distances/angles for this connection.

line: ***str*** A line from a parameter file to be parsed.

Returns

HarmonicConnector: **`squid.forcefields.connectors.HarmonicConnector`** A HarmonicConnector object.

`assign_line`(*line*)
Parse line inputs and assign to this object.

Parameters

line: ***str*** A string that holds information.

Returns

None

`fix`(*params='all'*, *value=None*)
This will fix these parameters by assigning bounds to the values themselves.

Parameters

params: ***str, optional*** Whether to fix everything (all) or just a specific parameter by name.

value: ***float, optional*** The value to fix the parameter to.

Returns

None

`classmethod load_opls`(*atom_types*, *pfile_name=None*, *restrict=None*)
Given a parameter file, import the Coulomb parameters if possible.

Parameters

atom_types: ***list, dict, ...*** Atom types from a parsed opls parameter file.

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

coul_objs: ***list, Coul*****, or** ***None*** Returns a list of Coul objects if possible, else None.

`classmethod load_smrff`(*parsed_file*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the LJ parameters if possible.

Parameters

parsed_file: ***str*** A parsed smrff parameter file input string (no comments or trailing white spaces)

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then parsed_file is ignored (you may simply pass None as parsed_file).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

obj: ***list, . . . ,* or *None*** Returns a list of objects if possible, else None.

pack (*params*, *with_indices=False*)
This function packs the LJ object from a list.

Parameters

params: ***list*** A list holding the indices, sigma, and epsilon (IN THAT ORDER).

with_indices: ***bool, optional*** Whether indices are included in params or not.

Returns

None

static parse_line (*line*)
Parse line inputs.

Parameters

line: ***str*** A string that holds coulomb information.

Returns

None

printer (*bounds=None*, *with_indices=False*, *map_indices=None*)
This prints out a representation of this LJ object, in the format that is output to the smrff parameter file.

Parameters

bounds: ***int, optional*** Whether to output the lower bounds (0), or upper boudns (1). If None, then the parameters themselves are output instead (default).

with_indices: ***bool, optional*** Whether indices should be returned or not.

map_indices: ***func, optional*** A function to map the indices. This is useful when converting from OPLS atom type to structure type.

Returns

lj: ***str*** A string representation of LJ. The indices, sigma, and epsilon are printed, in that precise order. Note, numbers are printed to exactly 3 decimal places.

unpack (*with_indices=False*)
This function unpacks the LJ object into a list.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the list.

Returns

coul: ***list, str/float*** A list, holding the string of the indices and the float of the charge.

validate ()
This function will validate data integrity. In this case, we simply ensure data types are appropriate.

Returns

None

3.3.2 coulomb

class squid.forcefields.coulomb.**Coul**(*index=None*, *charge=None*, *mass=None*, *element=None*, *line=None*)

Initialize the coulomb object. This should be done with either individual information (index, charge, mass, element) or parsed from a string (line).

This object contains the following:

- `assign_line()`
- `fix()`
- `generate()`
- `load_opls()`
- `load_smrff()`
- `pack()`
- `parse_line()`
- `print_lower()`
- `print_upper()`
- `unpack()`
- `validate()`

Parameters

index: ***str or int*** The index of the atom type.

charge: ***float*** The charge.

mass: ***float*** The mass.

element: ***str*** The atomic element string/symbol.

line: ***str*** A line from a parameter file to be parsed.

Returns

coulomb: **squid.forcefields.coulomb.Coul** A Coul object.

assign_line(*line*)

Parse line.

Parameters

line: ***str*** A string that holds coulomb information.

Returns

None

fix(*params='all'*, *value=None*)

This will fix these parameters by assigning bounds to the values themselves.

Parameters

params: ***str, optional*** Whether to fix everything (all) or just the charge (charge)

value: ***list, float, or float, optional*** The value to fix the param to. If None, then it is fixed to the current value. If params is all, then value must be a list of values.

Returns

None

classmethod generate (*atom_types*, *elems*, *signs*)
Randomly generate parameters for coulomb.

Parameters

atom_types: ***list, str*** A list of all the atom types to have parameters generated for.

elems: ***list, str*** List of the elements (in the same order as atom_types).

signs: ***list, float*** The list of the signs of the charges (in the same order as the atom_types).

Returns

coul_objs: ***list,*** **squid.forcefields.coulomb.Coul** Returns a list of Coul objects.

classmethod load_opls (*atom_types*, *pfile_name=None*, *restrict=None*)
Given a parameter file, import the Coulomb parameters if possible.

Parameters

atom_types: ***list, dict, ...*** Atom types from a parsed opls parameter file.

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

coul_objs: ***list,*** **squid.forcefields.coulomb.Coul, or** ***None*** Returns a list of Coul objects if possible, else None.

classmethod load_smrff (*parsed_file*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the coulomb parameters if possible.

Parameters

parsed_file: ***str*** A parsed smrff parameter file input string (no comments or trailing white spaces)

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then parsed_file is ignored (you may simply pass None as parsed_file).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

coul_objs: ***list,*** **squid.forcefields.coulomb.Coul, or** ***None*** Returns a list of Coul objects if possible, else None.

pack (*params*)
This function packs the coulomb object from a list.

Parameters

params: ***list*** A list holding the index and the charge. Note, if you wish to only assign charge, then do so manually.

Returns

None

static parse_line(*line*)
Parse line inputs and assign to this object.

Parameters

line: ***str*** A string that holds coulomb information.

Returns

index: ***str*** The label in the line corresponding to atom type.

charge: ***float*** The atomic charge.

element: ***str*** The atomic symbol.

mass: ***float*** The atomic mass.

print_lower()
Print the lower bounds.

Returns

bounds: ***str*** The lower bounds.

print_upper()
Print the upper bounds.

Returns

bounds: ***str*** The upper bounds.

unpack(*with_indices=True*, *with_bounds=False*)
This function unpacks the coulomb object into a list.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the list.

with_bounds: ***bool, optional*** Whether to return bounds as well or not.

Returns

coul: ***list, str/float*** A list, holding the string of the index and the float of the charge.

validate()
This function will validate data integrity. In this case, we simply ensure data types are appropriate.

Returns

None

3.3.3 helper

squid.forcefields.helper.**check_restriction**(*p*, *restrict*)
Checks if p is within the restricted set.

Parameters

p: ***obj*** Some parameter object, such as Coulomb, Morse, etc.

restrict: ***list, int*** A list of indices that we want to use.

Returns

contained: ***bool*** Whether p is completely in restrict (True) or not (False).

squid.forcefields.helper.**map_to_lmp_index**(*p*, *restrict*)

Given some set of labels, return the corresponding lammps index. For example, we hold restrict as a list of atom types for LAMMPS. The corresponding index in restrict is one less than the lammps type we output (say, in the data file). As such, this function tries to robustly convert p into its corresponding lammps index.

Parameters

p: ***obj*** Some parameter object, such as Coulomb, Morse, etc.

restrict: ***list, int*** A list of indices that we want to use.

Returns

mapped_indices: ... Usually a list of the mapped indices.

squid.forcefields.helper.**random_in_range**(*bounds*)

Return a random number in the given bounds: [lower, upper).

Parameters

bounds: ***list, float*** A lower and upper bound.

Returns

rand: ***float*** A random number in the specified range.

3.3.4 lj

class squid.forcefields.lj.**LJ**(*index=None*, *sigma=None*, *epsilon=None*, *line=None*)

Initialize the LJ object. Either pass index + sigma + epsilon, or pass line. If all are passed, then an error will be thrown.

This object contains the following:

- assign_line()
- fix()
- generate()
- load_opls()
- load_smrff()
- pack()
- parse_line()
- pair_coeff_dump()
- print_lower()
- print_upper()
- unpack()
- validate()

Parameters

index: ***str or int*** The index of the atom type.

sigma: ***float*** Sigma in LJ expression.

epsilon: ***float*** Epsilon in LJ expression.

line: ***str*** A line from a parameter file to be parsed.

Returns

lj: `squid.forcefields.lj.LJ` A LJ object.

`assign_line`(*line*)
Parse line inputs and assign to this object.

Parameters

line: *str* A string that holds LJ information.

Returns

None

`fix`(*params='all'*, *value=None*)
This will fix these parameters by assigning bounds to the values themselves.

Parameters

params: *str, optional* Whether to fix everything (all), or a specific value (sigma or epsilon).

value: *list, float, or float, optional* The value to fix the param to. If None, then it is fixed to the current value. If params is all, then value must be a list of values.

Returns

None

`classmethod generate`(*atom_types*)
Randomly generate parameters for lj sigma and epsilon.

Parameters

atom_types: *list, str* A list of all the atom types to have parameters generated for.

Returns

lj_objs: *list,* `squid.forcefields.lj.LJ` Returns a list of LJ objects.

`classmethod load_opls`(*atom_types*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the LJ parameters if possible.

Parameters

atom_types: *list, dict, ...* Atom types from a parsed opls parameter file.

pfile_name: *str* The name of a parameter file to be parsed. If specified, then pfile is ignored (you may simply pass None as pfile).

restrict: *list, str, optional* A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

lj_objs: *list,* `squid.forcefields.lj.LJ`, or *None* Returns a list of LJ objects if possible, else None.

`classmethod load_smrff`(*parsed_file*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the coulomb parameters if possible.

Parameters

parsed_file: *str* A parsed smrff parameter file input string (no comments or trailing white spaces)

pfile_name: *str* The name of a parameter file to be parsed. If specified, then parsed_file is ignored (you may simply pass None as parsed_file).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

lj_objs: ***list,*** **`squid.forcefields.lj.LJ`, or** ***None*** Returns a list of LJ objects if possible, else None.

`pack`(*params*)
This function packs the LJ object from a list.

Parameters

params: ***list*** A list holding the index, sigma, and epsilon (IN THAT ORDER).

Returns

None

`pair_coeff_dump`()
Return a string representation of the pair coefficients. In this case, it simply is the epsilon and sigma values with a space.

Returns

coeff_str: ***str*** A string representation of the pair coefficients.

`static parse_line`(*line*)
Parse line inputs.

Parameters

line: ***str*** A string that holds LJ information.

Returns

index: ***str*** The label in the line corresponding to atom type.

sigma: ***float*** The position where the potential well equals 0.

epsilon: ***float*** The depth of the well.

`print_lower`()
This prints out a representation of this LJ object's lower bounds, in the format that is output to the smrff parameter file.

Returns

lj: ***str*** A string representation of LJ. The index, sigma, and epsilon are printed, in that precise order. Note, numbers are printed to exactly 2 decimal places.

`print_upper`()
This prints out a representation of this LJ object's upper bounds, in the format that is output to the smrff parameter file.

Returns

lj: ***str*** A string representation of LJ. The index, sigma, and epsilon are printed, in that precise order. Note, numbers are printed to exactly 2 decimal places.

`unpack`(*with_indices=True*, *with_bounds=False*)
This function unpacks the LJ object into a list.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the list.

with_bounds: ***bool, optional*** Whether to also return the bounds or not.

Returns

coul: ***list, str/float*** A list, holding the string of the index and the float of the charge.

validate(*warn=True*)
This function will validate data integrity. In this case, we simply ensure data types are appropriate.

Parameters

warn: ***bool, optional*** At times, weird parameters may exist (ex. sigma=0). If this is the case, we will push them to a more realistic lowerbound and print a warning (True) or crash (False).

Returns

None

3.3.5 morse

class squid.forcefields.morse.**Morse**(*indices=None*, *D0=None*, *alpha=None*, *r0=None*, *rc=None*, *line=None*)
Initialize the Morse object. The potential form can be found on the LAMMPs webpage (http://lammps.sandia.gov/doc/pair_Morse.html). Either specify all the parameters, or pass a string to line, but not both. If both are specified, an error will be thrown.

This object contains the following:

- `assign_line()`
- `fix()`
- `generate()`
- `load_smrff()`
- `pack()`
- `pair_coeff_dump()`
- `parse_line()`
- `print_lower()`
- `print_upper()`
- `set_binder()`
- `set_nonbinder()`
- `unpack()`
- `validate()`

Parameters

indices: ***list or tuple, str or int*** The indices of the atom types in this pairwise interaction.

D0: ***float*** D0 describes the well depth (energy units) (defined relative to the dissociated atoms).

alpha: ***float*** alpha controls the 'width' of the potential (the smaller alpha is, the larger the well).

r0: ***float*** r0 describes the equilibrium bond distance.

rc: ***float*** rc describes the cutoff of the pairwise interaction.

line: ***str*** A line from a parameter file to be parsed.

Returns

Morse: `squid.forcefields.morse.Morse` A Morse object.

`assign_line`(*line*)
Parse line inputs and assign to this object.

Parameters

line: *str* A string that holds a three-body Morse parameter set.

Returns

None

`fix`(*params='all'*, *value=None*)
This will fix these parameters by assigning bounds to the values themselves.

Parameters

params: *str, optional* Whether to fix everything (all), or a specific value (D0, alpha, r0, or rc).

value: *list, float, or float, optional* The value to fix the charge to. If None, then it is fixed to the current value. If params is all, then value must be a list of values.

Returns

None

`classmethod generate`(*atom_types*, *gen_rc=False*)
Randomly generate parameters for morse.

Parameters

atom_types: *list, str* A list of all the atom types to have parameters generated for.

gen_rc: *bool, optional* Whether to generate the cutoff radius, or not.

Returns

morse_objs: *list,* `squid.forcefields.morse.Morse` Returns a list of Morse objects.

`classmethod load_smrff`(*parsed_file*, *pfile_name=None*, *restrict=None*)
Given a parameter file, import the Morse parameters if possible.

Parameters

parsed_file: *str* A parsed smrff parameter file input string. (no comments or trailing white spaces)

pfile_name: *str* The name of a parameter file to be parsed. If specified, then parsed_file is ignored. (you may simply pass None as parsed_file)

restrict: *list, str, optional* A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

Morse_objs: *list,* `squid.forcefields.morse.Morse`, or *None* Returns a list of Morse objects if possible, else None.

`pack`(*params*)
This function packs the Morse object from a list.

Parameters

params: *list* A list holding the indices, D0, alpha, r0, rc.

Returns

None

pair_coeff_dump()
Return a string representation of the pair coefficients.

Returns

coeff_str: ***str*** A string representation of the pair coefficients.

static parse_line(*line*)
Parse line inputs.

Parameters

line: ***str*** A string that holds a three-body Morse parameter set.

Returns

indices: ***tuple, str*** The labels in the line corresponding to atom types.

D0: ***float*** The depth of the potential well.

alpha: ***float*** The width of the potential well.

r0: ***float*** The interatomic distance associated with the minimum.

rc: ***float*** The cutoff of the potential.

print_lower()
This prints out a representation of this Morse object's lower bound, in the format that is output to the smrff parameter file.

Returns

Morse: ***str*** A string representation of Morse parameters. It is in the following order: indices D0 alpha r0 rc

print_upper()
This prints out a representation of this Morse object's upper bound, in the format that is output to the smrff parameter file.

Returns

Morse: ***str*** A string representation of Morse parameters. It is in the following order: indices D0 alpha r0 rc

set_binder()
This will adjust bounds such that the parameters are in a range of a morse "Bond". This means that:

0.5 < r < 4.0

Returns

None

set_nonbinder()
This will adjust bounds such that the parameters are in a range of a morse "Non-Bond". This means that:

4.0 < r < 10.0 0.1 < D0 < 50.0 0.1 < alpha < 5.0

Returns

None

unpack(*with_indices=True*, *bounds=None*, *with_bounds=False*)
This function unpacks the Morse object into a list.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the list.

bounds: ***int, optional*** Whether to output the lower bounds (0), or upper bounds (1). If None, then the parameters themselves are output instead (default).

Returns

Morse: ***list, str/float*** A list, holding the string of the indices, D0, alpha, r0, rc.

validate()
This function will validate data integrity. In this case, we simply ensure data types are appropriate.

Returns

None

3.3.6 opls

squid.forcefields.opls.**parse_pfile**(*parameter_file='/home/hherbol/programs/squid/squid/forcefields/potentials/oplsaa*
pair_style='lj/cut')
Reads an opls parameter file written in the Tinker file format.

Parameters

parameter_file: ***str, optional*** Relative or absolute path to an opls parameter file, written in the Tinker file format.

pair_style: ***str, optional*** The pair style to be assigned.

Returns

atom_types: ***list, dict*** A list of the forcefield types for atoms, stored as a dictionary.

bond_types: ***list, dict*** A list of the forcefield types for bonds, stored as a dictionary.

angle_types: ***list, dict*** A list of the forcefield types for angles, stored as a dictionary.

dihedral_types: ***list, dict*** A list of the forcefield types for dihedrals, stored as a dictionary.

3.3.7 parameters

class squid.forcefields.parameters.**Parameters**(*restrict*, *opls_file='/home/hherbol/programs/squid/squid/forcefield*
smrff_file=None,
force_ters_2body_symmetry=False)
A Parameters object that holds force field parameters. It requires the input of which atom types to get parameters for, so as to not read in an entire force field.

This object contains the following:

- dump_angles()
- dump_bonds()
- dump_dihedrals()
- dump_lj_cut_coul_cut()
- dump_lj_cut_coul_long()

- dump_morse()
- dump_set_charge()
- dump_smooths()
- dump_style()
- dump_tersoff()
- fix()
- generate()
- get_smrff_style()
- load_opls()
- load_smrff()
- mapper()
- num_free_parameters()
- pack()
- set_smoothed_pair_potentials()
- set_all_masks()
- set_mask()
- set_opls_mask()
- unpack()
- write_smrff()

Parameters

restrict: ***list, str*** A list of strings specifying which types are to be used. Note, you must have unique types, lest they be overwritten. If None is passed, then everything available is read in.

opls_file: ***str, optional*** The path to an OPLS parameter file. Default is internally stored parameters in squid.

smrff_file: ***str, optional*** The path to a SMRFF parameter file. Default is None.

force_ters_2body_symmetry: ***bool, optional*** Whether to force any read in tersoff parameters to have the 2-body symmetry we expect.

Returns

params: **squid.forcefields.parameters.Parameters** This object.

dump_angles(*in_input_file=True*)

Get a string for lammps input in regards to assigning angle coeffs.

Parameters

in_input_file: ***bool, optional*** Whether to dump the bonds in the input file style format (True) or the data file style format (False)

Returns

coeffs: ***str*** A string of angle coeffs, with new line characters between different angles.

dump_bonds(*in_input_file=True*)
Get a string for lammps input in regards to assigning bond coeffs.

Parameters

in_input_file: ***bool, optional*** Whether to dump the bonds in the input file style format (True) or the data file style format (False)

Returns

coeffs: ***str*** A string of bond coeffs, with new line characters between different bonds.

dump_dihedrals(*in_input_file=True*)
Get a string for lammps input in regards to assigning dihedral coeffs.

Parameters

in_input_file: ***bool, optional*** Whether to dump the bonds in the input file style format (True) or the data file style format (False)

Returns

coeffs: ***str*** A string of dihedral coeffs, with new line characters between different dihedrals.

dump_lj_cut_coul_cut()
This function will get the lammps command line argument for lj/cut/coul/cut of everything within the Parameters object.

Parameters

None

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each lj/cut/coul/cut command possible within this parameter set.

dump_lj_cut_coul_long()
This function will get the lammps command line argument for lj/cut/coul/long of everything within the Parameters object.

Parameters

None

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each lj/cut/coul/long command possible within this parameter set.

dump_morse()
This function will get the lammps command line argument for morse of everything within the Parameters object.

Parameters

None

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each morse command possible within this parameter set.

dump_set_charge()
This function will get the lammps command line argument for "set type" of everything within the Parameters object.

Parameters

None

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each morse command possible within this parameter set.

dump_smooths()
This function will get the lammps command line argument for smrff smooths of everything within the Parameters object.

Parameters

None

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each smrff smooth command possible within this parameter set.

dump_style(*style=None*, *tfile_name=None*, *tstyle_smrff=False*, *write_file=False*, *in_input_file=True*)
This function will dump LAMMPS commands "pair_coeff" for chosen styles.

Parameters

style: ***str*** Whether to ignore the universal bounds, assigned in this smrff.py file.smrff

tfile_name: ***str*** The name of the tersoff file.

tstyle_smrff: ***bool, optional*** Whether to output for SMRFF style (one line allocates memory, the rest overwrites the parameters) or not.

write_file: ***bool, optional*** Whether to write any files (ex. tersoff files) or not.

in_input_file: ***bool, optional*** Whether to dump the bonds in the input file style format (True) or the data file style format (False)

Returns lammps_command: *str*

dump_tersoff(*tfile_name*, *tstyle_smrff*)
This function will get the lammps command line argument for tersoff of everything within the Parameters object.

Parameters

tfile_name: ***str*** The name of the tersoff file.

tstyle_smrff: ***bool, optional*** Whether to output for SMRFF style (one line allocates memory, the rest overwrites the parameters) or not. If True, this will NOT generate a tersoff file.

Returns

cmds: ***str*** A string, separated with new lines, with pair_coeff for each tersoff command possible within this parameter set.

fix(*style*, *label*, *params='all'*, *value=None*)
This function will fix a specific style (coul, lj, morse, etc), label (where label is the atom label/type you want to fix), and the component (ex, sigma in LJ).

Parameters

style: ***str*** Which style to fix. Options are coul, lj, morse, and ters.

label: ... Which atom type should be fixed. If *, then everything. Note that for some situations this may be a list of values (as in the case of tersoff).

params: *str, optional* Whether to fix everything (all), or a specific value (style dependant).

value: *list, float, or float, optional* The value to fix the param to. If None, then it is fixed to the current value. If params is all, then value must be a list of values.

Returns

None

generate(*elems*, *signs=None*, *couple_smooths=True*, *tersoff_form='original'*)
For every mask that is true, we will generate random data.

Parameters

elems: *list, str* A list of the elements, matching 1-to-1 with the given smrff_types.

signs: *list, float* A list of charge signs, matching 1-to-1 with the given smrff_types.

couple_smooths: *bool, optional* Whether to couple together like smooths. If true, when you have a situation in which, say, sin_l and sin_r match up, then this will ensure that these two overlap perfectly.

tersoff_form: *str* Whether to use the original tersoff_form (m=3, gamma=1) or the Albe et al tersoff_form (m=1, beta=1) for tersoff parameters. Must be original or albe.

Returns

None

get_smrff_style()
This will return the smrff style.

Returns

lammps_smrff_style: *str* The input script line for LAMMPS for the smrff pair style.

load_opls(*fname*)
Given an OPLS file name, in the Tinker format, parse it and load it into this parameters object. Note, you should specify restrict before calling this function or else everything in the file will be loaded. If restrict is specified, then restrict_structure will be automatically generated during this function call.

Parameters

fname: *str* The path to the opls file.

Returns

None

load_smrff(*fname*)
A function to read in a SMRFF parameter file.

Parameters

fname: *str* The path to a smrff parameter file. ???.smrff

Returns

None

mapper(*x*)
A generalized function to map indices of atom types to the corresponding lammps index. Note, this is generalized and should allow for a wide range of x objects.

Parameters

x: ***list or tuple or int or str or obj*** Some way of identifing the atom type. Note, if obj, then it will check for a .index, .indices, or .index2s property.

Returns

mapper_obj: ***list, str or str*** The lammps index, as either a list (if bond/angle/dihedral) or a string (if charge/lj).

num_free_parameters()

This function will return the number of unfixed parameters.

Returns

free_params: ***int*** The number of free parameters.

pack(*params*, *with_indices=False*)

Packs the parameters object from a 1D array. NOTE! This is done in primarily for parameterization; which means the 1D array will NOT have the indices in it. This can be overridden, however, by specifying the with_indices flag.

Keep in mind, to maintain symmetry requirements in tersoff (where the two-body parameters are the same between A-B B and B-A A), this function will be tied closely to the unpack function.

Parameters

params: ***list, float/int*** A list of parameters

with_indices: ***bool, optional*** Whether to account for the indices in the flat array.

Returns

None

set_all_masks(*set_on*)

Either turn all masks to being on or off.

Parameters

set_on: ***bool*** What to set all values to (True/False).

Returns

None

set_mask(*mask*)

Given a style, turn on the respective masks.

Parameters

mask: ***str*** The potential for which masks should be turned on (ex. morse)

Returns

None

set_opls_mask()

Turn off ALL masks, but leave OPLS ones on.

Returns

None

set_smoothed_pair_potentials(*local_potentials*)

This will assign masks appropriately for the input forcefield. It will also assign any necessary parameters for the desired forcefield.

NOTE! This is a rudimentary starting point and should be improved on. pair style smrff allows for N transitions, whereas the way this is currently written only allows for short-range to long-range.

Parameters

local_potentials: ***list, tuple, ...*** A list of tuples, each holding three values. The first is the potential (such as morse or tersoff). The second is the global cutoff (likely in Angstroms). The final is the smooth function to apply to this potential (such as NULL for none, or some sin_X smooth).

Returns

None

unpack(*with_indices=False*, *with_bounds=False*)

Unpacks the parameters object into a 1D array for parameterization. This means that the indices are not included during unpacking! Note, this can be overridden though if needed.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the flat array.

with_bounds: ***bool, optional*** Whether to also output the bounds for the parameters.

Returns

flat_array: ***list, float/int*** A list of floats/ints of our parameters.

bounds_lower: ***list, float/int*** If with_bounds is specified, then the lower bounds are returned.

bounds_upper: ***list, float/int*** If with_bounds is specified, then the upper bounds are returned.

write_smrff(*fname*)

A function to save a SMRFF parameter file.

Parameters

fname: ***str*** The file name to save the parameters to.

Returns

None

3.3.8 smrff

squid.forcefields.smrff.**parse_pfile**(*fname*)

This function will, given a smrff parameter file, will parse it by removing comments, trailing whitespaces, and empty lines.

Parameters

fname: ***str*** The name of the parameter file to be parsed.

Returns

parsed: ***str*** A parsed string of said parameter file.

3.3.9 tersoff

class squid.forcefields.tersoff.**Tersoff**(*indices=None*, *m=None*, *gamma=None*, *lambda3=None*, *c=None*, *d=None*, *costheta0=None*, *n=None*, *beta=None*, *lambda2=None*, *B=None*, *R=None*, *D=None*, *lambda1=None*, *A=None*, *line=None*, *form='original'*)

Initialize the Tersoff object. The potential form can be found on the LAMMPs webpage (http://lammps.sandia.gov/doc/pair_tersoff.html). Either specify all the parameters, or pass a string to line, but not both. If both are specified, an error will be thrown.

This object contains the following:

- `assign_line()`
- `dump_line()`
- `fix()`
- `generate()`
- `load_smrff()`
- `pack()`
- `parse_line()`
- `print_lower()`
- `print_upper()`
- `set_default_bounds()`
- `sorted_force_2body_symmetry()`
- `tag_tersoff_for_duplicate_2bodies()`
- `turn_off()`
- `turn_off_3body()`
- `unpack()`
- `update_2body()`
- `validate()`
- `verify_tersoff_2body_symmetry()`

Parameters

indices: ***list or tuple, str or int*** The indices of the atom types in this three-body interaction.

m: ***int*** m is an exponential term in the zeta component of the Tersoff potential. It is either 1 or 3.

gamma: ***float*** gamma is a prefactor to g(theta) in the Tersoff potential, and is between 0 (completely off) and 1 (completely on).

lambda3: ***float*** lambda3 is the exponential coefficient of the three-body tersoff interaction.

c: ***float*** c describes the numerator part of the three-body interaction (found withing the g(theta) term).

d: ***float*** d describes the denominator part of the three-body interaction (found withing the g(theta) term).

costheta0: ***float*** costheta0 gives an equilibrium angle of sorts for the three-body interaction. As such, it is restricted between -1 and 1.

n: ***float*** n is a power that the three-body interaction is taken to (or to some function of n).

beta: ***float*** beta is some scaling to the three-body interactions, found in the b_ij term of the Tersoff potential.

lambda2: ***float*** lambda2 is the exponential coefficient of the two-body attraction term in the tersoff interaction.

B: ***float*** B is the pre-factor to the two-body attraction term in the tersoff interaction.

R: ***float*** R is a component of the tersoff potential's cutoff distance.

D: ***float*** D is a component of the tersoff potential's cutoff distance.

lambda1: ***float*** lambda1 is the exponential coefficient of the two-body repulsion term in the tersoff interaction.

A: ***float*** A is the pre-factor to the two-body repulsion term in the tersoff interaction.

line: ***str*** A line from a parameter file to be parsed.

form: ***str*** Whether to use the original form (m=3, gamma=1) or the Albe et al form (m=1, beta=1). Must be original or albe.

Returns

tersoff: **`squid.forcefields.tersoff.Tersoff`** A Tersoff object.

`assign_line`(*line*, *validate=True*)

Parse line inputs and assign to this object.

Parameters

line: ***str*** A string that holds a three-body tersoff parameter set.

validate: ***bool, optional*** Whether to validate these parameters or not.

Returns

None

`dump_line`()

This function will output the pair_coeff line for tersoff in LAMMPS. Note - This line output only exists if SMRFF is installed.

Returns

line: ***str*** A pair_coeff line output in tersoff.

`fix`(*params='all'*, *value=None*)

This will fix these parameters by assigning bounds to the values themselves.

Parameters

params: ***str, optional*** Whether to fix everything (all), or a specific value (m, gamma, lambda3, c, d, costheta0, n, beta, lambda2, B, R, D, lambda1, A).

value: ***list, float, or float, optional*** The value to fix the param to. If None, then it is fixed to the current value. If params is all, then value must be a list of values.

Returns

None

classmethod generate(*atom_types*, *form='original'*)
Randomly generate parameters for tersoff.

Parameters

atom_types: ***list, str*** A list of all the atom types to have parameters generated for.

form: ***str*** Whether to use the original form (m=3, gamma=1) or the Albe et al form (m=1, beta=1). Must be original or albe.

Returns

ters_objs: ***list,*** **squid.forcefields.tersoff.Tersoff** Returns a list of Tersoff objects.

classmethod load_smrff(*parsed_file*, *pfile_name=None*, *restrict=None*)
Given a parameter file, inport the coulomb parameters if possible.

Parameters

parsed_file: ***str*** A parsed smrff parameter file input string (no comments or trailing white spaces)

pfile_name: ***str*** The name of a parameter file to be parsed. If specified, then parsed_file is ignored (you may simply pass None as parsed_file).

restrict: ***list, str, optional*** A list of atom labels to include when loading. If not specified, everything is loaded.

Returns

tersoff_objs: ***list,*** **squid.forcefields.tersoff** **or** ***None*** Returns a list of Tersoff objects if possible, else None.

pack(*params*)
This function packs the tersoff object from a list.

Parameters

params: ***list*** A list holding the indices, m, gamma, lambda3, c, d, costheta0, n, beta, lambda2, B, R, D, lambda1, A.

Returns

None

static parse_line(*line*)
Parse line inputs.

Parameters

line: ***str*** A string that holds a three-body tersoff parameter set.

Returns

indices: ***tuple, str*** Tersoff Parameter.

m: ***float*** Tersoff Parameter.

gamma: ***float*** Tersoff Parameter.

lambda3: ***float*** Tersoff Parameter.

c: ***float*** Tersoff Parameter.

d: ***float*** Tersoff Parameter.

costheta0: ***float*** Tersoff Parameter.

n: *float* Tersoff Parameter.

beta: *float* Tersoff Parameter.

lambda2: *float* Tersoff Parameter.

B: *float* Tersoff Parameter.

R: *float* Tersoff Parameter.

D: *float* Tersoff Parameter.

lambda1: *float* Tersoff Parameter.

A: *float* Tersoff Parameter.

print_lower()

This prints out a representation of this tersoff object's upper bound, in the format that is output to the smrff parameter file.

Returns

tersoff: *str* A string representation of Tersoff parameters. It is in the following order:

indices m gamma lambda3 c d costheta0 n beta lambda2 B R D lambda1 A

print_upper()

This prints out a representation of this tersoff object's upper bound, in the format that is output to the smrff parameter file.

Returns

tersoff: *str* A string representation of Tersoff parameters. It is in the following order:

indices m gamma lambda3 c d costheta0 n beta lambda2 B R D lambda1 A

set_default_bounds()

Assign default bounds. Further, if this is the case of A-B-B vs A-B-C we can simplify the bounds as only in the case of A-B-B are two body parameters n, Beta, lambda2, lambda1, and A read in.

Returns

None

turn_off()

This function essentially turns off the Tersoff potential. This is accomplished by:

1. Setting a small cutoff distance.
2. Assigning an impossibly large repulsion energy (A is large)
3. Removing attractive potential (B is 0)
4. Setting beta to 0 (this removes the three-body interaction)
5. Setting all benign (unused) parameters to 1

Returns

None

turn_off_3body()

This function essentially turns off the 3-body component of the Tersoff potential. This is accomplished by:

1. Setting beta to 0 (this removes the three-body interaction)
2. Setting all benign (unused) parameters to 1

Returns

None

unpack(*with_indices=True*, *bounds=None*, *with_bounds=False*, *for_output=False*)
This function unpacks the tersoff object into a list.

Parameters

with_indices: ***bool, optional*** Whether to also include the indices in the list.

bounds: ***int, optional*** Whether to output the lower bounds (0), or upper bounds (1). If None, then the parameters themselves are output instead (default).

with_bounds: ***bool, optional*** Whether to output the bounds or not.

for_output: ***bool, optional*** Whether this is for output (in which case we disregard 2-body sym flag)

Returns

tersoff: ***list, str/float*** A list, holding the string of the indices, m, gamma, lambda3, c, d, cos-theta0, n, beta, lambda2, B, R, D, lambda1, A.

update_2body(*other*)
Given a tersoff parameter object, update the current one with the 2-body parameters (n, beta, lambda1, lambda2, A, B).

Parameters

other: **squid.forcefields.tersoff.Tersoff** A Tersoff parameter object to get 2-body parameters from.

Returns

None

validate()
This function will validate data integrity. In this case, we simply ensure data types are appropriate.

Returns

None

squid.forcefields.tersoff.**sorted_force_2body_symmetry**(*tersoff_params*)
In the case of randomly generating parameters, we may want to randomly force the 2body symmetry condition.

Parameters

tersoff_params: ***list,*** **squid.forcefields.tersoff.Tersoff** A list of Tersoff objects.

Returns

corrected_tersoff_params: ***list,*** **squid.forcefields.tersoff.Tersoff** A list of Tersoff objects with the 2body symmetry condition ensured.

squid.forcefields.tersoff.**tag_tersoff_for_duplicate_2bodies**(*tersoff_params*)
This function will mimic the sorted_force_2body_symmetry and tag the duplicates that would be set by sorted_force_2body_symmetry.

Parameters

tersoff_params: ***list,*** **squid.forcefields.tersoff.Tersoff** A list of Tersoff objects.

Returns

tagged_tersoff_params: ***list,*** **squid.forcefields.tersoff.Tersoff** A list of the unique Tersoff objects.

Returns

squid.forcefields.tersoff.**verify_tersoff_2body_symmetry**(*tersoff_params*)

Given a list of tersoff parameters, verify that they are such that the two-body parameters are symmetric. What this means is that when we consider A-B B and B-A A, the two body parameters must be the same. This is necessary as LAMMPs will randomly isolate the two body interaction (A-B or B-A) and use the corresponding parameters.

Parameters

tersoff_params: ***list,*** **squid.forcefields.tersoff.Tersoff** A list of Tersoff objects.

Returns

None

3.4 g09

TO DO

3.5 geometry

3.5.1 misc

squid.geometry.misc.**get_center_of_geometry**(*atoms*, *skip_H=False*)

Calculate the center of geometry of the molecule.

Parameters

atoms: ***list,*** **structures.atom.Atom** A list of atoms.

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

cog: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of geometry.

squid.geometry.misc.**get_center_of_mass**(*atoms*, *skip_H=False*)

Calculate the center of mass of the molecule.

Parameters

atoms: ***list,*** **structures.atom.Atom** A list of atoms.

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

com: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of mass.

squid.geometry.misc.**rotate_atoms**(*atoms*, *m*, *around='com'*)

Rotate atoms by the given matrix *m*. Note, this happens in place. That means that the atoms in the input list will themselves be rotated. This is done so that we may rotate molecules and systems using the same code! If you do not wish for this to happen, pass to rotate_atoms a deepcopy of the atoms.

Parameters

atoms: ***list,*** **structures.atom.Atom** A list of atoms to be rotated.

m: ***list, list, float*** A 3x3 matrix describing the rotation to be applied to this molecule.

around: ***str, optional*** Whether to rotate around the center of mass (com), center of geometry (cog), or neither ("None" or None).

Returns

atoms: ***list,*** **`structures.atom.Atom`** The rotated atomic coordinates.

3.5.2 packmol

`squid.geometry.packmol.`**`get_packmol_obj`**`()`

This function will find the packmol executable and handle errors accordingly.

Returns

packmol_path: ***str*** The path to packmol.

`squid.geometry.packmol.`**`packmol`**`(`*system_obj*, *molecules*, *molecule_ratio=(1,)*, *density=1.0*, *seed=1*, *persist=True*, *number=None*, *additional=''*, *custom=None*, *extra_block_at_beginning=''*, *extra_block_at_end=''*, *tolerance=2.0*`)`

Given a list of molecules, pack this system appropriately. Note, we now will pack around what is already within the system! This is done by first generating a packmol block for the system at hand, followed by a block for the solvent.

A custom script is also allowed; however, if this path is chosen, then ensure all file paths for packmol exist. We change directories within this function to a sys_packmol folder, where all files are expected to reside.

Parameters

system_obj: **`structures.system.System`** The system object to pack the molecules into.

molecules: ***list,*** **`structures.molecule.Molecule`** Molecules to be added to this system.

molecule_ratio: ***tuple, float, optional*** The ration that each molecule in *molecules* will be added to the system.

density: ***float, optional*** The density of the system in g/mL

seed: ***float, optional*** Seed for random generator.

persist: ***bool, optional*** Whether to maintain the generated sys_packmol directory or not.

number: ***int or list, int, optional*** Overide density and specify the exact number of molecules to pack. When using a list of molecules, you must specify each in order within a list.

additional: ***str, optional*** Whether to add additional constraints to the standard packmol setup.

custom: ***str, optional*** A custom packmol script to run for the given input molecules. Note, you should ensure all necessary files are within the sys_packmol folder if using this option.

extra_block_at_beginning: ***str, optional*** An additional block to put prior to the standard block.

extra_block_at_end: ***str, optional*** An additional block to put after the standard block.

tolerance: ***float, optional*** The tolerance around which we allow atomic overlap/proximity.

Returns

None

References

- Packmol - http://www.ime.unicamp.br/~martinez/packmol/home.shtml

3.5.3 spatial

squid.geometry.spatial.**motion_per_frame**(*frames*)

Determine the root mean squared difference between atomic positions of adjacent frames. Note, as we have differences between frames, this means that we return len(frames) - 1 values.

Parameters

frames: ***list, list,*** **squid.structures.atom.Atom** List of lists of atoms.

Returns

motion: ***np.array, float*** List of motion between consecutive frames (frame_i vs frame_(i - 1)).

squid.geometry.spatial.**mvee**(*points*, *tol=0.001*)

Generate a Minimum Volume Enclosing Ellipsoid (MVEE) around atomic species. The ellipsoid is calculated for the "center form": (x-c).T * A * (x-c) = 1

For useful values, you can get the radii as follows:

```
U, Q, V = np.linalg.svd(A)
r_i = 1/sqrt(Q[i])
vol = (4/3.) * pi * sqrt(1 / np.product(Q))
```

Further, note that V is the rotation matrix giving the orientation of the ellipsoid.

NOTE! You must have a minimum of 4 atoms for this to work.

Parameters

points: ***list,*** **squid.structures.atom.Atom** A list of Atom objects.

tol: ***float, optional*** Tolerance for ellipsoid generation.

Returns

A: ***list, list, float*** Positive definite symmetric matrix of the ellipsoid's center form. This contains the ellipsoid's orientation and eccentricity.

c: ***list, float*** Center of the ellipsoid.

References

- https://www.mathworks.com/matlabcentral/fileexchange/9542-minimum-volume-enclosing-ellipsoid?requestedDomain=www.mathworks.com
- http://stackoverflow.com/questions/14016898/port-matlab-bounding-ellipsoid-code-to-python/14025140#14025140

squid.geometry.spatial.**orthogonal_procrustes**(*A*, *ref_matrix*, *reflection=False*)

Using the orthogonal procrustes method, we find the unitary matrix R with det(R) > 0 such that ||A*R - ref_matrix||^2 is minimized. This varies from that within scipy by the addition of the reflection term, allowing and disallowing inversion. NOTE - This means that the rotation matrix is used for right side multiplication!

Parameters

A: ***list,*** **squid.structures.atom.Atom** A list of atoms for which R will minimize the frobenius norm ||A*R - ref_matrix||^2.

ref_matrix: ***list,*** **squid.structures.atom.Atom** A list of atoms for which *A* is being rotated towards.

reflection: ***bool, optional*** Whether inversion is allowed (True) or not (False).

Returns

R: ***list, list, float*** Right multiplication rotation matrix to best overlay A onto the reference matrix.

scale: ***float*** Scalar between the matrices.

Derivation

Goal: minimize ||A*R - ref||^2, switch to trace

trace((A*R-ref).T*(A*R-ref)), now we distribute

trace(R'*A'*A*R) + trace(ref.T*ref) - trace((A*R).T*ref) - trace(ref.T*(A*R)), trace doesn't care about order, so re-order

trace(R*R.T*A.T*A) + trace(ref.T*ref) - trace(R.T*A.T*ref) - trace(ref.T*A*R), simplify

trace(A.T*A) + trace(ref.T*ref) - 2*trace(ref.T*A*R)

Thus, to minimize we want to maximize trace(ref.T * A * R)

u*w*v.T = (ref.T*A).T

ref.T * A = w * u.T * v

trace(ref.T * A * R) = trace (w * u.T * v * R)

differences minimized when trace(ref.T * A * R) is maximized, thus when trace(u.T * v * R) is maximized

This occurs when u.T * v * R = I (as u, v and R are all unitary matrices so max is 1)

R is a rotation matrix so R.T = R^-1

u.T * v * I = R^-1 = R.T

R = u * v.T

Thus, R = u.dot(vt)

References

- https://github.com/scipy/scipy/blob/v0.16.0/scipy/linalg/ _procrustes.py#L14
- http://compgroups.net/comp.soft-sys.matlab/procrustes-analysis -without-reflection/896635

`squid.geometry.spatial.`**`random_rotation_matrix`**(*limit_angle=None*, *lower_bound=0.1*, *MAXITER=1000000*)

Generate a random rotation matrix.

Parameters

limit_angle: ***float, optional*** Whether to confine your random rotation (in radians).

lower_bound: ***float, optional*** A lower bound for limit_angle, at which the identity is simply returned. This is necessary as the procedure to generate the limit_angle method is incredibly slow at small angles.

MAXITER: ***int, optional*** A maximum iteration for when we try to calculate a rotation matrix with some limit_angle specified.

Returns

frames: ***np.array, list, float*** A random rotation matrix.

References

- http://tog.acm.org/resources/GraphicsGems/, Ed III

squid.geometry.spatial.**rotation_matrix**(*axis*, *theta*, *units='deg'*)
Obtain a left multiplication rotation matrix, given the axis and angle you wish to rotate by. By default it assumes units of degrees. If theta is in radians, set units to rad.

Parameters

axis: ***list, float*** The axis in which to rotate around.

theta: ***float*** The angle of rotation.

units: ***str, optional*** The units of theta (deg or rad).

Returns

rotatation_matrix: ***list, list, float*** The left multiplication rotation matrix.

References

- http://stackoverflow.com/questions/6802577/python-rotation-of-3d-vector/25709323#25709323

3.5.4 transform

squid.geometry.transform.**align_centroid**(*atoms*, *recenter=True*, *skip_H=True*)
Generate a Minimum Volume Enclosing Ellipsoid (MVEE) around atomic species to align the atoms along the x-axis.

Parameters

atoms: ***list,*** **squid.structures.atom.Atom** A list of Atom objects.

recenter: ***bool, optional*** Whether to recenter the new coordinates around the origin or not. Note, this is done via the center of geometry, NOT the center of mass.

skip_H: ***bool, optional*** Whether to skip hydrogen during recentering (that is, do not take them into accound when calculating the center of geometry).

Returns

molec.atoms: ***list,*** **squid.structures.atom.Atom** Rotated atomic coordinates.

A: ***list, list, float*** Rotated positive definite symmetric matrix of the ellipsoid's center form. This contains the ellipsoid's orientation and eccentricity.

squid.geometry.transform.**interpolate**(*frame_1*, *frame_2*, *N*)
Linearly interpolate N frames between two given frames.

Parameters

frame_1: ***list,*** **squid.structures.atom.Atom** List of atoms.

frame_2: ***list,*** **squid.structures.atom.Atom** List of atoms.

N: ***int*** Number of new frames you want to generate during interpolation.

Returns

frames: ***list, list, float*** List of interpolated frames, inclusive of frame_1 and frame_2.

squid.geometry.transform.**perturbate**(*atoms*, *dx=0.1*, *dr=5*, *around='com'*, *rotate=True*)
Given a list of atomic coordinates, randomly perturbate them and apply a slight rotation.

Parameters

atoms: ***list,*** **squid.structures.atom.Atom** A list of atomic coordinates to be perturbated

dx: ***float, optional*** By how much you are willing to perturbate via translation.

dr: ***float, optional*** By how much you are willing to perturbate via rotation in degrees.

around: ***str, optional*** Whether to rotate around the center of mass (com), center of geometry (cog), or neither ("None" or None).

rotate: ***bool, optional*** Whether to randomly rotate the molecule or not.

Returns

perturbated_atoms: ***list,*** **`squid.structures.atom.Atom`** The perturbated list of atomic coordinates.

`squid.geometry.transform.`**`procrustes`**(*frames*, *count_atoms=None*, *append_in_loop=True*, *reflection=False*)

Propogate rotation along a list of lists of atoms to smooth out transitions between consecutive frames. This is done by rigid rotation and translation (no scaling and no inversions). Rotation starts at frames[0].

Parameters

frames: ***list, list,*** **`squid.structures.atom.Atom`** List of lists of atoms.

count_atoms: ***list, int, optional*** A list of indices for which translation and rotations will be calculated from.

append_in_loop: ***bool, optional*** If rotation matrices for every atom (True) is desired vs rotation matrices for every frame (False). Every rotation matrix for atoms within the same frame is the same. Thus, when this is True, multiplicates will appear.

reflection: ***bool, optional*** Whether inversion is allowed (True) or not (False).

Returns

full_rotation: ***list, list, float*** List of every rotation matrix applied. NOTE - These matrices are applied via right side multiplication.

See also

For more information, see `squid.geometry.spatial.orthogonal_procrustes()`.

`squid.geometry.transform.`**`smooth_xyz`**(*frames*, *R_max=0.5*, *F_max=25*, *N_frames=None*, *use_procrustes=True*, *fname=None*, *verbose=False*)

Smooth out an xyz file by linearly interpolating frames to minimize the maximum motion between adjacent frames. Further, this can use procrustes to best overlap adjacent frames.

Parameters

frames: ***list, list,*** **`squid.structures.atom.Atom`** A list of lists of atoms.

R_max: ***float, optional*** The maximum motion allowed between consecutive frames.

F_max: ***int, optional*** The maximum number of frames allowed before failing the smooth function.

N_frames: ***int, optional*** If this is specified, forgo the R_max and F_max and just interpolate out into N_frames. Note, if more than N_frames exists, this also cuts back into exactly N_frames.

use_procrustes: ***bool, optional*** Whether procrustes is to be used during smoothing (True), or not (False).

fname: ***str, optional*** An output file name for the smoothed frames (without the .xyz extension). If None, then no file is made.

verbose: ***bool, optional*** Whether additional stdout is desired (True), or not (False).

Returns

frames: ***list, list,*** **`squid.structures.atom.Atom`** Returns a list of smoothed frames

3.6 jdftx

TODO

3.7 jobs

3.7.1 container

class squid.jobs.container.**JobObject**(*name*, *process_handle=None*, *job_id=None*)
Job class to wrap simulations for queue submission.

Parameters

name: ***str*** Name of the simulation on the queue.

process_handle: ***process_handle, optional*** The process handle, returned by subprocess.Popen.

job_id: ***str, optional*** The job id. Usually this should be unique.

Returns

job_obj: **squid.jobs.container.JobObject** A Job object.

get_all_jobs()
Get a list of all jobs that are running and/or pending.

Parameters

detail: ***int, optional*** How much detail to get when finding jobs on the queue.

Returns

jobs_on_queue: ***list, ...*** A list of all jobs on the queue, and any other relevant information requested.

is_finished()
Check if simulation has finished or not.

Returns

is_on_queue: ***bool*** Whether the simulation is still running (True), or not (False).

wait(*tsleep=60*, *verbose=False*)
Hang until simulation has finished.

tsleep: ***int, optional*** How long to wait before checking if the job has finished in the loop. Default is 1 minute.

verbose: ***bool, optional*** Whether to print repeatedly on each check or not.

Returns

None

3.7.2 nbs

class squid.jobs.nbs.**Job**(*name*, *process_handle=None*, *job_id=None*)
Job class to wrap simulations for queue submission.

Parameters

name: ***str*** Name of the simulation on the queue.

process_handle: ***process_handle, optional*** The process handle, returned by subprocess.Popen.

job_id: ***str, optional*** The job id. Usually this should be unique.

Returns

job_obj: **squid.jobs.nbs.Job** A Job object.

get_all_jobs()
Get a list of all jobs that are running and/or pending.

Parameters

detail: ***int, optional*** How much detail to get when finding jobs on the queue.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0:** ***list, str*** List of all jobs on the queue.
- ***details* =1:** ***list, tuple, str***

  List of all jobs on the queue as: (job name, time run, job status)
- ***details* =2:** ***list, tuple, str***

  List of all jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

squid.jobs.nbs.**get_job**(*s_flag*, *detail=0*)
Get a list of all jobs currently on your queue. From this, only return the values that have s_flag in them. The *detail* variable can be used to specify how much information you want returned.

Parameters

s_flag: ***str*** A string to parse out job information with.

detail: ***int, optional*** The amount of information you want returned.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0:** ***list, str*** List of all jobs on the queue.
- ***details* =1:** ***list, tuple, str***

  List of all jobs on the queue as: (job name, time run, job status)
- ***details* =2:** ***list, tuple, str***

  List of all jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

squid.jobs.nbs.**get_nbs_queues**()
Get a list of all available queues to submit a job to.

Returns

avail_queues: ***list, str*** A list of available queues by name.

squid.jobs.nbs.**submit_job**(*name*, *job_to_submit*, ***kwargs*)
Code to submit a simulation to the specified queue and queueing system.

Parameters

name: ***str*** Name of the job to be submitted to the queue.

job_to_submit: ***str*** String holding code you wish to submit.

queue: ***str, optional*** What queue to run the simulation on (queueing system dependent).

walltime: ***str, optional*** How long to post the job on the queue for in d-h:m:s where d are days, h are hours, m are minutes, and s are seconds. Default is for 30 minutes (00:30:00).

nprocs: ***int, optional*** How many processors to run the simulation on. Note, the actual number of cores mpirun will use is procs * ntasks.

sub_flag: ***str, optional*** Additional strings/flags/arguments to add at the end when we submit a job using jsub. That is: jsub demo.nbs sub_flag.

unique_name: ***bool, optional*** Whether to force the requirement of a unique name or not. NOTE! If you submit simulations from the same folder, ensure that this is True lest you have a redundancy problem! To overcome said issue, you can set redundancy to True as well (but only if the simulation is truly redundant).

outfile_name: ***str, optional*** Whether to give a unique output file name, or one based on the sim name.procs

xhosts: ***str*** **or** ***list, str, optional*** Which cpu to submit the job to.

email: ***str, optional*** An email address for sending job information to.

priority: ***int, optional*** What priority to give the submitted job.

sandbox: ***bool, optional*** Whether to sandbox the job or not.

redundancy: ***bool, optional*** With redundancy on, if the job is submitted and unique_name is on, then if another job of the same name is running, a pointer to that job will instead be returned.

Returns

job_obj: **`squid.jobs.nbs.Job`** A Job object.

3.7.3 queue_manager

`squid.jobs.queue_manager.`**`Job`**(*name*, ***kwargs*)

This function will return a job object depending on the queue system.

Parameters

name: ***str*** Name of the simulation on the queue.

process_handle: ***process_handle, optional*** The process handle, returned by subprocess.Popen.

job_id: ***str, optional*** The job id. Usually this should be unique.

Returns

jobContainer: **`squid.jobs.container.JobObject`** A job object, or a class built off of it, to handle job submission to a given queue manager.

`squid.jobs.queue_manager.`**`get_all_jobs`**(*detail=0*)

Get a list of all jobs currently on your queue. The *detail* variable can be used to specify how much information you want returned.

Parameters

detail: ***int, optional*** The amount of information you want returned.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0:** ***list, str*** List of all jobs on the queue.
- ***details* =1:** ***list, tuple, str***

  List of all jobs on the queue as: (job name, time run, job status)
- ***details* =2:** ***list, tuple, str***

  List of all jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

squid.jobs.queue_manager.**get_available_queues**()

Get a list of all available queues to submit a job to.

Returns

avail_queues: ***list, str*** A list of available queues by name.

squid.jobs.queue_manager.**get_pending_jobs**(*detail=0*)

Get a list of all jobs currently pending on your queue. The *detail* variable can be used to specify how much information you want returned.

Parameters

detail: ***int, optional*** The amount of information you want returned.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0:** ***list, str*** List of all pending jobs on the queue.
- ***details* =1:** ***list, tuple, str***

  List of all pending jobs on the queue as: (job name, time run, job status)
- ***details* =2:** ***list, tuple, str***

  List of all pending jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

squid.jobs.queue_manager.**get_queue_manager**()

This function will determine what the current queueing system is, and return relevant functionality.

Returns

queue_manager: ***str*** The name of the queue manager as either slurm, nbs, or None.

squid.jobs.queue_manager.**get_running_jobs**(*detail=0*)

Get a list of all jobs currently running on your queue. The *detail* variable can be used to specify how much information you want returned.

Parameters

detail: ***int, optional*** The amount of information you want returned.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0:** ***list, str*** List of all running jobs on the queue.
- ***details* =1:** ***list, tuple, str***

  List of all running jobs on the queue as: (job name, time run, job status)

- ***details* =2: *list, tuple, str***

  List of all running jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

3.7.4 slurm

class squid.jobs.slurm.**Job**(*name*, *process_handle=None*, *job_id=None*)

Job class to wrap simulations for queue submission.

Parameters

name: ***str*** Name of the simulation on the queue.

process_handle: ***process_handle, optional*** The process handle, returned by subprocess.Popen.

job_id: ***str, optional*** The job id. Usually this should be unique.

Returns

job_obj: **squid.jobs.slurm.Job** A Job object.

get_all_jobs()

Get a list of all jobs that are running and/or pending.

Parameters

detail: ***int, optional*** How much detail to get when finding jobs on the queue.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0: *list, str*** List of all jobs on the queue.
- ***details* =1: *list, tuple, str***

  List of all jobs on the queue as: (job name, time run, job status)
- ***details* =2: *list, tuple, str***

  List of all jobs on the queue as:

  (job name, time run, job status, queue, number of processors)

squid.jobs.slurm.**get_job**(*s_flag*, *detail=0*)

Get a list of all jobs currently on your queue. From this, only return the values that have s_flag in them. The *detail* variable can be used to specify how much information you want returned.

Parameters

s_flag: ***str*** A string to parse out job information with.

detail: ***int, optional*** The amount of information you want returned.

Returns

all_jobs: ***list*** Depending on *detail*, you get the following:

- ***details* =0: *list, str*** List of all jobs on the queue.
- ***details* =1: *list, tuple, str***

  List of all jobs on the queue as: (job name, time run, job status)
- ***details* =2: *list, tuple, str***

List of all jobs on the queue as:

(job name, time run, job status, queue, number of processors)

squid.jobs.slurm.**get_slurm_queues**()
Get a list of all available queues to submit a job to.

Returns

avail_queues: ***list, str*** A list of available queues by name.

squid.jobs.slurm.**submit_job**(*name*, *job_to_submit*, ***kwargs*)
Code to submit a simulation to the specified queue and queueing system.

Parameters

name: ***str*** Name of the job to be submitted to the queue.

job_to_submit: ***str*** String holding code you wish to submit.

queue: ***str, optional*** What queue to run the simulation on (queueing system dependent).

walltime: ***str, optional*** How long to post the job on the queue for in d-h:m:s where d are days, h are hours, m are minutes, and s are seconds. Default is for 30 minutes (00:30:00).

cpus_per_task: ***int, optional*** How many processors to run the simulation on. Note, the actual number of cores mpirun will use is nprocs * ntasks.

ntasks: ***int, optional*** How many processors to run the simulation on. Note, the actual number of cores mpirun will use is nprocs * ntasks.

nodes: ***int, optional*** How many nodes to run the simulation on.

sub_flag: ***str, optional*** Additional strings/flags/arguments to add at the end when we submit a job using sbatch. That is: sbatch demo.slurm sub_flag.

unique_name: ***bool, optional*** Whether to force the requirement of a unique name or not. NOTE! If you submit simulations from the same folder, ensure that this is True lest you have a redundancy problem! To overcome said issue, you can set redundancy to True as well (but only if the simulation is truly redundant).

outfile_name: ***str, optional*** Whether to give a unique output file name, or one based on the sim name.procs

allocation: ***str, optional*** The SLURM allocation to submit the job to.

jobarray: ***str, optional*** If specified, instead of indicating a range for job arrays, we will use these specific values. For example, jobarray=1,2,4,5 would submit jobs, but skip the 3rd index by name.

gpu: ***int, optional*** How many GPUs to use, if submitting to a GPU node.

redundancy: ***bool, optional*** With redundancy on, if the job is submitted and unique_name is on, then if another job of the same name is running, a pointer to that job will instead be returned.

Returns

job_obj: **squid.jobs.slurm.Job** A Job object.

3.7.5 submission

squid.jobs.submission.**pysub**(*name*, ***kwargs*)
Submission of python scripts to run on your queue.

Parameters

name: ***str*** Name of the python script (with or without the .py extension).

ompi_threads: ***int, optional*** The number OMP_NUM_THREADS should be manually assigned to.

preface_mpi: ***bool, optional*** Whether to run python via mpirun or not.

path: ***str, optional*** What directory your python script resides in. Note, this does NOT have a trailing /.

args: ***list, str, optional*** A list of arguments to pass to the python script on the queue.

jobarray: ***tuple, int, optional*** Specifies a job array of this python script should be run. In this case, the python script is submitted with a final argument corresponding to the index of the job array. NOTE - This will only work on SLURM.

modules: ***list, str, optional*** A list of modules to load prior to running this python script. Requires an installed version of lmod.

kwargs: **...** Any other keywords necessary for a given job submission script (NBS/SLURM). See the other submission sections for more details.

Returns

None

`squid.jobs.submission.`**`submit_job`**(*name*, *job_to_submit*, ***kwargs*)

Code to submit a simulation to the specified queue and queueing system.

Parameters

name: ***str*** Name of the job to be submitted to the queue.

job_to_submit: ***str*** String holding code you wish to submit.

kwargs: **...** Additional keyword arguments to NBS/SLURM for job submission. For more details, see the relevant section.

Returns

job_obj: **`squid.jobs.container.JobObject`** A Job object.

3.8 lammps

3.8.1 io.data

`squid.lammps.io.data.`**`write_lammps_data`**(*system*, ***kwargs*)

Writes a lammps data file from the given system.

Parameters

system: **`squid.structures.system.System`** Atomic system to be written to a lammps data file.

pair_coeffs_included: ***bool, optional*** Whether to write pair coefficients into the data file (True), or not (False).

Returns

None

3.8.2 io.dump

squid.lammps.io.dump.**read_dump**(*fptr, ext='.dump', coordinates=['x', 'y', 'z'], extras=[]*)
Function to read in a generic dump file. Currently it (1) requires element, x, y, z in the dump. You can also use xu, yu, and zu if the unwraped flag is set to True.

Due to individual preference, the extension was separated. Thus, if you dump to .xyz, have ext=".xyz", etc.

Parameters

fptr: ***str*** Name of the dump file with NO extension (ex. 'run' instead of 'run.dump'). This can also be a relative path. If no relative path is given, and the file cannot be found, it will default check in lammps/fptr/fptr+ext.

ext: ***str, optional*** The extension for the dump file. Note, this is default ".dump" but can be anything (ensure you have the ".").

coordinates: ***list, str, optional*** A list of strings describing how the coordinates are specified (x vs xs vs xu vs xsu)

extras: ***list, str, optional*** An additional list of things you want to read in from the dump file.

Returns

frames: ***list, list*** **squid.structures.atom.Atom** A list of lists, each holding atom structures.

squid.lammps.io.dump.**read_dump_gen**(*fptr, ext='.dump', coordinates=['x', 'y', 'z'], extras=[]*)
Function to read in a generic dump file. Currently it (1) requires element, x, y, z in the dump. You can also use xu, yu, and zu if the unwraped flag is set to True.

Due to individual preference, the extension was separated. Thus, if you dump to .xyz, have ext=".xyz", etc.

Parameters

fptr: ***str*** Name of the dump file with NO extension (ex. 'run' instead of 'run.dump'). This can also be a relative path. If no relative path is given, and the file cannot be found, it will default check in lammps/fptr/fptr+ext.

ext: ***str, optional*** The extension for the dump file. Note, this is default ".dump" but can be anything (ensure you have the ".").

coordinates: ***list, str, optional*** A list of strings describing how the coordinates are specified (x vs xs vs xu vs xsu)

extras: ***list, str, optional*** An additional list of things you want to read in from the dump file.

Returns

frames: ***list, list*** **squid.structures.atom.Atom** A list of lists, each holding atom structures.

3.8.3 io.thermo

3.8.4 job

squid.lammps.job.**get_lmp_obj**(*parallel=True*)
This function will find the lmp executable and a corresponding mpi executable. It will handle errors accordingly.

Parameters

parallel: ***bool, optional*** Whether to get corresponding mpiexec info or not.

Returns

lmp_path: ***str*** Path to a lammps executable.

mpi_path: ***str*** Path to an mpi executable.

`squid.lammps.job.`**`job`**(*run_name*, *input_script*, *system=None*, *queue=None*, *walltime='00:30:00'*, *nprocs=1*, *ntasks=1*, *nodes=1*, *email=None*, *pair_coeffs_in_data_file=True*, *no_echo=False*, *redundancy=False*, *unique_name=True*, *slurm_allocation=None*)

Wrapper to submitting a LAMMPs simulation.

Parameters

run_name: ***str*** Name of the simulation to be run.

input_script: ***str*** Input script for LAMMPs simulation.

system: **`squid.structures.system.System`** System object for our simulation.

queue: ***str, optional*** What queue to run the simulation on (queueing system dependent).

walltime: ***str, optional*** How long to post the job on the queue for in d-h:m:s where d are days, h are hours, m are minutes, and s are seconds. Default is for 30 minutes (00:30:00).

nprocs: ***int, optional*** How many processors to run the simulation on. Note, the actual number of cores mpirun will use is nprocs * ntasks.

ntasks: ***int, optional*** (For SLURM) The number of tasks this job will run, each task uses nprocs number of cores. Note, the actual number of cores mpirun will use is nprocs * ntasks.

nodes: ***int, optional*** (For SLURM) The number of nodes this job requires. If requesting ntasks * nprocs < 24 * nodes, a warning is printed, as on MARCC each node has only 24 cores.

email: ***str, optional*** An email address for sending job information to.

pair_coeffs_in_data_file: ***bool, optional*** Whether we have included the pair coefficients to be written to our lammps data file (True) or not (False).

no_echo: ***bool, optional*** Whether to pipe the terminal output to a file instead of printing.

redundancy: ***bool, optional*** With redundancy on, if the job is submitted and unique_name is on, then if another job of the same name is running, a pointer to that job will instead be returned.

unique_name: ***bool, optional*** Whether to force the requirement of a unique name or not. NOTE! If you submit simulations from the same folder, ensure that this is True lest you have a redundancy problem! To overcome said issue, you can set redundancy to True as well (but only if the simulation is truly redundant).

slurm_allocation: ***str, optional*** Whether to use a slurm allocation for this job or not. If so, specify the name.

Returns

job: **`squid.jobs.container.JobObject`** If running locally, return the process handle, else return the job container.

3.8.5 parser

The LMP_Parser code is code from the pizza.py toolkit (www.cs.sandia.gov/~sjplimp/pizza.html) developed by Steve Plimpton (sjplimp@sandia.gov) with some additional warning interspersed through the thermo output.

- *LMP_Parser*

class squid.lammps.parser.**LMP_Parser**(**list*)

Class object to assist in parsing lammps outputs.

Parameters

list: ***str*** Path to the lammps log file that is to be parsed. Note, several files can be included in this string as long as they are separated by spaces.

read_all: ***int, optional*** If this is set to 0, don't read in the whole file upon initialization. This lets you use the *next()* functionality.

Contains

nvec: ***int*** Number of vectors.

nlen: ***int*** Length of each vector.

names: ***list, str*** List of vector names.

ptr: ***dict*** Dictionary corresponding the thermo keys to which column of the output they reside in. ptr[thermo_key] = which column this data is in

data: ***list, list, float*** Raw data from file, organized into 2d array.

style: ***int*** What style the LAMMPs log file is in. 1 = multi, 2 = one, 3 = gran

firststr: ***str*** String that begins a thermo section in log file.

increment: ***int*** 1 if log file being read incrementally

eof: ***int*** ptr into incremental file for where to start next read

Returns

This *LMP_Parser* object.

get(**keys*)

Read specific values from thermo output.

Parameters

keys: ***str*** Which thermo outputs you want by ID. Not, this is as many requests as you want. ex. l.get("Time", "KE", ...)

Returns

vecs: ***list*** Desired outputs.

next()

Read the next line of thermo information from the file. Note, this is used when two arguments are passed during initialization.

Returns

timestep: ***int*** The timestep of the parsed thermo output.

write(*filename*, **keys*)

Write parsed vectors to a file.

Parameters

filename: ***str*** The name of the file you want to dump all your outputs to.

keys: ***str, optional*** Which specific vectors you want output to the file. ex. >>> l.write("file.txt","Time", "KE", ...)

Returns

None

3.9 maths

3.9.1 lhs

squid.maths.lhs.**create_lhs**(*N_points*, *N_samples*, *sample_bounds*, *params=None*)

Generate a latin hypercube sample for an n dimensional space specified by the sample_bounds keyword. An example is to call create_lhs to sample the lennard jones parameter space:

Assuming we want to sample 5 times parameters = create_lhs(2, 5, [(0, 10), (0, 10)])

Parameters

N_points: ***int*** The dimensionality of our system.

N_samples: ***int*** How many samples we want to do.

sample_bounds: ***list, tuple, float*** **or** ***list, list, int/float*** The min and max values for each dimension. Note, in special cases we may want to specify that a value is discrete from a list, or specifically an integer. Finally, if neither a list nor tuple is passed, we assume the value is static. Thus, all the following cases are allowed:

- [(0, 10), (3, 20), (-3, 2)]
- [3, [2, 3], (-5., 2.3, float)]

In the second case, the first parameter is set to 3, the second parameter is chosen as either 2 or 3, and the third parameter is force cast to a float.

params: ***list, str, optional*** Current return is a list of lists, each holding the randomly chosen N_points. However, by specifying params, the return can be made into a dictionary, with each point associated with the string in params. Note, this is one to one with the sample_bounds. That is, params[i] has the bounds specified by sample_bounds[i].

Returns

params: ***list, dict/list, float*** A list of lists, each holding a 1D array of points chosen from the LHC method. Note, if params was specified then instead a list of dictionaries is returned.

3.10 optimizers

3.10.1 bfgs

squid.optimizers.bfgs.**bfgs**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*)

A Broyden-Fletcher-Goldfarb-Shanno optimizer, overloaded for NEB use.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params, returns the gradient.

NEB_obj: **neb.NEB** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

step_size: ***float*** Step size to take.

step_size_adjustment: ***float*** A factor to adjust step_size when a bad step is made.

max_step: ***float*** A maximum allowable step length. If 0, any step is ok.

target_function: ***func*** A function that will help decide if backtracking is needed or not. This function will be used to verify BFGS is minimizing. If nothing is passed, but NEB_obj is not None, the NEB_obj.get_error function will be called.

armijo_line_search_factor: ***float*** A factor for the armijo line search.

linesearch: ***str*** Whether to use the *armijo* or *backtrack* linesearch method. If None is passed, a static step_size is used.

reset_when_in_trouble: ***bool*** Whether to reset the Hessian to Identity when bad steps have been taken.

reset_step_size: ***int*** How many iterations of 'good' steps to take before resetting step_size to its initial value.

N_reset_hess: ***int*** A hard reset to the hessian to be applied every N iterations.

start_hess: ***int, float, or matrix*** A starting matrix to use instead of the identity. If an integer or float is passed, then the starting hessian is a scaled identity matrix.

use_numopt_start: ***bool*** Whether to use the starting hessian guess laid out by Nocedal and Wright in the Numerical Operations textbook, page 178. H0 = (<y|s>) / (<y|y>) * I. If chosen, start_hess is set to the identity matrix.

accelerate: ***bool*** Whether to accelerate via increasing step_size by 1/step_size_adjustment when no bad steps are taken after *reset_step_size* iterations.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

dimensions: ***int*** The number of dimensions for the optimizer to run in. By default this is 3 (for NEB atomic coordinates.)

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.10.2 conjugate_gradient

squid.optimizers.conjugate_gradient.**conjugate_gradient**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*, *extra_args_gradient=None*, *extra_args_target=None*)

A conjugate gradient optimizer, overloaded for NEB use.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params and an abstract list of extra arguments, returns the gradient.

NEB_obj: **neb.NEB** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

step_size: ***float*** Step size to take.

step_size_adjustment: ***float*** A factor to adjust step_size when a bad step is made.

method: ***str*** Whether to use the Fletcher-Reeves (FR) method of calculating beta, or the Polak-Ribiere (PR) method.

max_step: ***float*** A maximum allowable step length. If 0, any step is ok.

target_function: ***func*** A function that will help decide if backtracking is needed or not. This function will be used to verify BFGS is minimizing. If nothing is passed, but NEB_obj is not None, the NEB_obj.get_error function will be called.

armijo_line_search_factor: ***float*** A factor for the armijo line search.

linesearch: ***str*** Whether to use the *armijo* or *backtrack* linesearch method. If None is passed, a static step_size is used.

reset_step_size: ***int*** How many iterations of 'good' steps to take before resetting step_size to its initial value.

accelerate: ***bool*** Whether to accelerate via increasing step_size by 1/step_size_adjustment when no bad steps are taken after *reset_step_size* iterations.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

dimensions: ***int*** The number of dimensions for the optimizer to run in. By default this is 3 (for NEB atomic coordinates.)

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.10.3 fire

squid.optimizers.fire.**fire**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*)

A FIRE optimizer, overloaded for NEB use.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params, returns the gradient.

NEB_obj: **neb.NEB** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

dt: ***float*** Time step size to take.

dtmax: ***float, optional*** The maximum dt allowed.

max_step: ***float*** The maximum step size to take.

Nmin: ***int, optional*** The minimum number of steps before acceleration occurs.

finc: ***float, optional*** The factor by which dt increases.

fdec: ***float, optional*** The factor by which dt decreases.

astart: ***float, optional*** The starting acceleration.

fa: ***float, optional*** The factor by which the acceleration is scaled.

viscosity: ***float*** The viscosity within a verlet step (used if euler is False).

euler: ***bool*** Whether to make an euler step or not.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.10.4 lbfgs

squid.optimizers.lbfgs.**lbfgs**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*, *extra_args_gradient=None*, *extra_args_target=None*)

A Limited Memory Broyden-Fletcher-Goldfarb-Shanno optimizer, overloaded for NEB use.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params, returns the gradient.

NEB_obj: **`neb.NEB`** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

step_size: ***float*** Step size to take.

step_size_adjustment: ***float*** A factor to adjust step_size when a bad step is made.

max_step: ***float*** A maximum allowable step length. If 0, any step is ok.

max_steps_remembered: ***int*** The maximum number of previous iterations to save.

target_function: ***func*** A function that will help decide if backtracking is needed or not. This function will be used to verify LBFGS is minimizing. If nothing is passed, but NEB_obj is not None, the NEB_obj.get_error function will be called.

armijo_line_search_factor: ***float*** A factor for the armijo line search.

linesearch: ***str*** Whether to use the *armijo* or *backtrack* linesearch method. If None is passed, a static step_size is used.

reset_when_in_trouble: ***bool*** Whether to reset the stored parameters and gradients when a bad step has been taken.

reset_step_size: ***int*** How many iterations of 'good' steps to take before resetting step_size to its initial value.

N_reset_hess: ***int*** A hard reset to the hessian to be applied every N iterations.

start_hess: ***int, float, or matrix*** A starting integer or float to scale the starting hessian.

use_numopt_start: ***bool*** Whether to use the starting hessian guess laid out by Nocedal and Wright in the Numerical Operations textbook, page 178. H0 = (<y|s>) / (<y|y>) * I. If chosen, start_hess is set to the identity matrix.

accelerate: ***bool*** Whether to accelerate via increasing step_size by 1/step_size_adjustment when no bad steps are taken after *reset_step_size* iterations.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

dimensions: ***int*** The number of dimensions for the optimizer to run in. By default this is 3 (for NEB atomic coordinates.)

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.10.5 quick_min

squid.optimizers.quick_min.**quick_min**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*)

A quick min optimizer, overloaded for NEB use. Note, this will ONLY work for use within the NEB code.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params, returns the gradient.

NEB_obj: **neb.NEB** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

dt: ***float*** Time step size to take.

max_step: ***float*** The maximum step size to take.

viscosity: ***float*** The viscosity within a verlet step (used if euler is False).

euler: ***bool*** Whether to make an euler step or not.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

verbose: ***bool*** Whether to have additional output.

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.10.6 steepest_descent

squid.optimizers.steepest_descent.**steepest_descent**(*params*, *gradient*, *NEB_obj=None*, *new_opt_params={}*, *extra_args_gradient=None*, *extra_args_target=None*)

A steepest descent optimizer, overloaded for NEB use.

Parameters

params: ***list, float*** A list of parameters to be optimized.

gradient: ***func*** A function that, given params, returns the gradient.

NEB_obj: **neb.NEB** An NEB object to use.

new_opt_params: ***dict*** A dictionary holding any changes to the optimization algorithm's parameters. This includes the following -

step_size: ***float*** Step size to take.

step_size_adjustment: ***float*** A factor to adjust step_size when a bad step is made.

max_step: ***float*** A maximum allowable step length. If 0, any step is ok.

target_function: ***func*** A function that will help decide if backtracking is needed or not. This function will be used to verify BFGS is minimizing. If nothing is passed, but NEB_obj is not None, the NEB_obj.get_error function will be called.

armijo_line_search_factor: ***float*** A factor for the armijo line search.

linesearch: ***str*** Whether to use the *armijo* or *backtrack* linesearch method. If None is passed, a static step_size is used.

reset_when_in_trouble: ***bool*** Whether to reset the Hessian to Identity when bad steps have been taken.

reset_step_size: ***int*** How many iterations of 'good' steps to take before resetting step_size to its initial value.

accelerate: ***bool*** Whether to accelerate via increasing step_size by 1/step_size_adjustment when no bad steps are taken after *reset_step_size* iterations.

maxiter: ***int*** Maximum number of iterations for the optimizer to run. If None, then the code runs indefinitely.

g_rms: ***float*** The RMS value for which to optimize the gradient to.

g_max: ***float*** The maximum gradient value to be allowed.

fit_rigid: ***bool*** Remove erroneous rotation and translations during NEB.

dimensions: ***int*** The number of dimensions for the optimizer to run in. By default this is 3 (for NEB atomic coordinates.)

callback: ***func, optional*** A function to be run after each optimization loop.

Returns

params: ***list, float*** A list of the optimized parameters.

code: ***int*** An integer describing how the algorithm converged. This can be identified in the constants file.

iters: ***int*** The number of iterations the optimizer ran for.

3.11 orca

3.11.1 io

`squid.orca.io.`**`engrad_read`**(*input_file*, *force='Ha/Bohr'*, *pos='Bohr'*)

General read in of all possible data from an Orca engrad file (.orca.engrad).

Parameters

input_file: ***str*** Orca .orca.engrad file to be parsed.

force: ***str, optional*** Units you want force to be returned in. Default is Ha/Bohr.

pos: ***str, optional*** Units you want position to be returned in. Default is Bohr.

Returns

atoms: ***list,*** **squid.structures.atom.Atom** A list of the final atomic state, with forces appended to each atom.

energy: ***float*** The total energy of this simulation.

squid.orca.io.**read**(*input_file*)

General read in of all possible data from an Orca output file (.out). It should be mentioned that atomic positions are 0 indexed.

Parameters

input_file: ***str*** Orca .out file to be parsed.

Returns

data: **squid.structures.results.DFT_out** Generic DFT output object containing all parsed results.

3.11.2 job

squid.orca.job.**job**(*run_name*, *route=None*, *atoms=[]*, *extra_section=''*, *grad=False*, *queue=None*, *walltime='00:30:00'*, *sandbox=False*, *nprocs=1*, *ntasks=1*, *nodes=1*, *charge=0*, *multiplicity=1*, *redundancy=False*, *use_NBS_sandbox=False*, *unique_name=True*, *previous=None*, *mem=2000*, *priority=None*, *xhost=None*, *slurm_allocation=None*)

Wrapper to submitting an Orca simulation.

Parameters

run_name: ***str*** Name of the simulation to be run.

route: ***str, optional*** The DFT route line, containing the function, basis set, etc. Note, if route=None and previous != None, the route from the previous simulation will be used instead.

atoms: ***list,*** **squid.structures.atom.Atom**,***optional*** A list of atoms for the simulation. If this is an empty list, but previous is used, then the last set of atomic coordinates from the previous simulation will be used.

extra_section: ***str, optional*** Additional DFT simulation parameters. If None and previous is not None, then previous extra section is used.

grad: ***bool, optional*** Whether to force RunTyp Gradient.

queue: ***str, optional*** What queue to run the simulation on (queueing system dependent).

walltime: ***str, optional*** The walltime the job is given when submitted to a queue. Format is in day-hr:min:sec.

sandbox: ***bool, optional*** Whether to run the job in a sandbox or not.

nprocs: ***int, optional*** How many processors to run the simulation on. Note, the actual number requested by orca will be nprocs * ntasks.

ntasks: ***int, optional*** (For SLURM) The number of tasks this job will run, each task uses nprocs number of cores. Note, the actual number requested by orca will be nprocs * ntasks.

nodes: ***int, optional*** (For SLURM) The number of nodes this job requires. If requesting ntasks * nprocs < 24 * nodes, a warning is printed, as on MARCC each node has only 24 cores.

charge: ***int, optional*** Charge of the system. The default charge of 0 is used.

multiplicity: ***int, optional*** Multiplicity of the system. The default multiplicity of 1 is used. Recall, multiplicity M = 2*S + 1 where S is the total system spin.

redundancy: ***bool, optional*** With redundancy on, if the job is submitted and unique_name is on, then if another job of the same name is running, a pointer to that job will instead be returned.

use_NBS_sandbox: ***bool, optional*** Whether to use the NBS sandboxing headers (True), or manually copy files (False).

unique_name: ***bool, optional*** Whether to force the requirement of a unique name or not. NOTE! If you submit simulations from the same folder, ensure that this is True lest you have a redundancy problem! To overcome said issue, you can set redundancy to True as well (but only if the simulation is truly redundant).

previous: ***str, optional*** Name of a previous simulation for which to try reading in information using the MORead method.

mem: ***float, optional*** Amount of memory per processor that is available (in MB).

priority: ***int, optional*** Priority of the simulation (queueing system dependent). Priority ranges (in NBS) from a low of 1 (start running whenever) to a high of 255 (start running ASAP).

xhost: ***list, str or str, optional*** Which processor to run the simulation on(queueing system dependent).

slurm_allocation: ***str, optional*** Whether to use a slurm allocation for this job or not. If so, specify the name.

Returns

job: **`squid.jobs.container.JobObject`** Teturn the job container.

`squid.orca.job.`**`jobarray`**(*run_name*, *route*, *frames*, *n_frames=None*, *extra_section=''*, *grad=False*, *queue=None*, *walltime='00:30:00'*, *sandbox=False*, *nprocs=1*, *ntasks=1*, *nodes=1*, *charge=0*, *multiplicity=1*, *redundancy=False*, *unique_name=True*, *previous=None*, *mem=2000*, *priority=None*, *xhost=None*, *jobarray_values=None*, *slurm_allocation=None*, *batch_serial_jobs=None*)

Wrapper to submitting various Orca simulations as a job array on a SLURM system. This is used when there are many atomic systems, stored in a list, that need to have the same DFT calculation performed on each.

Note - When requesting nprocs/ntasks/nodes, these will be per-job. As such, do **NOT** multiply out. For instance, if you request ntasks=4, and len(frames) = 10, you will be running 10 jobs, each with 4 tasks.

Parameters

run_name: ***str*** Name of the simulation to be run.

route: ***str*** The DFT route line, containing the function, basis set, etc. Note, if route=None and previous != None, the route from the previous simulation will be used instead.

frames: ***list,*** **`squid.structures.atom.Atom`** Each atomic system that needs to be simulated.

n_frames: ***int, optional*** The number of frames.

extra_section: ***str, optional*** Additional DFT simulation parameters. If None and previous is not None, then previous extra section is used.

grad: ***bool, optional*** Whether to force RunTyp Gradient.

queue: ***str, optional*** What queue to run the simulation on (queueing system dependent).

walltime: ***str, optional*** The walltime the job is given when submitted to a queue. Format is in day-hr:min:sec.

sandbox: ***bool, optional*** Whether to run the job in a sandbox or not.

nprocs: ***int, optional*** How many processors to run the simulation on. Note, the actual number requested by orca will be nprocs * ntasks.

ntasks: ***int, optional*** (For SLURM) The number of tasks this job will run, each task uses nprocs number of cores. Note, the actual number requested by orca will be nprocs * ntasks.

nodes: ***int, optional*** (For SLURM) The number of nodes this job requires. If requesting ntasks * nprocs < 24 * nodes, a warning is printed, as on MARCC each node has only 24 cores.

charge: ***float, optional*** Charge of the system. If this is used, then charge_and_multiplicity is ignored. If multiplicity is used, but charge is not, then default charge of 0 is chosen.

multiplicity: ***int, optional*** Multiplicity of the system. If this is used, then charge_and_multiplicity is ignored. If charge is used, but multiplicity is not, then default multiplicity of 1 is chosen.

redundancy: ***bool, optional*** With redundancy on, if the job is submitted and unique_name is on, then if another job of the same name is running, a pointer to that job will instead be returned.

unique_name: ***bool, optional*** Whether to force the requirement of a unique name or not. NOTE! If you submit simulations from the same folder, ensure that this is True lest you have a redundancy problem! To overcome said issue, you can set redundancy to True as well (but only if the simulation is truly redundant).

previous: ***str, optional*** Name of a previous simulation for which to try reading in information using the MORead method.

mem: ***float, optional*** Amount of memory per processor that is available (in MB).

priority: ***int, optional*** Priority of the simulation (queueing system dependent). Priority ranges (in NBS) from a low of 1 (start running whenever) to a high of 255 (start running ASAP).

xhost: ***list, str or str, optional*** Which processor to run the simulation on(queueing system dependent).

jobarray_values: ***str, optional*** If specified, instead of indicating a range for job arrays, we will use these specific values. For example, jobarray_values=1,2,4,5 would submit jobs, but skip the 3rd index by name.

slurm_allocation: ***str, optional*** Whether to use a slurm allocation for this job or not. If so, specify the name.

batch_serial_jobs: ***int, optional*** Whether to batch jobs at N at a time (locally on serial job submission).

Returns

job: **`squid.jobs.container.JobObject`** Teturn the job container.

3.11.3 mep

(c) 2013 Marius Retegan License: BSD-2-Clause Description: Create a .cube file of the electrostatic potential using ORCA. Run: python mep.py fname npoints (e.g. python mep.py water 40) Arguments: fname - file name without the extension;

this should be the same for the .gbw and .scfp.

npoints - number of grid points per side (80 should be fine)

Dependencies: numpy

Source: https://gist.github.com/mretegan/5501553

Notes: Slight modifications made to incorporate into Squid.

Disclaimer: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

squid.orca.mep.**electrostatic_potential_cubegen**(*fname*, *npoints=80*)

Given the name of a simulation, generate the cube file.

Parameters

fname: ***str*** The orca simulation name.

npoints: ***int, optional*** How fine the grid should be. Larger, more fine, more expensive to calculate.

Returns

None

3.11.4 post_process

squid.orca.post_process.**gbw_to_cube**(*name*, *mo*, *spin=0*, *grid=40*)

Pipe in flags to orca_plot to generate a cube file for the given molecular orbital. Note, this is assumed to be running from the parent directory (ie, gbw is in the orca/BASENAME/BASENAME.orca.gbw).

Parameters

name: ***str*** The base name of the gbw file. Thus, 'water' instead of 'water.orca.gbw'.

mo: ***int*** Which molecular orbital to generate the cube file for. Note, this is 0 indexed.

spin: ***int, optional*** Whether to plot the alpha or beta (0 or 1) operator.

grid: ***int, optional*** The grid resolution, default being 40.

Returns

mo_name: ***str*** The name of the output MO file.

squid.orca.post_process.**mo_analysis**(*name*, *orbital=None*, *HOMO=True*, *LUMO=True*, *wireframe=True*, *hide=True*, *iso=0.04*)

Post process an orca job using orca_plot and vmd to display molecular orbitals and the potential surface. NOTE! By default Orca does not take into account degenerate energy states when populating. To do so, ensure the following is in your extra_section:

'%scf FracOcc true end'.

Parameters

name: ***str*** Orca file name. Only use the name, such as 'water' instead of 'water.gbw'. Note, do not pass a path as it is assumed you are in the parent directory of the job to analyze. If not, use the path variable.

orbital: ***list, int, optional*** **or** ***int, optional*** The orbital(s) to analyze (0, 1, 2, 3, ...). By default HOMO and LUMO will be analyzed, thus this only is useful if you wish to see other orbitals.

HOMO: ***bool, optional*** If you want to see the HOMO level.

LUMO: ***bool, optional*** If you want to see the LUMO level.

wireframe: ***bool, optional*** If you want to view wireframe instead of default surface.

hide: ***bool, optional*** Whether to have the representations all off by or not when opening.

iso: ***float, optional*** Isosurface magnitude. Set to 0.04 by default, but 0.01 may be better.

Returns

None

`squid.orca.post_process.`**`pot_analysis`**(*name*, *wireframe=True*, *npoints=80*)

Post process an orca job using orca_plot and vmd to display the electrostatic potential mapped onto the electron density surface.

Parameters

name: ***str*** Orca file name. Only use the name, such as 'water' instead of 'water.gbw'. Note, do not pass a path as it is assumed you are in the parent directory of the job to analyze.

wireframe: ***bool, optional*** If you want to view wireframe instead of default surface.

npoints: ***int, optional*** The grid size for the potential surface.

Returns

None

3.11.5 utils

`squid.orca.utils.`**`get_orca_obj`**(*parallel=True*)

This function will find the orca executable and the corresponding openmpi executable. It will handle errors accordingly.

Parameters

parallel: ***bool, optional*** Whether we guarantee the relevant parallel openmpi is setup (True) or not (False).

Returns

orca_path: ***str*** The path to the orca executable.

3.12 post_process

3.12.1 debyer

Python hooks for the debyer code. Link: https://debyer.readthedocs.io/en/latest/

- `get_pdf()`

`squid.post_process.debyer.`**`get_pdf`**(*frames*, *start=0.0*, *stop=5.0*, *step=0.1*, *cutoff=10.0*, *rho=1.0*, *quanta=0.001*, *output=None*, *persist=False*)

Obtain the pair distribution function of a list of atoms using the Debyer code.

Parameters

frames: ***str or list,*** **squid.structures.atom.Atom** An xyz file name (with or without the .xyz extension) or an input frame to calculate the pdf for.

start: ***float, optional*** The starting radial distance in Angstroms for the calculated pattern.

stop: ***float, optional*** The ending radial distance in Angstroms for the calculated pattern.

step: ***float, optional*** Step in Angstroms for the calculated pattern.

cutoff: ***float, optional*** Cutoff distance in Angstroms for Interatomic Distance (ID) calculations.

rho: ***float, optional*** Numeric density of the system.

quanta: ***float, optional*** Interatomic Distance (ID) discritization quanta.

output: ***str, optional*** Output file name with NO extension given

persist: ***bool, optional*** Whether to persist made .g and .xyz files (True), or remove them (False)

Returns

pdf: ***list, tuple, float*** A list of tuples holding the pdf data (distance in Angstroms and Intensity).

References

- https://debyer.readthedocs.io/en/latest/

3.12.2 ovito

The visualization module automates some visualization procedures for post processing data.

An example of using this is as follows:

```
import files
import visualization as vis

vis.ovito_xyz_to_gif(files.read_xyz("CNH_HCN.xyz"), "/fs/home/hch54/tmp", renderer=
↪'Tachyon')
```

- `ovito_xyz_to_image()`
- `ovito_xyz_to_gif()`

`squid.post_process.ovito.`**`get_ovito_obj`**(*version='2.9.0'*)

This function returns the ovito object. Note, currently the code below only works on version 2.9.0.

`squid.post_process.ovito.`**`ovito_xyz_to_gif`**(*frames*, *scratch*, *fname='image'*, *camera_pos=(10, 0, 0)*, *camera_dir=(-1, 0, 0)*, *size=(800, 600)*, *delay=10*, *display_cell=False*, *renderer='OpenGLRenderer'*, *renderer_settings={}*, *overwrite=False*)

This function will, using the ovito python api, generate either a single image or a gif of the input frames. Note, a gif is only generated when more than one frame exists.

Parameters

frames: ***str*** **or** ***list,*** **squid.structures.atom.Atom** A list of frames you wish to generate an image for, or a path to an xyz file.

scratch: ***str*** A directory you want to have each image saved to.

fname: ***str, optional*** The prefix for the image names.

camera_pos: ***tuple, float, optional*** A tuple of x, y, and z coordinates for the camera to be positioned.

camera_dir: ***tuple, float, optional*** The direction the camera is facing.

size: ***tuple, int, optional*** Image size (width, height).

delay: ***int, optional*** In the event of a gif, how long it should play for.

display_cell: ***bool, optional*** Whether to display the box around the system or not.

renderer: ***str, optional*** What kind of renderer you wish to use: OpenGL or Tachyon.

renderer_settings: ***dict, optional*** Here you can change specific renderer settings.

overwrite: ***bool, optional*** Whether to delete any files already existing in the scratch dir.

Returns

None

squid.post_process.ovito.**ovito_xyz_to_image**(*xyz*, *scratch*, *fname='image'*, *camera_pos=(10, 0, 0)*, *camera_dir=(-1, 0, 0)*, *size=(800, 600)*, *renderer='OpenGLRenderer'*, *display_cell=False*, *renderer_settings={}*)

This function will, using the ovito python api, generate a png image of an xyz file.

Parameters

xyz: ***str*** A path to an xyz file.

scratch: ***str*** A directory you want to have each image saved to.

fname: ***str, optional*** The prefix for the image names.

camera_pos: ***tuple, float, optional*** A tuple of x, y, and z coordinates for the camera to be positioned.

camera_dir: ***tuple, float, optional*** The direction the camera is facing.

size: ***tuple, int, optional*** Image size (width, height).

delay: ***int, optional*** In the event of a gif, how long it should play for.

renderer: ***str, optional*** What kind of renderer you wish to use: OpenGL or Tachyon.

display_cell: ***bool, optional*** Whether to display the box around the system or not.

renderer_settings: ***dict, optional*** Here you can change specific renderer settings.

Returns

None

3.12.3 vmd

The vmd package automates various vmd post-processing tasks.

- *plot_MO_from_cube()*

---

squid.post_process.vmd.**plot_MO_from_cube**(*fptrs*, *wireframe=True*, *hide=True*, *iso=0.04*)

A function to generate a VMD visualization of a molecular orbital from a cube file.

Parameters

fptrs: ***list, str, or str*** Strings giving the path to the cube file.

wireframe: ***bool, optional*** If you want to view wireframe (True) or not (False) for the orbitals.

hide: ***bool, optional*** Whether to hide the representations (True) on startup, or not (False).

iso: ***float, optional*** Isosurface magnitude. Set to 0.04 by default, but 0.01 may be better.

Returns

None

squid.post_process.vmd.**plot_electrostatic_from_cube**(*fptr_rho*, *fptr_pot*, *wireframe=True*)

A function to generate a VMD visualization of a electrostatic potential mapped onto an electron density isosurface.

Parameters

fptr_rho: ***str*** Path to the electron density cube file.

fptr_pot: ***str*** Path to the electrostatic potential cube file.

wireframe: ***bool, optional*** If you want to view wireframe (True) or not (False) for the orbitals.

Returns

None

3.13 qe

TO DO

3.14 structures

3.14.1 atom

The atom object holds atomic information in and transformations.

- `Atom`

---

class squid.structures.atom.**Atom**(*element*, *x*, *y*, *z*, *index=None*, *molecule_index=1*, *label=None*, *charge=None*)

A structure to hold atom information.

Parameters

element: ***str*** The atomic element.

x: ***float*** The x coordinate of the atom.

y: ***float*** The y coordinate of the atom.

z: ***float*** The z coordinate of the atom.

index: ***int, optional*** The atomic index within a molecule.

molecule_index: ***int, optional*** Which molecule the atom is contained in.

label: ***str, optional*** The label of the atomic type within the given forcefield.

charge: ***float, optional*** The atomic charge.

Returns

atom: `squid.structures.atom.Atom` The Atom class container.

`flatten`()
Obtain simplified position output.

Returns

pos: *np.array, float* A numpy array holding the x, y, and z position of this atom.

`get_id_tag`()
Return the id tag of this atom. This is defined as:

molecule_index:index

Note, if either is None then it is returned as the string. Thus:

None:1 1:None None:None

Are all possible.

`scale`(*v*)
Scale the atom by a vector. This can be useful if we want to change coordinate systems.

Parameters

v: *list, float* A vector of 3 floats specifying the x, y, and z scalars to be applied.

Returns

None

`set_position`(*pos*)
Manually set the atomic positions by passing a tuple/list.

Parameters

pos: *list, float or tuple, float* A vector of 3 floats specifying the new x, y, and z coordinate.

Returns

None

`translate`(*v*)
Translate the atom by a vector.

Parameters

v: *list, float* A vector of 3 floats specifying the x, y, and z offsets to be applied.

Returns

None

`unravel`()
Like flatten; however, this method will unravel all properties of the atom into a tuple.

Returns

props: *tuple, ...* Return all atom properties as they are, in the following order:

element, x, y, z, index, molecule_index, label, charge

3.14.2 molecule

- Molecule

---

class squid.structures.molecule.**Molecule**(*atoms*, *bonds=[]*, *angles=[]*, *dihedrals=[]*, *molecule_index=None*, *assign_indices=True*)

A molecule object to store atoms and any/all associated interatomic connections.

Parameters

atoms: ***list,*** **squid.structures.atom.Atom** A list of atoms.

bonds: ***list,*** **squid.structures.topology.Connector,** ***optional*** A list of all bonds within the system.

angles: ***list,*** **squid.structures.topology.Connector,** ***optional*** A list of all angles within the system.

dihedrals: ***list,*** **squid.structures.topology.Connector,** ***optional*** A list of all dihedrals within the system.

molecule_index: ***int, optional*** The index to be assigned for this molecule.

assign_indices: ***bool, optional*** Whether to assign the molecule a default index of 1 (if no other is specified), and to assign the atomic indices, indexed at 1. If molecule_index is specified, that will take precedence over the default of 1; however, the atom re-indexing will still take place.

Returns

molecule: **squid.structures.molecule.Molecule** The Molecule class container.

assign_angles_and_dihedrals()

Given a list of atom structures with bonded information, calculate angles and dihedrals.

Returns

None

flatten()

Flatten out all atoms into a 1D array.

Returns

atoms: ***list, float*** A 1D array of atomic positions.

get_center_of_geometry(*skip_H=False*)

Calculate the center of geometry of the molecule.

Parameters

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

cog: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of geometry.

get_center_of_mass(*skip_H=False*)

Calculate the center of mass of the molecule.

Parameters

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

com: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of mass.

merge(*other*, *deepcopy=False*)
This function merges another molecule into this one, offsetting indices as needed. When merging, the atom indices of this molecule is reassigned. Futher, if deepcopy is False, then the atom indices of the other molecule are also reassigned.

Parameters

deepcopy: ***bool, optional*** Whether to merge via a deep copy, in which atoms are replicated (that is, the atom pointers are different between the molecules), or to merge via pointers, in which the other molecule has similar pointers.

Returns

None

net_charge()
Return the net charge of the molecule. This requires that atoms have charges associated with them.

Returns

charge: ***float*** The atomic charge of the system.

reassign_indices(*offset=0*)
Simply reassign atomic indices based on their current positions in the atoms array. This is zero indexed, unless otherwise specified. Further, assign the molecule_index of the atoms to be the same as this molecule object.

Parameters

offset: ***int, optional*** What offset to use when indexing.

Returns

None

rotate(*m*, *around='com'*)
Rotate the molecule by the given matrix *m*.

Parameters

m: ***list, list, float*** A 3x3 matrix describing the rotation to be applied to this molecule.

around: ***str, optional*** Whether to rotate around the center of mass (com), center of geometry (cog), or neither ("None" or None).

Returns

None

scale(*v*)
Apply a scalar to this molecule.

Parameters

v: ***list, float*** A vector of 3 floats specifying the x, y, and z scalars to be applied.

Returns

None

set_positions(*positions*, *new_atom_list=False*)
Manually specify atomic positions of your molecule.

Parameters

positions: ***list, float*** A list, either 2D or 1D, of the atomic positions. Note, this should be in the same order that the atoms are stored in.

new_atom_list: ***bool, optional*** Whether to generate an entirely new atom list (True) or re-write atom positions of those atoms already stored (False). Note, if a new list is written, connections are wiped out.

Returns

None

translate(*v*)

Apply a translation to this molecule.

Parameters

v: ***list, float*** A vector of 3 floats specifying the x, y, and z offsets to be applied.

Returns

None

3.14.3 results

The results module contains data structures to hold simulation output.

- DFT_out
- sim_out

---

class squid.structures.results.**DFT_out**(*name*, *dft='orca'*)

A generic class to hold dft data.

Parameters

name: ***str*** Given name for this simulation object.

dft: ***str, optional*** Identifier for which dft code this data is from.

Contains

route: ***str*** The 'route' line describing the functional, basis set, and other dft configurations.

extra_section: ***str*** The 'extra section' in the simulation.

charge_and_multiplicity: ***str*** The charge and multiplicity, in that order, of the system.

frames: ***list, list,*** **squid.structures.atom.Atom** A list lists of atoms describing each iteration in the dft simulation.

atoms: ***list,*** **squid.structures.atom.Atom** Atomic information of the last iteration in the dft simulation.

gradients: ***list, list, float*** Gradient of the potential, stored for each atom in *atoms* and *frames[-1]*.

energy: ***float*** The total energy of the last iteration.

charges_MULLIKEN: ***list, float*** Mulliken charges for each atom in *atoms* and *frames[-1]*.

charges_LOEWDIN: ***list, float*** Loewdin charges for each atom in *atoms* and *frames[-1]*.

charges_CHELPG: ***list, float*** Chelpg charges for each atom in *atoms* and *frames[-1]*.

charges: ***list, float*** Charges for each atom in *atoms* and *frames[-1]*. Typically a copy of Mulliken charges.

MBO: ***list, list,*** **squid.structures.atom.Atom**, ***float*** A list of lists, each list holding (1) a list of atoms in the bond and (2) the Mayer Bond Order (MBO) of said bond.

vibfreq: ***list, float*** A list of the vibrational frequencies if available, otherwise None.

convergence: ***list, str*** **VERIFY** A list of convergence criteria and matching values.

converged: ***bool*** Whether the simulation converged (True), or not (False).

time: ***float*** Total time in seconds that the simulation ran for.

bandgap: ***float*** Bandgap of the final configuration.

bandgaps: ***float*** Bandgap of each configuration.

orbitals: ***list, tuple, float, float*** A list of tuples, each holding the information of the occupation and energy (Ha) of a molecular orbital. NOTE! This does not take into account degenerate energy states, so ensure that whatever DFT software you're using has already done so.

finished: ***bool*** Whether the simulation completed normally (True), or not (False).

warnings: ***list, str*** Warnings output by the simulation.

class squid.structures.results.**sim_out**(*name*, *program='lammps'*)

A generic class to hold simulation data, particularly lammps trajectory files.

Parameters

name: ***str*** Given name for this simulation object.

program: ***str, optional*** Identifier for which program this data is from.

Contains

frames: ***list, list,*** **squid.structures.atom.Atom** A list lists of atoms describing each iteration in the simulation.

atoms: ***list,*** **squid.structures.atom.Atom** Atomic information of the last iteration in the simulation.

timesteps: ***list, int*** Recorded timesteps within the output.

final_timestep: ***int*** Final timestep of the output.

atom_counts: ***list, int*** List of how many atoms for each timestep.

atom_count: ***int*** List of how many atoms in the final timestep.

box_bounds_list: ***list, dict*** List of box bounds for each timestep.

box_bounds: ***dict*** List of box bounds for the final timestep.

3.14.4 system

class squid.structures.system.**System**(*name*, *box_size=(10.0, 10.0, 10.0)*, *box_angles=(90.0, 90.0, 90.0)*, *periodic=False*)

A system object to store molecules for one's simulations.

Parameters

name: ***str*** System Name. This is used when any files/folders are generated.

box_size: ***tuple, float, optional*** System x, y, and z lengths.

box_angles: ***tuple, float, optional*** System xy, yz, and xz angles in degrees.

periodic: ***bool, optional*** Whether to have periodic boundaries on or off.

Returns

system: `squid.structures.system.System` The System class container.

`add`(*molecule*, *mol_offset=1*, *deepcopy=True*)

A function to add a molecule to this system. Note, this addition can be either a deepcopy or not. If it is not a deepcopy, then the molecule is added as a pointer and can be adjusted externally. By default it is added via a deepcopy to prevent untracked errors.

Parameters

molecule: `squid.structures.molecule.Molecule` A Molecule structure.

mol_offset: *int, optional* The offset to apply to molecule_index.

deepcopy: *bool, optional* Whether to add the molecule into the system via a deepcopy or not.

Returns

None

`contains_molecule`(*molecule*)

Check if this system contains a molecule, based on the atoms, bonds, angles and dihedrals.

Parameters

molecule: `squid.structures.molecule.Molecule` A molecule to be checked if it resides within this system.

Returns

is_contained: *bool* A boolean specifying if the molecule passed to this function is contained within this System object. This implies that all atoms, bonds, angles, and dihedrals within the molecule are present in a molecule within the system.

`dump_angles_data`()

This function will dump all the bond information to a LAMMPS data file.

bond_index bond_type_index atoms...atoms

Returns

connection_info: *str* A string of all the connector information with proper data.

`dump_atoms_data`()

This function will dump all the atom information to a LAMMPS data file.

atom_index mol_index type charge x y z

Returns

atomic_info: *str* A string of all the atomic information with proper data.

`dump_bonds_data`()

This function will dump all the bond information to a LAMMPS data file.

bond_index bond_type_index atoms...atoms

Returns

connection_info: *str* A string of all the connector information with proper data.

`dump_dihedrals_data`()

This function will dump all the bond information to a LAMMPS data file.

bond_index bond_type_index atoms...atoms

Returns

connection_info: *str* A string of all the connector information with proper data.

dump_pair_coeffs()
Will try to dump all available pair coefficients. Currently, this means that Coulomb, Lennard-Jones, and Morse will be attempted. If you prefer that one or another not be output, you must set the appropriate masks to your parameter object. For example, assume this System object is called "solv_box", you can do the following prior to dumping the pair coeffs:

solv_box.parameters.coul_mask = True solv_box.parameters.lj_mask = False solv_box.parameters.morse_mask = False

Note - this function dumps pair coeffs for the input script, NOT the data file. If you wish for the alternative, see dump_pair_coeffs_data()

dump_pair_coeffs_data()
Will try to dump all available pair coefficients. Currently, this means that Coulomb, Lennard-Jones, and Morse will be attempted. If you prefer that one or another not be output, you must set the appropriate masks to your parameter object. For example, assume this System object is called "solv_box", you can do the following prior to dumping the pair coeffs:

solv_box.parameters.coul_mask = True solv_box.parameters.lj_mask = False solv_box.parameters.morse_mask = False

Note - this function dumps pair coeffs for the data file, NOT the input script. If you wish for the alternative, see dump_pair_coeffs()

get_atom_masses()
This simplifies using data file writing by getting the masses of all the atoms in the correct order.

Returns

masses: ***list, float*** A list of the masses for each atom type.

get_center_of_geometry(*skip_H=False*)
Calculate the center of geometry of the system.

Parameters

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

cog: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of geometry.

get_center_of_mass(*skip_H=False*)
Calculate the center of mass of the system.

Parameters

skip_H: ***bool, optional*** Whether to include Hydrogens in the calculation (False), or not (True).

Returns

com: ***np.array, float*** A np.array of the x, y, and z coordinate of the center of mass.

get_elements()
This simplifies using dump_modify by getting a list of the elements in this system, sorted by their weight. Note, duplicates will exist if different atom types exist within this system!

Returns

elements: ***list, str*** A list of the elements, sorted appropriately for something like dump_modify.

reassign_indices(*mol_offset=1*, *atom_offset=1*)
Given a system of many molecules and atoms, reassign all the indices to be consistent.

Parameters

mol_offset: ***int, optional*** The molecule atom offset. Do we start at 0 or 1 or 2? By default it is 0.

atom_offset: ***int, optional*** The atom offset. Do we start at 0 or 1 or 2? By default it is 0.

Returns

None

rotate(*m*, *around='com'*)
Rotate the system by the given matrix *m*.

Parameters

m: ***list, list, float*** A 3x3 matrix describing the rotation to be applied to this molecule.

around: ***str, optional*** Whether to rotate around the center of mass (com), center of geometry (cog), or neither ("None" or None).

Returns

None

set_types(*opls_file='/home/hherbol/programs/squid/squid/structures/../forcefields/potentials/oplsaa.prm'*, *smrff_file=None*, *params=None*)
Given the atoms, bonds, angles, and dihedrals in a system object, generate a list of the unique atom, bond, angle, dihedral types and assign that to the system object.

Parameters

params: **squid.forcefields.parameters.Parameters,** ***optional*** A parameter object that already exists.

opls_file: ***str, optional*** A path to an opls file. By default, this points to the stored values in squid. If None, then no OPLS file is read. Otherwise, a custom file is read.

smrff_file: ***str, optional*** A path to a smrff file. By default, none is read.

zhi = None
Otherwise, we have a monoclinic box. We will set the center to the euclidean origin.

3.14.5 topology

class squid.structures.topology.**Connector**(*atoms*, *length=None*, *angle=None*)
The Connector class works to hold connection information between atoms. This is used primarily for bonds, angles, and dihedrals. A corresponding length and angle can also be held in the object (defaults to None).

Parameters

atoms: ***list,*** **squid.structures.atom.Atom** A list of atoms to connect. If you connect atoms for an angle, the second atom is the center atom.

length: ***float, optional*** The bond length in Angstroms.

angle: ***float, optional*** The angle of the connection in degrees.

Returns

connection: **squid.structures.topology.Connector** This connector object.

squid.structures.topology.**get_angle**(*a*, *center=None*, *b=None*, *deg=True*)
Determine the angle between three atoms. In this case, determine the angle a-center-b.

Parameters

a: **squid.structures.atom.Atom** First atom in the angle.

center: **squid.structures.atom.Atom** Center atom of the angle.

b: **squid.structures.atom.Atom** Last atom in the angle.

deg: ***bool, optional*** Whether to return the angle in degrees (True) or radians (False).

Returns

theta: ***float*** Return the angle, default is degrees.

squid.structures.topology.**get_dihedral_angle**(*a*, *b=None*, *c=None*, *d=None*, *deg=True*)
Use the Praxeolitic formula to determine the dihedral angle between 4 atoms.

Parameters

a: **squid.structures.atom.Atom** First atom in the dihedral, or a tuple of all 4.

b: **squid.structures.atom.Atom, *optional*** Second atom in the dihedral.

c: **squid.structures.atom.Atom, *optional*** Third atom in the dihedral.

d: **squid.structures.atom.Atom, *optional*** Fourth atom in the dihedral.

deg: ***bool, optional*** Whether to return the angle in degrees (True) or radians (False).

Returns

theta: ***float*** Return the dihedral angle, default is degrees.

References

- http://stackoverflow.com/a/34245697

3.15 utils

3.15.1 cast

squid.utils.cast.**assert_vec**(*v*, *length=3*, *numeric=True*)
Given what should be a vector of N values, we assert that they are indeed valid.

Parameters

v: ***array-like*** Some array like object.

length: ***int, optional*** The required length of the array.

numeric: ***bool, optional*** Whether to require the array be of numerical values or not.

Returns

None

squid.utils.cast.**check_vec**(*v*, *length=3*, *numeric=True*)
Given what should be a vector of N values, we check that they are indeed valid.

Parameters

v: ***array-like*** Some array like object.

length: ***int, optional*** The required length of the array.

numeric: ***bool, optional*** Whether to require the array be of numerical values or not.

Returns

valid: *bool* Whether the array follows the specifications defined or not.

squid.utils.cast.**is_array**(*v*)
Simply check if v is array like

Parameters

v: *array-like* Some array like object.

Returns

valid: *bool* Whether the object is array like or not.

squid.utils.cast.**is_numeric**(*x*)
A simple function to test if x can be cast to a float.

Parameters

v: *numeric-object* Some variable that should be numeric.

Returns

valid: *bool* Whether the object is numeric or not.

squid.utils.cast.**simplify_numerical_array**(*values*)
Given integer values, simplify to a numerical array. Note, values may also be given as a comma separated string. This is used in jobarray.

Parameters

values: *list, int* or *str* A list of integers, or a comma separated string of integers.

Returns

simple_string: *str* A single string simplifying the order.

3.15.2 print_helper

squid.utils.print_helper.**bytes2human**(*n*)
Convert n bytes (as integer) to a human readable string. Code was found online at activestate (see references).

Parameters

n: *int* The number of bytes.

Returns

n_in_str: *str* The bytes in string format.

References

- http://code.activestate.com/recipes/578019

squid.utils.print_helper.**color_set**(*s*, *c*)
Colourize a string for linux terminal output.

Parameters

s: *str* String to be formatted.

c: *str* Colour or format for the string, found in constants.COLOUR.

Returns

s: *str* Coloured or formatted string.

squid.utils.print_helper.**colour_set**(*s*, *c*)
Colourize a string for linux terminal output.

Parameters

s: ***str*** String to be formatted.

c: ***str*** Colour or format for the string, found in constants.COLOUR.

Returns

s: ***str*** Coloured or formatted string.

squid.utils.print_helper.**printProgressBar**(*iteration*, *total*, *prefix=''*, *suffix=''*, *decimals=1*, *length=20*, *fill='+'*, *buf=None*, *pad=False*)
NOTE! THIS IS COPIED FROM STACK OVERFLOW (with minor changes), USER Greenstick Link: https://stackoverflow.com/a/34325723

Call in a loop to create terminal progress bar.

Parameters

iteration: ***int*** Current iteration.

total: ***int*** Total number of iterations.

prefix: ***str, optional*** Prefix for the loading bar.

suffix: ***str, optional*** Suffix for the loading bar.

decimals: ***int, optional*** Positive number of decimals in percent complete

length: ***int, optional*** Character length of the loading bar.

fill: ***str, optional*** Bar fill character.

pad: ***bool, optional*** Whether to pad the right side with spaces until terminal width.

squid.utils.print_helper.**spaced_print**(*sOut, delim=['\t', ' '], buf=4*)
Given a list of strings, or a string with new lines, this will reformat the string with spaces to split columns. Note, this only works if there are no headers to the input string/list of strings.

Parameters

sOut: ***str*** **or** ***list, str*** String/list of strings to be formatted.

delim: ***list, str*** List of delimiters in the input strings.

buf: ***int*** The number of spaces to have between columns.

Returns

spaced_s: ***str*** Appropriately spaced output string.

squid.utils.print_helper.**strip_color**(*s*)
Remove colour and/or string formatting due to linux escape sequences.

Parameters

s: ***str*** String to strip formatting from.

Returns

s: ***str*** Unformatted string.

squid.utils.print_helper.**strip_colour**(*s*)
Remove colour and/or string formatting due to linux escape sequences.

Parameters

s: ***str*** String to strip formatting from.

Returns

s: ***str*** Unformatted string.

3.15.3 units

squid.utils.units.**convert**(*old*, *new*, *val*)
A generic converter of fractional units. This works only for one unit in the numerator and denomenator (such as Ha/Ang to eV/Bohr).

Parameters

old: ***str*** Units for which val is in.

new: ***str*** Units to convert to.

val: ***float*** Value to convert.

Returns

new_val: ***float*** Converted value in units of new.

squid.utils.units.**convert_dist**(*d0*, *d1*, *d_val*)
Convert distance units.

Parameters

d0: ***str*** Unit of distance that d_val is in.

d1: ***str*** Unit of distance that you wish to convert to.

d_val: ***float*** Value to be converted.

Returns

distance: ***float*** Converted d_val to units of d1.

squid.utils.units.**convert_energy**(*e0*, *e1*, *e_val*)
Convert energy units.

Parameters

e0: ***str*** Unit of energy that e_val is in.

e1: ***str*** Unit of energy that you wish to convert to.

e_val: ***float*** Value to be converted.

Returns

energy: ***float*** Converted e_val to units of e1.

squid.utils.units.**convert_pressure**(*p0*, *p1*, *p_val*)
Convert pressure units.

Parameters

p0: ***str*** Unit of pressure that p_val is in.

p1: ***str*** Unit of pressure that you wish to convert to.

p_val: ***float*** Value to be converted.

Returns

pressure: ***float*** Converted p_val to units of p1.

squid.utils.units.**elem_i2s**(*elem_int*)
Get the elemental symbol, given its atomic number.

Parameters

elem_int: ***int*** Atomic number of an element.

Returns

elem_sym: ***str*** Elemental symbol.

squid.utils.units.**elem_s2i**(*elem_sym*)
Get the atomic number, given its elemental symbol.

Parameters

elem_sym: ***str*** Elemental symbol of an element.

Returns

elem_int: ***int*** Atomic number.

squid.utils.units.**elem_sym_from_weight**(*weight*, *delta=0.1*)
Get the element that best matches the given weight (in AMU).

Parameters

weight: ***float*** Weight of an element in AMU.

delta: ***float, optional*** How close you permit the matching to be in AMU.

Returns

elem_sym: ***str*** The elemental symbol.

squid.utils.units.**elem_weight**(*elem*)
Get the weight of an element, given its symbol or atomic number.

Parameters

elem: ***str or int*** Elemental symbol or atomic number.

Returns

elem_weight: ***float*** Weight of the element in AMU.

CHAPTER

FOUR

CONSOLE SCRIPTS

Various console scripts exist to aid users in simple tasks.

4.1 chkDFT

This command allows one to post-process a dft (orca or g09) output file and summarize the findings to the terminal.

```
chkDFT
---------
A command to quickly get a glimpse of a DFT simulation.
chkDFT [Sim_Name] [Options]

    Flag        Default     Description
-help, -h     :         :  Print this help menu
-dft          :  orca   :  Specify what type of dft simulation you want to
                           parse. By default it is 'g09', but can be
                           'orca' or 'jdftx'.
-units, -u    :  Ha     :  Specify the units you want the output to be in.
                           By default this is Hartree.
-scale        :  1.0    :  Scale all energies by this value
-out, -o      :  out    :  Make an output file with this name holding all
                           xyz coordinates. If no xyz data is available
                           this will not run. Default output name is
                           'out.xyz' but user can choose their own using
                           this command.
-vmd, -v      :         :  Opens output xyz file in vmd. Flag turns on.
-ovito, -ov   :         :  Opens output xyz file in ovito. Flag turns on.
-me           :         :  Forces the .xyz file to be saved to ~/out.xyz

ex. chkDFT water -dft orca -u kT_300
```

4.2 scanDFT

This command allows one to compile together a Nudged Elastic Band reaction pathway from various output DFT calculations. It will both graph the energy pathway, and generate a final xyz file of the last frames.

```
scanDFT
---------
A command to view the energy landscape over several configurations.
There are two ways to implement this:
```

(continues on next page)

(continued from previous page)

```
1.
  Note, the START STOP range is inclusive on either end.

  scanDFT [Sim_Name%%d] START STOP [Options]

2.
  scanDFT Sim_Name

The first method is useful when utilizing flags.  The second method will
prompt the user for information.  Note, in the second instance it will assume
an appendage of -%%d-%%d, describing the iteration and frame.  It also assumes
and NEB scan is desired.

    Flag          Default     Description
-help, -h     :            :  Print this help menu
-dft          :    orca    :  Specify what type of dft simulation you want to
                               get the energy landscape of. Other options
                               include 'orca'.
-units, -u    :   kT_300   :  Specify the units you want the output to be in.
-scale        :    1.0     :  Scale all energies by this value. Applied AFTER
                               unit conversion from simulation units ('Ha') to
                               -units.
-out, -o      :    out     :  Make an output file with this name holding all
                               xyz coordinates of what you're scanning over.
-step         :    1.0     :  Steps to take between your start and stop range
-c            :            :  Compile multiple energy landscapes on the same
                               graph. Takes three arguments, separated by commas
                               with no spaces:
                                    char,start,stop
                               The character is a unique identifier in the
                               Sim_Name that will be replaced with values from
                               start to stop (inclusive)
-neb          :            :  In NEB calculations, each iteration after the
                               first does not include the first and last
                               energies. Giving this flag and a run name for
                               the first in the NEB list will tack on these
                               energies to the rest of the simulations.

-title, -t    :            :  Title for the output graph
-lx           :            :  Label for the x-axis
-ly           :            :  Label for the y-axis
-xrange       :            :  Set the x-axis range
-yrange       :            :  Set the y-axis range
-xvals        :            :  Set a custom label for x-axis (comma separated).
-print, -p    :            :  Print out the values that are plotted.
-save, -s     :            :  Whether to save the graph to out.png (True) or
                               not (False). Note, when saving it will not
                               display the graph.

ex: scanDFT water
ex: scanDFT water_ 1 10
ex: scanDFT water_%d 1 10
ex: scanDFT water%d_opt 1 10
ex: scanDFT water_^_%d 1 10 -c ^,0,4 -dft orca
ex: scanDFT water_^_%d 1 10 -c ^,2,4 -dft orca -neb water_0_0,water_0_10
ex: scanDFT water_opt_%d 1 10 -t "Water Optimization" -xrange 0,5
```

4.3 procrustes

This command allows one to quickly, from the command line, clean-up an xyz file of several frames. It will remove the rigid rotations between consecutive frames, and also allows for linear interpolation.

```
procrustes
---------
A command line tool to run procrustes along an xyz file.

procrustes [file.xyz] [Options]

    Flag            Default         Description
-help, -h        :            :  Print this help menu
-overwrite, -o   :            :  Overwrite the initial file
-append, -a      :   _proc     :  Change the appended name alteration
-interpolate, -i :            :  This will turn on linear interpolation
-rmax            :    0.5      :  The default max rms for interpolation
-fmax            :     25      :  The default max number of frames for
                                   interpolation
-nframes, -n     :            :  If specified, interpolate to exactly n
                                   frames.
-between, -b     :            :  If specified, then interpolation is only
                                   run between the two frames.  Note, this
                                   is [x, y) inclusive.

Default behaviour is to use procrustes on an xyz to best align
the coordinates, and then to save a new xyz file with the name
OLD_proc.xyz (where OLD is the original xyz file name).

NOTE! If you specify -o and -a, then appending will occur instead
of overwritting.

Ex.

procrustes demo.xyz
procrustes demo.xyz -i -n 20
procrustes demo.xyz -i -rmax 0.1 -fmax 30
procrustes demo.xyz -i -b 5 8 -n 6
```

4.4 pysub

This command allows one to quickly submit a python script to run either in the background locally, or on a queue/partition within a cluster.

```
pysub
---------
A command line tool to submit jobs to the queue.

pysub [script.py] [Options]

    Flag          Default     Description
-help, -h      :            :  Print this help menu
-n             :     1      :  Number of processors to use
-nt, -tasks    :     1      :  Number of tasks this job will run
```

(continues on next page)

(continued from previous page)

```
-o, -omp       :            :  Manually specify what OMP_NUM_THREADS should be.
-mpi           :            :  Whether to run python with mpirun or not.
-q             :            :  Which queue to submit to
-walltime, -t  :  00:30:00  :  The walltime to use
-priority, -p  :            :  Manually specify job priority
-unique, -u    :   False    :  Whether to require a unique simulation name.

-jobarray, -ja :   None     :  Whether to run a job array.  If this flag is
                               specified, it MUST be followed by two values to
                               indicate the lower and upper bounds of the
                               indexing.

-xhost, -x     :            :  If needed, specify computer
-args, -a      :            :  A list of arguments for the python code
-mods, -m      :            :  Specify the modules you wish to use here.
-mo            :   False    :  Whether to override the default modules.
-keep, -k      :            :  Whether to keep the submission file

-py3           :            :  Whether to use python 3, or 2 (2 is default).
-alloc, -A     :   None     :  Whether to specify a SLURM Allocation.
-gpu           :   None     :  The number of desired GPUs you want.

Default behaviour is to generate a job with the same name
as the python script and to generate a .log file with the
same name as well.

When using -mpi, it will only be effective if nprocs > 1.

NOTE! If using xhost or args, make sure it is the last flag
as we assume all remaining inputs are the desired strings. This
means that only xhost or args can be used at a time (both would
lead to errors).
```

CHAPTER

FIVE

EXAMPLES

On the squid github repo, we include an examples folder that describes simple use cases. These are replicated here:

- Using NEB to find the MEP of CNH Isomerization
- Calculating and visualizing the molecular orbitals of Water
- Calculating and visualizing the electrostatic potential surface of Water
- Using procrustes and linear interpolation to smooth predicted reaction pathways
- Equilibrating a box of benzene and acetone in Molecular Dynamics

5.1 Nudged Elastic Band Demo

The below code shows how one can generate a reaction pathway, and ultimately run NEB on it to find the minimum energy pathway (MEP). Further, it automates the submission of an eigenvector following Transition State optimization from the peak, and verifies a transition state was found. Note, the endpoints and NEB should use the same DFT level of theory, otherwise your endpoints may not remain local minima within the potential energy surface.

```
from squid import orca
from squid import files
from squid import geometry
from squid.calcs import NEB
from squid import structures


if __name__ == "__main__":
    # In this example we will generate the full CNH-HCN isomerization using
    # only squid.  Then we optimize the endpoints in DFT, smooth the frames,
    # and subsequently run NEB

    # Step 1 - Generate the bad initial guess
    print("Step 1 - Generate the bad initial guess...")
    H_coords = [(2, 0), (2, 0.5), (1, 1), (0, 1), (-1, 0.5), (-1, 0)]
    CNH_frames = [[
        structures.Atom("C", 0, 0, 0),
        structures.Atom("N", 1, 0, 0),
        structures.Atom("H", x, y, 0)]
        for x, y in H_coords
    ]
    # Save initial frames
    files.write_xyz(CNH_frames, "bad_guess.xyz")
```

(continues on next page)

(continued from previous page)

```
    # Step 2 - Optimize the endpoints
    print("Step 2 - Optimize endpoints...")
    frame_start_job = orca.job(
        "frame_start", "! HF-3c Opt", atoms=CNH_frames[0], queue=None
    )
    frame_last_job = orca.job(
        "frame_last", "! HF-3c Opt", atoms=CNH_frames[-1], queue=None
    )
    # Wait
    frame_start_job.wait()
    frame_last_job.wait()

    # Step 3 - Read in the final coordiantes, and update the band
    print("Step 3 - Store better endpoints...")
    CNH_frames[0] = orca.read("frame_start").atoms
    CNH_frames[-1] = orca.read("frame_last").atoms
    # Save better endpoints
    files.write_xyz(CNH_frames, "better_guess.xyz")

    # Step 4 - Smooth out the band to 10 frames
    print("Step 4 - Smooth out the band...")
    CNH_frames = geometry.smooth_xyz(
        CNH_frames, N_frames=8,
        use_procrustes=True
    )
    # Save smoothed band
    files.write_xyz(CNH_frames, "smoothed_guess.xyz")

    # Step 5 - Run NEB
    print("Step 5 - Run NEB...")
    neb_handle = NEB(
        "CNH", CNH_frames, "! HF-3c",
        nprocs=1, queue=None, ci_neb=True)
    CNH_frames = neb_handle.optimize()[-1]
    # Save final band
    files.write_xyz(CNH_frames, "final.xyz")

    # Step 6 - Isolate the peak frame, and converge to the transition state
    print("Step 6 - Calculating Transition State...")
    ts_job = orca.job(
        "CNH_TS", "! HF-3c OptTS NumFreq",
        extra_section='''
%geom
    Calc_Hess true
    NumHess true
    Recalc_Hess 5
    end
''',
        atoms=CNH_frames[neb_handle.highest_energy_frame_index], queue=None
    )
    ts_job.wait()

    # Ensure we did find the transition state
    data = orca.read("CNH_TS")
    vib_freq = data.vibfreq
    if sum([int(v < 0) for v in vib_freq]) == 1:
        print("    Isolated a transition state with exactly 1 negative vibfreq.")
```

(continues on next page)

(continued from previous page)

```
        print("    Saving it to CNH_ts.xyz")
        files.write_xyz(data.atoms, "CNH_ts.xyz")
    else:
        print("FAILED!")
```

Example output is as follows:

```
-------------------------------------------------------------------------------------
----
Run_Name = CNH
DFT Package = orca
Spring Constant for NEB: 0.00367453 Ha/Ang = 0.1 eV/Ang
Running Climbing Image, starting at iteration 5

Running neb with optimization method LBFGS
    step_size = 1
    step_size_adjustment = 0.5
    max_step = 0.04
    Using numerical optimization starting hessian approximation.
    Will reset stored parameters and gradients when stepped bad.
    Will reset step_size after 20 good steps.
    Will accelerate step_size after 20 good steps.
    Will use procrustes to remove rigid rotations and translations
Convergence Criteria:
    g_rms = 0.001 (Ha/Ang) = 0.0272144 (eV/Ang)
    g_max = 0.001 (Ha/Ang) = 0.0272144 (eV/Ang)
    maxiter = 1000
---------------------------------------------
Step    RMS_F (eV/Ang)  MAX_F (eV/Ang)  MAX_E (kT_300)  Energies (kT_300)
----
0   28.7949     44.5242     223.9       -92.232 + 109.6 215.5 223.9 182.8  62.4  62.4␣
↪-24.8
1   15.339      21.7786     161.1       -92.232 +  44.3 143.0 161.1  88.4  13.7  20.3␣
↪-24.8
2   14.6517     20.8575     158.0       -92.232 +  41.7 139.4 158.0  86.2  11.9  17.2␣
↪-24.8
3   6.0258      9.376       122.9       -92.232 +  15.2 100.8 122.9  62.8  -5.4  -9.2␣
↪-24.8
4   5.6856      8.8098      121.5       -92.232 +  14.5  99.4 121.5  61.5  -5.7  -9.3␣
↪-24.8
5   1.8606      3.0362      107.7       -92.232 +   9.4  86.9 107.7  50.6  -8.2 -10.1␣
↪-24.8
6   1.1459      3.024       105.3       -92.232 +   9.3  85.5 105.3  49.1  -8.4 -11.0␣
↪-24.8
7   0.945       2.5354      105.2       -92.232 +   9.4  84.9 105.2  48.5  -8.5 -11.3␣
↪-24.8
8   0.9274      2.0697      107.9       -92.232 +   9.5  84.5 107.9  48.8  -8.5 -11.9␣
↪-24.8
9   0.8502      1.9055      110.2       -92.232 +   9.4  84.5 110.2  49.1  -8.6 -12.3␣
↪-24.8
10  0.9637      1.7608      114.3       -92.232 +   9.5  85.0 114.3  49.5  -8.4 -13.0␣
↪-24.8
11  0.8564      1.4446      115.4       -92.232 +   9.5  84.9 115.4  49.4  -8.4 -13.4␣
↪-24.8
12  0.8252      1.2095      116.8       -92.232 +   9.7  84.6 116.8  49.6  -8.2 -14.5␣
↪-24.8
13  0.3625      0.742       115.6       -92.232 +   9.6  84.2 115.6  49.3  -8.4 -14.3␣
↪-24.8
```

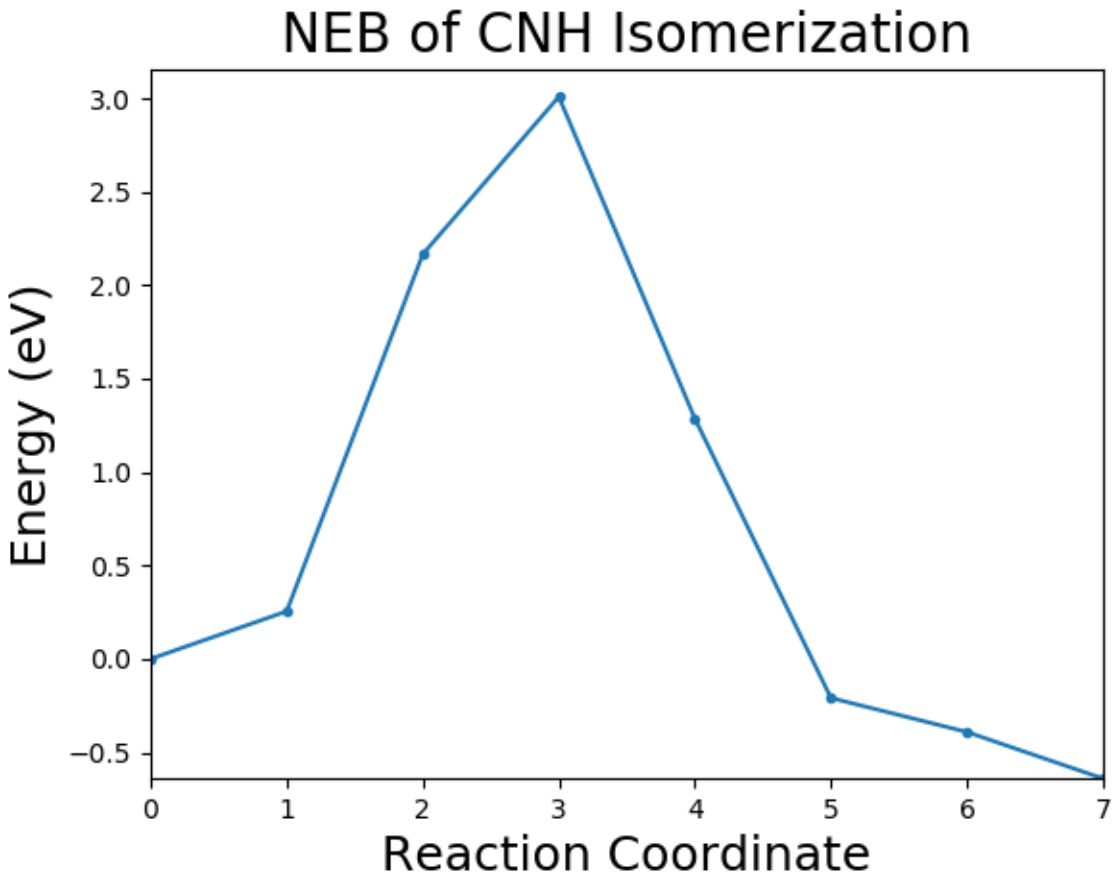
(continues on next page)

(continued from previous page)

```
14  0.8296      1.4249      116.8       -92.232 +   9.9  84.3 116.8  49.9  -8.1 -15.6␣
↪-24.8
15  0.6218      1.0318      116.8       -92.232 +   9.8  84.2 116.8  49.8  -8.2 -15.6␣
↪-24.8
16  0.2493      0.5738      116.4       -92.232 +   9.7  83.9 116.4  49.6  -8.2 -15.2␣
↪-24.8
17  0.1849      0.3175      116.4       -92.232 +   9.7  83.9 116.4  49.6  -8.2 -15.1␣
↪-24.8
18  0.1046      0.2349      116.3       -92.232 +   9.8  83.8 116.3  49.7  -8.2 -15.1␣
↪-24.8
19  0.0459      0.1069      116.3       -92.232 +   9.8  83.8 116.3  49.7  -8.1 -15.1␣
↪-24.8
20  0.0221      0.0458      116.3       -92.232 +   9.8  83.7 116.3  49.7  -8.1 -15.1␣
↪-24.8

NEB converged the RMS force.
------------------------------------------------------------------------------------
↪----
```

With the following graph made using the scanDFT command line tool:

```
scanDFT CNH-20-%d 1 6 -neb CNH-0-0,CNH-0-7 -t "NEB of CNH Isomerization" -lx
↪"Reaction Coordinate" -ly "Energy" -u eV
```



5.2 Molecular Orbital Visualization Demo

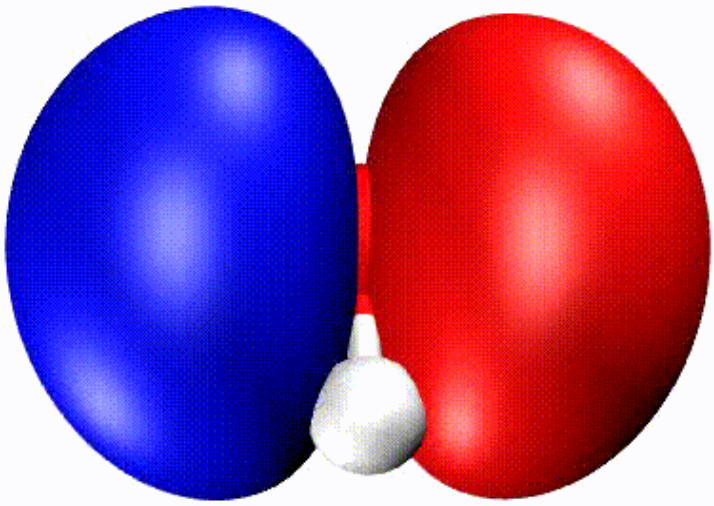
The below code shows how one can visualize molecular orbitals of a molecule (in this case water) using VMD.

```
from squid import orca
from squid import files


if __name__ == "__main__":
    # First, calculate relevant information
    frames = files.read_xyz('water.xyz')
    job_handle = orca.job(
        'water',
        '! PW6B95 def2-TZVP D3BJ OPT NumFreq',
        atoms=frames,
        queue=None)
    job_handle.wait()

    # Next, post process it
    orca.mo_analysis(
        "water", orbital=None,
        HOMO=True, LUMO=True,
        wireframe=False, hide=True, iso=0.04
    )
```

In the console output it'll show the following in blue:

```
Representations are as follows:

    1 - CPK of atoms
    2 - LUMO Positive
    3 - HOMO Positive
    4 - LUMO Negative
    5 - HOMO Negative
    6 - Potential Surface
    7 - MO 3
```

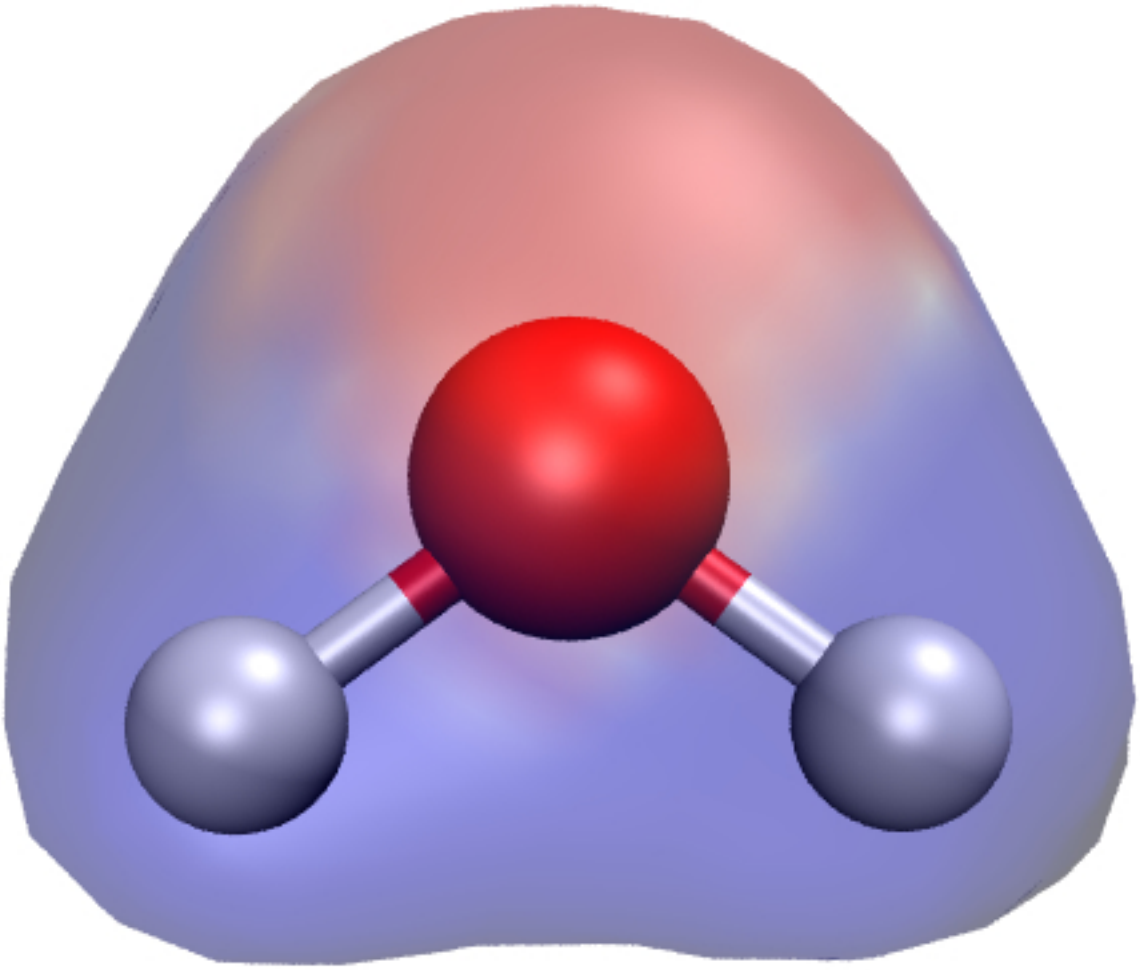
Choosing only displays 1, 3, and 5 we can see the HOMO level of water as follows (positive being blue and negative being red):



5.3 DFT - Electrostatic Potential Mapped on Electron Density Post Processing

We can also readily generate an electrostatic potential mapped onto an electron density isosurface using squid. One thing to note is that the final results are subjective depending on two primary values, which the user may set in VMD. These values are found under the *Graphics* > *Representations* tab as the *Isovalue* listed in *Draw Style* and the *Color Scale Data Range* listed under *Trajectory*.

```
from squid import orca
from squid import files


if __name__ == "__main__":
    # First, calculate relevant information
    frames = files.read_xyz('water.xyz')
    job_handle = orca.job(
        'water',
        '! PW6B95 def2-TZVP D3BJ OPT NumFreq',
        atoms=frames,
        queue=None)
    job_handle.wait()

    # Next, post process it
    orca.pot_analysis(
        "water", wireframe=False, npoints=80
    )
```



5.4 Geometry - Smoothing out a Reaction Coordinate

The below code shows how we can smooth out xyz coodinates in a reaction pathway using linear interpolation and procrustes superimposition.

import os import shutil from squid import files from squid import geometry from squid import structures from squid.post_process.ovito import ovito_xyz_to_gif

def example_1(): # In this example, we will generate a smooth CNH-HCN isomerization # guessed pathway

Step 1 - Generate the bad initial guess print("Step 1 - Generate the bad initial guess...") H_coords = [(2, 0), (2, 1), (1, 1), (0, 1), (-1, 1), (-1, 0)] CNH_frames = [[

structures.Atom("C", 0, 0, 0), structures.Atom("N", 1, 0, 0), structures.Atom("H", x, y, 0)] for x, y in H_coords

] # Further, randomly rotate the atoms CNH_frames = [

geometry.perturbate(frame, dx=0.0, dr=360) for frame in CNH_frames

] files.write_xyz(CNH_frames, “rotated_pathway.xyz”)

Step 2 - Use procrustes to remove rotations print(“Step 2 - Use Procrustes to remove rotations…”) geometry.procrustes(CNH_frames) files.write_xyz(CNH_frames, “procrustes_pathway.xyz”)

Step 3 - Smooth out the band by minimizing the RMS atomic motion between # consecutive frames until it is below 0.1 (with a max of 50 frames). print(“Step 3 - Smooth out the band…”) CNH_frames = geometry.smooth_xyz(

CNH_frames, R_max=0.1, F_max=50, use_procrustes=True

) # Save smoothed band files.write_xyz(CNH_frames, “smoothed_pathway.xyz”)

if __name__ == “__main__”: example_1()

Further, we can automate the generation of gifs using the ovitos python interface. Note, this is not always guaranteed to be a pretty image, as you would need to know exactly where to point the camera. In some situations it may be obvious where it should be placed; however, in many we simply recommend opening up Ovito and using their GUI interface directly.

import os import shutil from squid import files from squid import geometry from squid import structures from squid.post_process.ovito import ovito_xyz_to_gif

def example_2(): # In this example, we illustrate how we can automate the generation of # gifs of the reactions in example_1

print(“Step 4 - Generating gifs…”)

Generate a scratch folder for image generation scratch_folder = “./tmp” if os.path.exists(scratch_folder):

shutil.rmtree(scratch_folder)

print(” rotated_pathway.gif”) os.mkdir(scratch_folder) ovito_xyz_to_gif(

files.read_xyz(“rotated_pathway.xyz”), scratch_folder, fname=”rotated_pathway”, camera_pos=(0, 0, -10), camera_dir=(0, 0, 1))

shutil.rmtree(scratch_folder)

print(” procrustes_pathway.gif”) os.mkdir(scratch_folder) ovito_xyz_to_gif(

files.read_xyz(“procrustes_pathway.xyz”), scratch_folder, fname=”procrustes_pathway”, camera_pos=(0, 0, -10), camera_dir=(0, 0, 1))

shutil.rmtree(scratch_folder)

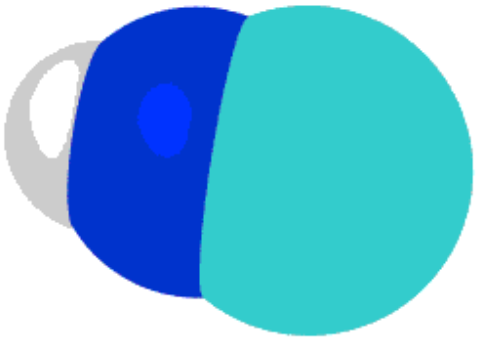
print(” smoothed_pathway.gif”) os.mkdir(scratch_folder) ovito_xyz_to_gif(

files.read_xyz(“smoothed_pathway.xyz”), scratch_folder, fname=”smoothed_pathway”, camera_pos=(0, 0, -10), camera_dir=(0, 0, 1))

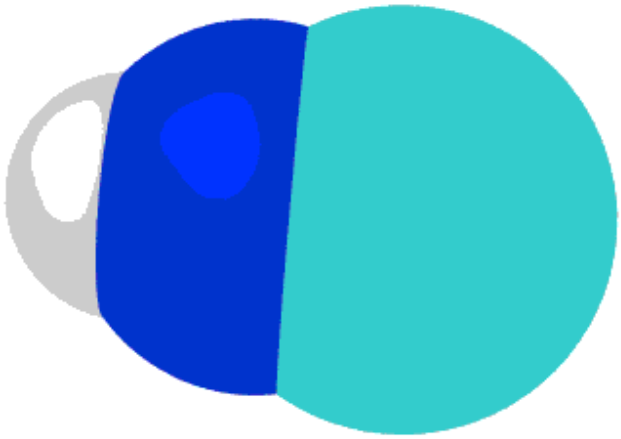
shutil.rmtree(scratch_folder)

if __name__ == “__main__”: example_2()

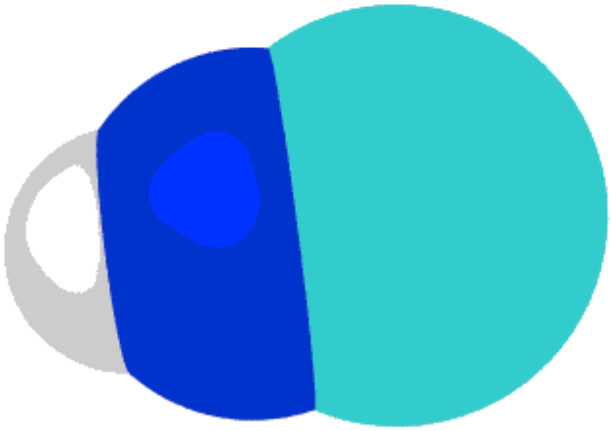
The rough reaction coordinate with rotations, shown below, would not lend itself to linear interpolation, as neighbouring frames would lead to atoms overlapping.



However, when using the procrustes method we remove the rigid rotation associated with this change of coordinate system, making it appear much better.



Finally, with the added linear interpolations we end up with a smooth reaction coordinate.



5.5 Molecular Dynamics Sovlent Box Equilibration

In this example, we equilibrate an MD box of two benzene molecules (offset by 10, 10, 10) and acetone (packed to a density of 1.0 using packmol).

```
from squid import files
from squid import lammps
from squid import geometry
from squid import structures

if __name__ == "__main__":
    # In this example, we discuss how to handle running MD using LAMMPS and
    # squid.  We start by generating a System object, add in some molecules,
    # and then pack this system object with solvents.  We then equilibrate
    # the system.

    # Step 1 - Generate the system
    world = structures.System(
        "solv_box", box_size=(15.0, 15.0, 15.0), periodic=True)

    # Step 2 - Get any molecules you want
    mol1 = files.read_cml("benzene.cml")[0]
    mol2 = mol1 + (10, 10, 10)
```

(continues on next page)

(continued from previous page)

```
    solv = files.read_cml("acetone.cml")[0]

    # Step 3 - Add them however you want
    world.add(mol1)
    world.add(mol2)
    geometry.packmol(world, [solv], persist=False, density=1.0)

    # Step 4 - Run a simulation
    world.set_types()

    input_script = """units real
atom_style full
pair_style lj/cut/coul/cut 10.0
bond_style harmonic
angle_style harmonic
dihedral_style opls

boundary p p p
read_data solv_box.data

pair_modify mix geometric

""" + world.dump_pair_coeffs() + """

dump 1 all xyz 100 solv_box.xyz
dump_modify 1 element """ + ' '.join(world.get_elements()) + """

compute pe all pe/atom
dump forces all custom 100 forces.dump id element x y z fx fy fz c_pe
dump_modify forces element """ + ' '.join(world.get_elements()) + """

thermo_style custom ke pe temp press
thermo 100

minimize 1.0e-4 1.0e-6 1000 10000

velocity all create 300.0 23123 rot yes dist gaussian
timestep 1.0

fix motion_npt all npt temp 300.0 300.0 100.0 iso 0.0 0.0 1000.0
run 10000
unfix motion_npt

fix motion_nvt all nvt temp 300.0 300.0 300.0
run 10000
unfix motion_nvt
"""
    job_handle = lammps.job("solv_box", input_script, system=world, nprocs=1)
    job_handle.wait()
```

In this example, we want to write a lammps data file without knowing any parameters, so we strip away all relevant information and write the file.

```
from squid import files
from squid import lammps
from squid import geometry
```

(continues on next page)

(continued from previous page)

```
from squid import structures

if __name__ == "__main__":
    # Step 1 - Generate the system
    world = structures.System(
        "solv_box", box_size=(15.0, 15.0, 15.0), periodic=True)

    # Step 2 - Get any molecules you want
    mol1 = files.read_cml("benzene.cml")[0]
    mol2 = mol1 + (10, 10, 10)
    solv = files.read_cml("acetone.cml")[0]

    # Step 3 - In the case that we do not know the atom types, but we still
    # want to generate a lammps data file, we can still do so!  We must first
    # in this example strip away all relevant bonding information.  Further,
    # and this is important: YOU MUST SET a.label and a.charge to the element
    # and some value (in this example I set it to 0.0).
    for mol in [mol1, mol2, solv]:
        for a in mol.atoms:
            a.label = a.element
            a.charge = 0.0
        mol.bonds = []
        mol.angles = []
        mol.dihedrals = []

    # Step 4 - Add them however you want
    world.add(mol1)
    world.add(mol2)
    geometry.packmol(world, [solv], persist=False, density=1.0)

    # Step 5 - Run a simulation
    world.set_types()
    lammps.write_lammps_data(world)
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

S

squid.files.misc, 24
squid.forcefields.smrff, 44
squid.geometry.misc, 50
squid.lammps.io.thermo, 63
squid.lammps.parser, 64
squid.maths.lhs, 66
squid.optimizers.bfgs, 66
squid.optimizers.conjugate_gradient, 68
squid.optimizers.fire, 69
squid.optimizers.lbfgs, 69
squid.optimizers.quick_min, 71
squid.optimizers.steepest_descent, 71
squid.post_process.vmd, 79
squid.utils.cast, 89
squid.utils.print_helper, 90
squid.utils.units, 92

INDEX

A

add() (*squid.structures.system.System method*), 86
align_centroid() (*in module squid.geometry.transform*), 54
align_coordinates() (*squid.calcs.aneb.ANEB method*), 17
align_coordinates() (*squid.calcs.neb.NEB method*), 21
ANEB (*class in squid.calcs.aneb*), 16
Angle (*class in squid.forcefields.connectors*), 25
assert_vec() (*in module squid.utils.cast*), 89
assign_angles_and_dihedrals() (*squid.structures.molecule.Molecule method*), 82
assign_line() (*squid.forcefields.connectors.HarmonicConnector method*), 27
assign_line() (*squid.forcefields.coulomb.Coul method*), 29
assign_line() (*squid.forcefields.lj.LJ method*), 33
assign_line() (*squid.forcefields.morse.Morse method*), 36
assign_line() (*squid.forcefields.tersoff.Tersoff method*), 46
Atom (*class in squid.structures.atom*), 80

B

bfgs() (*in module squid.optimizers.bfgs*), 66
Bond (*class in squid.forcefields.connectors*), 26
bytes2human() (*in module squid.utils.print_helper*), 90

C

check_restriction() (*in module squid.forcefields.helper*), 31
check_vec() (*in module squid.utils.cast*), 89
close_pipes() (*in module squid.files.misc*), 24
color_set() (*in module squid.utils.print_helper*), 90
colour_set() (*in module squid.utils.print_helper*), 90
conjugate_gradient() (*in module squid.optimizers.conjugate_gradient*), 68
Connector (*class in squid.structures.topology*), 88
contains_molecule() (*squid.structures.system.System method*), 86
convert() (*in module squid.utils.units*), 92
convert_dist() (*in module squid.utils.units*), 92
convert_energy() (*in module squid.utils.units*), 92
convert_pressure() (*in module squid.utils.units*), 92
Coul (*class in squid.forcefields.coulomb*), 29
create_lhs() (*in module squid.maths.lhs*), 66

D

DFT_out (*class in squid.structures.results*), 84
Dihedral (*class in squid.forcefields.connectors*), 26
dump_angles() (*squid.forcefields.parameters.Parameters method*), 39
dump_angles_data() (*squid.structures.system.System method*), 86
dump_atoms_data() (*squid.structures.system.System method*), 86
dump_bonds() (*squid.forcefields.parameters.Parameters method*), 39
dump_bonds_data() (*squid.structures.system.System method*), 86
dump_dihedrals() (*squid.forcefields.parameters.Parameters method*), 40
dump_dihedrals_data() (*squid.structures.system.System method*), 86
dump_line() (*squid.forcefields.tersoff.Tersoff method*), 46
dump_lj_cut_coul_cut() (*squid.forcefields.parameters.Parameters method*), 40
dump_lj_cut_coul_long() (*squid.forcefields.parameters.Parameters method*), 40
dump_morse() (*squid.forcefields.parameters.Parameters method*), 40

dump_pair_coeffs() (*squid.structures.system.System method*), 86
dump_pair_coeffs_data() (*squid.structures.system.System method*), 87
dump_set_charge() (*squid.forcefields.parameters.Parameters method*), 40
dump_smooths() (*squid.forcefields.parameters.Parameters method*), 41
dump_style() (*squid.forcefields.parameters.Parameters method*), 41
dump_tersoff() (*squid.forcefields.parameters.Parameters method*), 41

E

electrostatic_potential_cubegen() (*in module squid.orca.mep*), 76
elem_i2s() (*in module squid.utils.units*), 93
elem_s2i() (*in module squid.utils.units*), 93
elem_sym_from_weight() (*in module squid.utils.units*), 93
elem_weight() (*in module squid.utils.units*), 93
engrad_read() (*in module squid.orca.io*), 72

F

fire() (*in module squid.optimizers.fire*), 69
fix() (*squid.forcefields.connectors.HarmonicConnector method*), 27
fix() (*squid.forcefields.coulomb.Coul method*), 29
fix() (*squid.forcefields.lj.LJ method*), 33
fix() (*squid.forcefields.morse.Morse method*), 36
fix() (*squid.forcefields.parameters.Parameters method*), 41
fix() (*squid.forcefields.tersoff.Tersoff method*), 46
flatten() (*squid.structures.atom.Atom method*), 81
flatten() (*squid.structures.molecule.Molecule method*), 82

G

g09_results() (*in module squid.calcs.aneb*), 18
g09_results() (*in module squid.calcs.neb*), 21
g09_start_job() (*in module squid.calcs.aneb*), 18
g09_start_job() (*in module squid.calcs.neb*), 22
gbw_to_cube() (*in module squid.orca.post_process*), 76
generate() (*squid.forcefields.coulomb.Coul class method*), 30
generate() (*squid.forcefields.lj.LJ class method*), 33
generate() (*squid.forcefields.morse.Morse class method*), 36
generate() (*squid.forcefields.parameters.Parameters method*), 42
generate() (*squid.forcefields.tersoff.Tersoff class method*), 46
get() (*squid.lammps.parser.LMP_Parser method*), 65
get_all_jobs() (*in module squid.jobs.queue_manager*), 58
get_all_jobs() (*squid.jobs.container.JobObject method*), 56
get_all_jobs() (*squid.jobs.nbs.Job method*), 57
get_all_jobs() (*squid.jobs.slurm.Job method*), 60
get_angle() (*in module squid.structures.topology*), 88
get_atom_masses() (*squid.structures.system.System method*), 87
get_available_queues() (*in module squid.jobs.queue_manager*), 59
get_center_of_geometry() (*in module squid.geometry.misc*), 50
get_center_of_geometry() (*squid.structures.molecule.Molecule method*), 82
get_center_of_geometry() (*squid.structures.system.System method*), 87
get_center_of_mass() (*in module squid.geometry.misc*), 50
get_center_of_mass() (*squid.structures.molecule.Molecule method*), 82
get_center_of_mass() (*squid.structures.system.System method*), 87
get_dihedral_angle() (*in module squid.structures.topology*), 89
get_elements() (*squid.structures.system.System method*), 87
get_id_tag() (*squid.structures.atom.Atom method*), 81
get_job() (*in module squid.jobs.nbs*), 57
get_job() (*in module squid.jobs.slurm*), 60
get_lmp_obj() (*in module squid.lammps.job*), 63
get_nbs_queues() (*in module squid.jobs.nbs*), 57
get_orca_obj() (*in module squid.orca.utils*), 77
get_ovito_obj() (*in module squid.post_process.ovito*), 78
get_packmol_obj() (*in module squid.geometry.packmol*), 51
get_pdf() (*in module squid.post_process.debyer*), 77
get_pending_jobs() (*in module squid.jobs.queue_manager*), 59
get_queue_manager() (*in module squid.jobs.queue_manager*), 59
get_running_jobs() (*in module squid.jobs.queue_manager*), 59

get_slurm_queues() (*in module squid.jobs.slurm*), 61
get_smrff_style() (*squid.forcefields.parameters.Parameters method*), 42

H

HarmonicConnector (*class in squid.forcefields.connectors*), 26

I

interpolate() (*in module squid.geometry.transform*), 54
is_array() (*in module squid.utils.cast*), 90
is_exe() (*in module squid.files.misc*), 24
is_finished() (*squid.jobs.container.JobObject method*), 56
is_numeric() (*in module squid.utils.cast*), 90

J

Job (*class in squid.jobs.nbs*), 56
Job (*class in squid.jobs.slurm*), 60
Job() (*in module squid.jobs.queue_manager*), 58
job() (*in module squid.lammps.job*), 64
job() (*in module squid.orca.job*), 73
jobarray() (*in module squid.orca.job*), 74
JobObject (*class in squid.jobs.container*), 56

L

last_modified() (*in module squid.files.misc*), 24
lbfgs() (*in module squid.optimizers.lbfgs*), 69
LJ (*class in squid.forcefields.lj*), 32
LMP_Parser (*class in squid.lammps.parser*), 64
load_opls() (*squid.forcefields.connectors.Angle class method*), 25
load_opls() (*squid.forcefields.connectors.Bond class method*), 26
load_opls() (*squid.forcefields.connectors.Dihedral class method*), 26
load_opls() (*squid.forcefields.connectors.HarmonicConnector class method*), 27
load_opls() (*squid.forcefields.coulomb.Coul class method*), 30
load_opls() (*squid.forcefields.lj.LJ class method*), 33
load_opls() (*squid.forcefields.parameters.Parameters method*), 42
load_smrff() (*squid.forcefields.connectors.HarmonicConnector class method*), 27
load_smrff() (*squid.forcefields.coulomb.Coul class method*), 30
load_smrff() (*squid.forcefields.lj.LJ class method*), 33
load_smrff() (*squid.forcefields.morse.Morse class method*), 36
load_smrff() (*squid.forcefields.parameters.Parameters method*), 42
load_smrff() (*squid.forcefields.tersoff.Tersoff class method*), 47

M

map_to_lmp_index() (*in module squid.forcefields.helper*), 31
mapper() (*squid.forcefields.parameters.Parameters method*), 42
merge() (*squid.structures.molecule.Molecule method*), 83
mo_analysis() (*in module squid.orca.post_process*), 76
Molecule (*class in squid.structures.molecule*), 82
Morse (*class in squid.forcefields.morse*), 35
motion_per_frame() (*in module squid.geometry.spatial*), 52
mvee() (*in module squid.geometry.spatial*), 52

N

NEB (*class in squid.calcs.neb*), 20
net_charge() (*squid.structures.molecule.Molecule method*), 83
next() (*squid.lammps.parser.LMP_Parser method*), 65
num_free_parameters() (*squid.forcefields.parameters.Parameters method*), 43

O

orca_results() (*in module squid.calcs.aneb*), 18
orca_results() (*in module squid.calcs.neb*), 22
orca_start_job() (*in module squid.calcs.aneb*), 19
orca_start_job() (*in module squid.calcs.neb*), 22
orthogonal_procrustes() (*in module squid.geometry.spatial*), 52
ovito_xyz_to_gif() (*in module squid.post_process.ovito*), 78
ovito_xyz_to_image() (*in module squid.post_process.ovito*), 79

P

pack() (*squid.forcefields.connectors.HarmonicConnector method*), 28
pack() (*squid.forcefields.coulomb.Coul method*), 30
pack() (*squid.forcefields.lj.LJ method*), 34
pack() (*squid.forcefields.morse.Morse method*), 36
pack() (*squid.forcefields.parameters.Parameters method*), 43
pack() (*squid.forcefields.tersoff.Tersoff method*), 47
packmol() (*in module squid.geometry.packmol*), 51
pair_coeff_dump() (*squid.forcefields.lj.LJ method*), 34

pair_coeff_dump() (*squid.forcefields.morse.Morse method*), 37
Parameters (*class in squid.forcefields.parameters*), 38
parse_line() (*squid.forcefields.connectors.HarmonicConnector static method*), 28
parse_line() (*squid.forcefields.coulomb.Coul static method*), 30
parse_line() (*squid.forcefields.lj.LJ static method*), 34
parse_line() (*squid.forcefields.morse.Morse static method*), 37
parse_line() (*squid.forcefields.tersoff.Tersoff static method*), 47
parse_pfile() (*in module squid.forcefields.opls*), 38
parse_pfile() (*in module squid.forcefields.smrff*), 44
perturbate() (*in module squid.geometry.transform*), 54
plot_electrostatic_from_cube() (*in module squid.post_process.vmd*), 80
plot_MO_from_cube() (*in module squid.post_process.vmd*), 79
pot_analysis() (*in module squid.orca.post_process*), 77
print_lower() (*squid.forcefields.coulomb.Coul method*), 31
print_lower() (*squid.forcefields.lj.LJ method*), 34
print_lower() (*squid.forcefields.morse.Morse method*), 37
print_lower() (*squid.forcefields.tersoff.Tersoff method*), 48
print_upper() (*squid.forcefields.coulomb.Coul method*), 31
print_upper() (*squid.forcefields.lj.LJ method*), 34
print_upper() (*squid.forcefields.morse.Morse method*), 37
print_upper() (*squid.forcefields.tersoff.Tersoff method*), 48
printer() (*squid.forcefields.connectors.HarmonicConnector method*), 28
printProgressBar() (*in module squid.utils.print_helper*), 91
procrustes() (*in module squid.geometry.transform*), 55
pysub() (*in module squid.jobs.submission*), 61

Q

quick_min() (*in module squid.optimizers.quick_min*), 71

R

random_in_range() (*in module squid.forcefields.helper*), 32
random_rotation_matrix() (*in module squid.geometry.spatial*), 53
read() (*in module squid.orca.io*), 73
read_cml() (*in module squid.files.cml_io*), 23
read_dump() (*in module squid.lammps.io.dump*), 63
read_dump_gen() (*in module squid.lammps.io.dump*), 63
read_xyz() (*in module squid.files.xyz_io*), 24
read_xyz_gen() (*in module squid.files.xyz_io*), 25
reassign_indices() (*squid.structures.molecule.Molecule method*), 83
reassign_indices() (*squid.structures.system.System method*), 87
rotate() (*squid.structures.molecule.Molecule method*), 83
rotate() (*squid.structures.system.System method*), 88
rotate_atoms() (*in module squid.geometry.misc*), 50
rotation_matrix() (*in module squid.geometry.spatial*), 53

S

scale() (*squid.structures.atom.Atom method*), 81
scale() (*squid.structures.molecule.Molecule method*), 83
set_all_masks() (*squid.forcefields.parameters.Parameters method*), 43
set_binder() (*squid.forcefields.morse.Morse method*), 37
set_default_bounds() (*squid.forcefields.tersoff.Tersoff method*), 48
set_mask() (*squid.forcefields.parameters.Parameters method*), 43
set_nonbinder() (*squid.forcefields.morse.Morse method*), 37
set_opls_mask() (*squid.forcefields.parameters.Parameters method*), 43
set_position() (*squid.structures.atom.Atom method*), 81
set_positions() (*squid.structures.molecule.Molecule method*), 83
set_smoothed_pair_potentials() (*squid.forcefields.parameters.Parameters method*), 43
set_types() (*squid.structures.system.System method*), 88
sim_out (*class in squid.structures.results*), 85
simplify_numerical_array() (*in module squid.utils.cast*), 90
smooth_xyz() (*in module squid.geometry.transform*), 55

sorted_force_2body_symmetry() (*in module squid.forcefields.tersoff*), 49
spaced_print() (*in module squid.utils.print_helper*), 91
squid.calcs.aneb (*module*), 15
squid.calcs.neb (*module*), 19
squid.files.cml_io (*module*), 23
squid.files.misc (*module*), 24
squid.files.xyz_io (*module*), 24
squid.forcefields.connectors (*module*), 25
squid.forcefields.coulomb (*module*), 29
squid.forcefields.helper (*module*), 31
squid.forcefields.lj (*module*), 32
squid.forcefields.morse (*module*), 35
squid.forcefields.opls (*module*), 38
squid.forcefields.parameters (*module*), 38
squid.forcefields.smrff (*module*), 44
squid.forcefields.tersoff (*module*), 45
squid.geometry.misc (*module*), 50
squid.geometry.packmol (*module*), 51
squid.geometry.spatial (*module*), 52
squid.geometry.transform (*module*), 54
squid.jobs.container (*module*), 56
squid.jobs.nbs (*module*), 56
squid.jobs.queue_manager (*module*), 58
squid.jobs.slurm (*module*), 60
squid.jobs.submission (*module*), 61
squid.lammps.io.data (*module*), 62
squid.lammps.io.dump (*module*), 63
squid.lammps.io.thermo (*module*), 63
squid.lammps.job (*module*), 63
squid.lammps.parser (*module*), 64
squid.maths.lhs (*module*), 66
squid.optimizers.bfgs (*module*), 66
squid.optimizers.conjugate_gradient (*module*), 68
squid.optimizers.fire (*module*), 69
squid.optimizers.lbfgs (*module*), 69
squid.optimizers.quick_min (*module*), 71
squid.optimizers.steepest_descent (*module*), 71
squid.orca.io (*module*), 72
squid.orca.job (*module*), 73
squid.orca.mep (*module*), 75
squid.orca.post_process (*module*), 76
squid.orca.utils (*module*), 77
squid.post_process.debyer (*module*), 77
squid.post_process.ovito (*module*), 78
squid.post_process.vmd (*module*), 79
squid.structures.atom (*module*), 80
squid.structures.molecule (*module*), 82
squid.structures.results (*module*), 84
squid.structures.system (*module*), 85
squid.structures.topology (*module*), 88
squid.utils.cast (*module*), 89
squid.utils.print_helper (*module*), 90
squid.utils.units (*module*), 92
steepest_descent() (*in module squid.optimizers.steepest_descent*), 71
strip_color() (*in module squid.utils.print_helper*), 91
strip_colour() (*in module squid.utils.print_helper*), 91
submit_job() (*in module squid.jobs.nbs*), 57
submit_job() (*in module squid.jobs.slurm*), 61
submit_job() (*in module squid.jobs.submission*), 62
System (*class in squid.structures.system*), 85

T

tag_tersoff_for_duplicate_2bodies() (*in module squid.forcefields.tersoff*), 49
Tersoff (*class in squid.forcefields.tersoff*), 45
translate() (*squid.structures.atom.Atom method*), 81
translate() (*squid.structures.molecule.Molecule method*), 84
turn_off() (*squid.forcefields.tersoff.Tersoff method*), 48
turn_off_3body() (*squid.forcefields.tersoff.Tersoff method*), 48

U

unpack() (*squid.forcefields.connectors.HarmonicConnector method*), 28
unpack() (*squid.forcefields.coulomb.Coul method*), 31
unpack() (*squid.forcefields.lj.LJ method*), 34
unpack() (*squid.forcefields.morse.Morse method*), 37
unpack() (*squid.forcefields.parameters.Parameters method*), 44
unpack() (*squid.forcefields.tersoff.Tersoff method*), 49
unravel() (*squid.structures.atom.Atom method*), 81
update_2body() (*squid.forcefields.tersoff.Tersoff method*), 49

V

validate() (*squid.forcefields.connectors.HarmonicConnector method*), 28
validate() (*squid.forcefields.coulomb.Coul method*), 31
validate() (*squid.forcefields.lj.LJ method*), 35
validate() (*squid.forcefields.morse.Morse method*), 38
validate() (*squid.forcefields.tersoff.Tersoff method*), 49
verify_tersoff_2body_symmetry() (*in module squid.forcefields.tersoff*), 50

W

wait() (*squid.jobs.container.JobObject method*), 56
which() (*in module squid.files.misc*), 24
write() (*squid.lammps.parser.LMP_Parser method*), 65
write_cml() (*in module squid.files.cml_io*), 23
write_lammps_data() (*in module squid.lammps.io.data*), 62
write_smrff() (*squid.forcefields.parameters.Parameters method*), 44
write_xyz() (*in module squid.files.xyz_io*), 25

Z

zhi (*squid.structures.system.System attribute*), 88