Squid Documentation

Release 2.0.0

Henry Herbol

Jul 18, 2019

CONTENTS

Contents:

CHAPTER

ONE

SQUID

NOTICE - SQUID IS CURRENTLY UNDERGOUND AN UPDATE! The website and documentation is under development, with projected completion sometime Friday, July 19th, 2019. Please bare with us.

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

CODEBASE

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 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

2.3 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:

```python
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

2.4 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.

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)}


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()*

---

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,*** **`structures.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,*** **`structures.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,*** **`structures.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,*** **`structures.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: **`jobs.Job`** 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,*** **`structures.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,*** **`structures.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,*** **`structures.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: **`jobs.Job`** 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*, *procs=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,*** **`structures.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.

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, 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,*** **`structures.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,*** **`structures.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*, *procs*, *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,*** **`structures.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.

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.

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

Returns

g09_job: **`jobs.Job`** 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,*** **`structures.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,*** **`structures.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*, *procs*, *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,*** **`structures.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.

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

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

Returns

orca_job: `jobs.Job` A job container holding the orca simulation.

CHAPTER

FOUR

CONSOLE SCRIPTS

4.1 chkDFT

INFO HERE

4.2 scanDFT

INFO HERE

4.3 procrustes

INFO HERE

4.4 pysub

INFO HERE

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

CHAPTER

SIX

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

INDEX