Squid Documentation

Release 2.0.0

Henry Herbol

Jul 18, 2019

CONTENTS

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

CONSOLE SCRIPTS

3.1 chkDFT

INFO HERE

3.2 scanDFT

INFO HERE

3.3 procrustes

INFO HERE

3.4 pysub

INFO HERE

CHAPTER

FOUR

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

4.1 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",
        procs=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
```

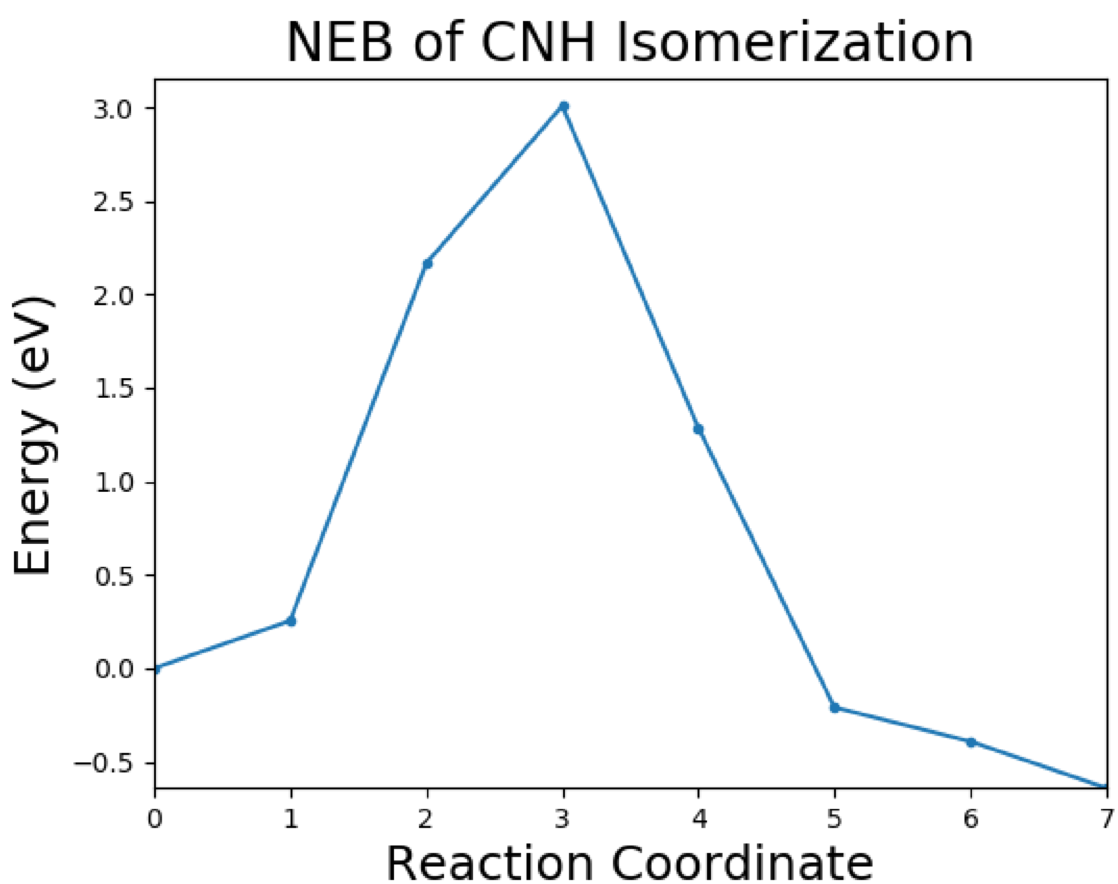
(continues on next page)

(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
```



4.2 Molecular Orbital Visualization Demo

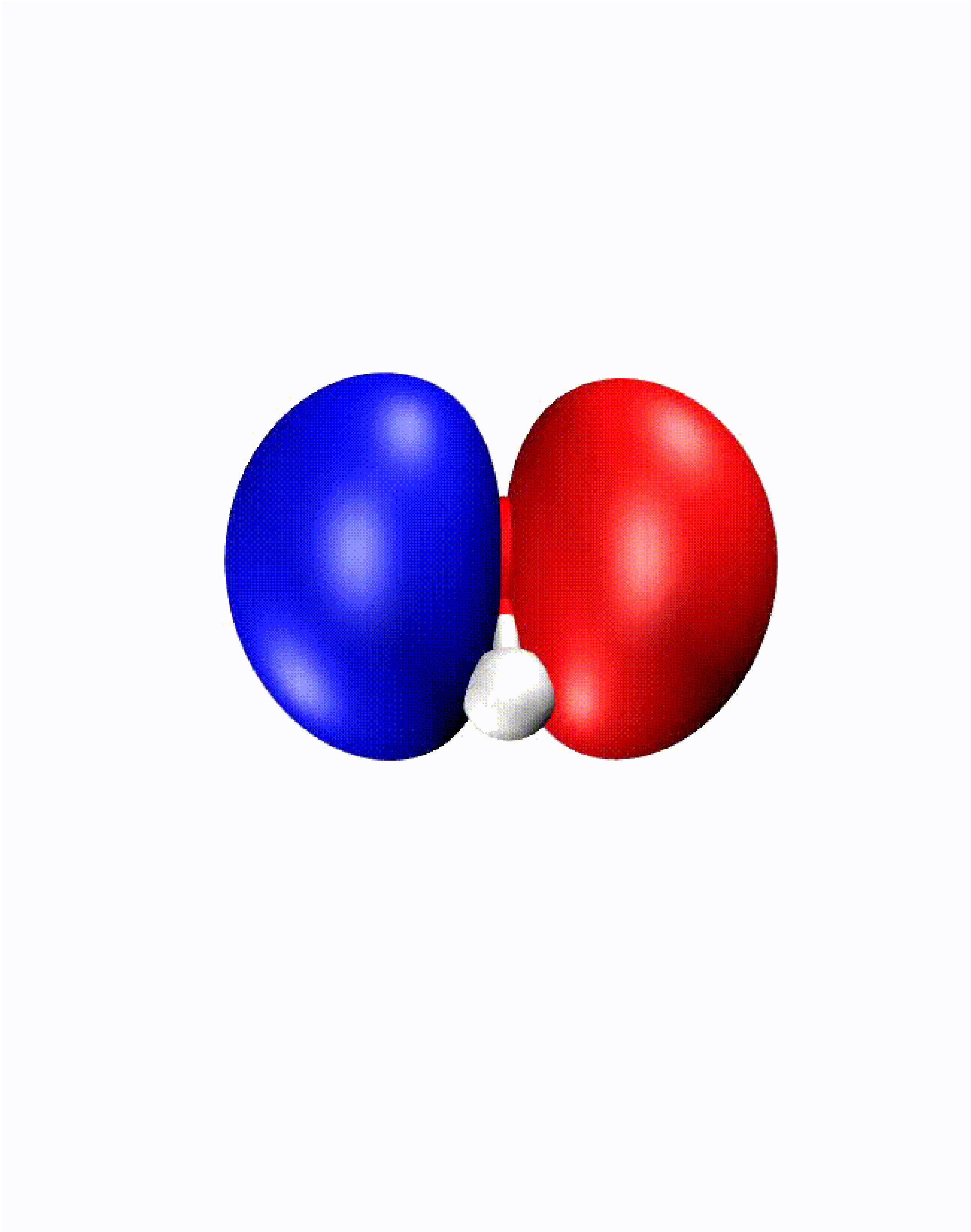
The below code shows how one can visualize molecular orbitals of a molecule (in this case water) using VMD.

```
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
```

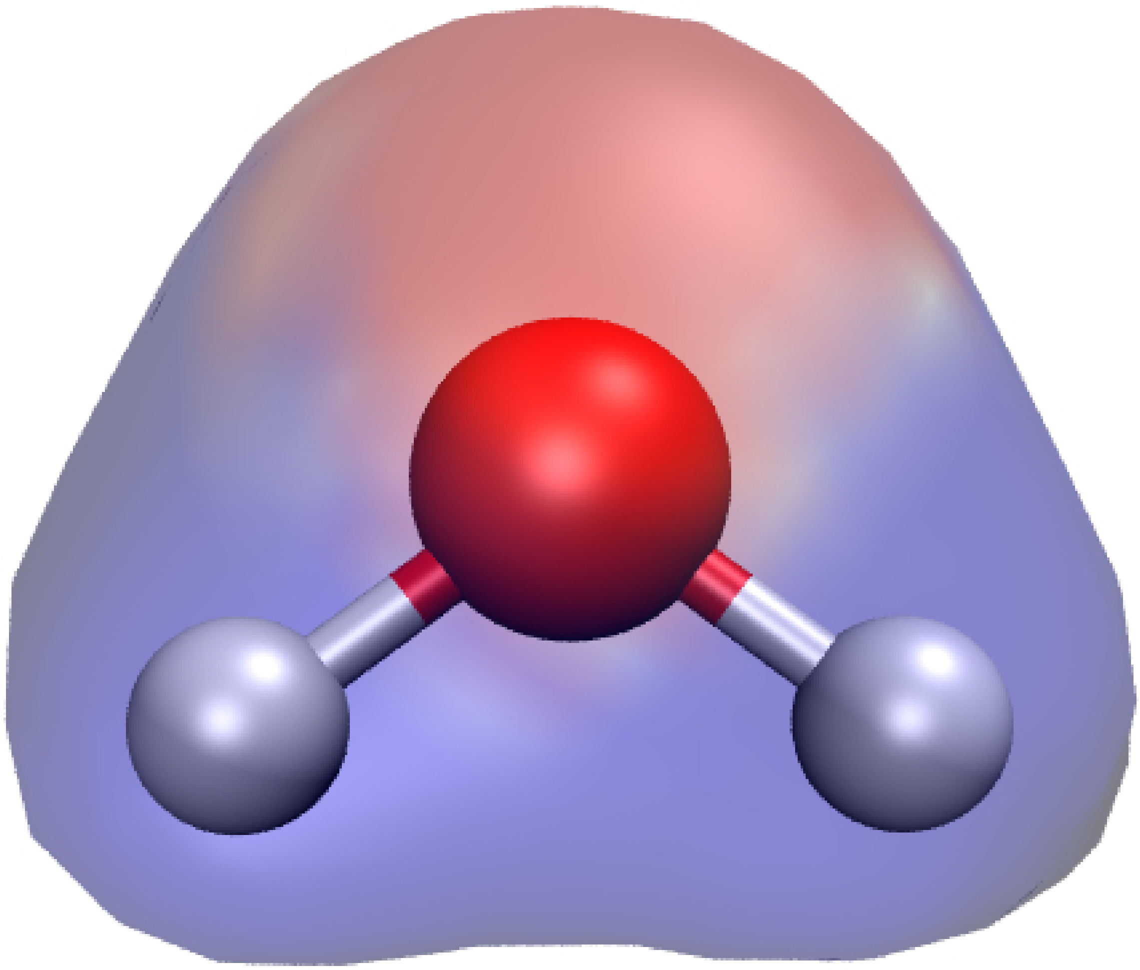
Choosing only displays 1, 3, and 5 we can see the HOMO level of water as follows (positive being blue and negative being red):



4.3 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
    )
```



4.4 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...")
```

(continues on next page)

(continued from previous page)

```
    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.

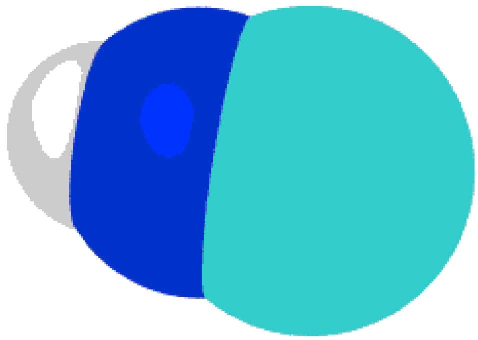
```
import os
import shutil
from squid import files
from squid import geometry
from squid import structures
from squid.post_process.ovito import ovito_xyz_to_gif


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)
```

(continues on next page)

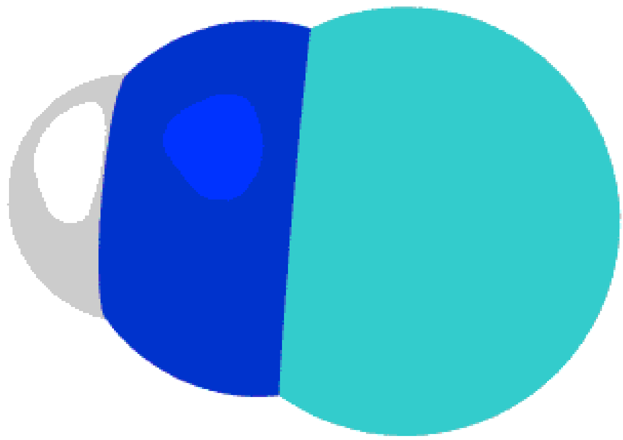
(continued from previous page)

```
    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()
```

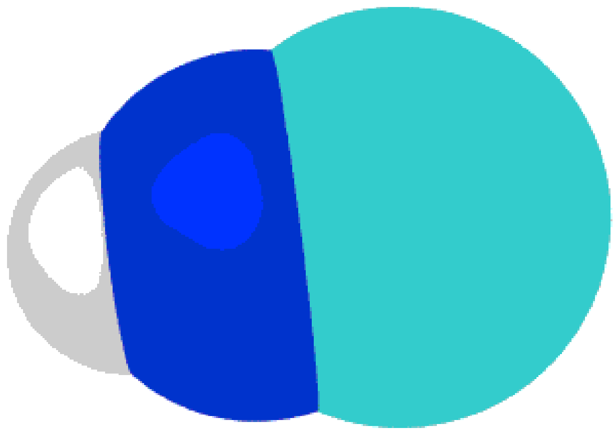
The rough reaction coordinate with rotations, shown below, would not lend itself to linear interpolation, as neighbouring frames would lead to atoms overlapping.



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.



4.5 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, procs=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
    lammps.write_lammps_data(world)
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search