
pysisyphus Documentation

Release 0.8.0b1.dev118+g3e6d331

Johannes Steinmetzer

May 16, 2024

CONTENTS:

| | | |
|----------|--|-----------|
| 1 | Overview of pysisyphus | 3 |
| 1.1 | Entry points | 4 |
| 1.1.1 | pysis | 4 |
| 1.1.2 | pysisplot | 4 |
| 1.1.3 | pysistrj | 5 |
| 1.2 | Available calculators | 5 |
| 1.2.1 | Excited state capabilities | 5 |
| 1.2.2 | Ground states capabilities | 5 |
| 1.2.3 | Pure python calculators & Wrappers | 5 |
| 1.3 | Available algorithms | 6 |
| 1.3.1 | Chain Of States Methods | 6 |
| 1.3.2 | Chain Of States Optimizer | 6 |
| 1.3.3 | Transition state optimization | 6 |
| 1.3.4 | Intrinsic Reaction Coordinate integrators | 6 |
| 1.4 | Additional remarks | 7 |
| 2 | Installation | 9 |
| 2.1 | Preparing an environment | 9 |
| 2.2 | Installation from PyPI with pip | 9 |
| 2.3 | Installation from source | 10 |
| 2.4 | Setting up .pysisyphusrc | 10 |
| 2.5 | Example runpsi4.sh | 12 |
| 2.6 | Example runmopac.sh | 12 |
| 2.7 | Example runcfour.sh | 12 |
| 2.8 | Verifying Installation | 13 |
| 3 | Installation via Nix | 15 |
| 3.1 | Prerequisites | 15 |
| 3.2 | Caching | 15 |
| 3.3 | Usage | 16 |
| 4 | Worked example | 17 |
| 5 | Coordinate Systems | 29 |
| 5.1 | Supported Coordinate Systems | 29 |
| 5.1.1 | Cartesian Coordinates | 29 |
| 5.1.2 | Redundant Internal Coordinates | 29 |
| 5.1.3 | Delocalized Internal Coordinates (DLC) | 30 |
| 5.1.4 | Translation & Rotation Internal Coordinates (TRIC) | 30 |
| 5.2 | Supported File Formats | 30 |

| | | |
|----------|---|-----------|
| 5.2.1 | Z-Matrix example | 30 |
| 5.3 | YAML Input | 31 |
| 5.3.1 | Inline input | 31 |
| 5.3.2 | Types of Primitive Coordinates | 31 |
| 5.3.3 | Define Additional Primitives | 33 |
| 5.3.4 | Freeze Atoms | 33 |
| 5.3.5 | Constraints | 34 |
| 5.3.6 | Isotopes | 34 |
| 5.4 | Geometry & RedundantCoords | 35 |
| 5.5 | Related Literature | 45 |
| 6 | Calculators | 47 |
| 6.1 | YAML input | 47 |
| 6.2 | Calculator base classes | 48 |
| 6.2.1 | Paramters | 50 |
| 6.3 | OverlapCalculator base class | 53 |
| 6.4 | Calculators with Excited state capabilities | 55 |
| 6.4.1 | Gaussian09 | 55 |
| 6.4.2 | Gaussian16 | 56 |
| 6.4.3 | OpenMolcas | 57 |
| 6.4.4 | ORCA 4.2.1 / 5.0.1 | 59 |
| 6.4.5 | PySCF 1.7.6 | 61 |
| 6.4.6 | Turbomole 7.x | 61 |
| 6.4.7 | DFTB+ 20.x | 63 |
| 6.5 | Calculators with Ground state capabilities | 64 |
| 6.5.1 | MOPAC 2016 | 64 |
| 6.5.2 | Psi4 | 65 |
| 6.5.3 | QCEngine | 66 |
| 6.5.4 | XTB 6.x | 66 |
| 6.5.5 | Dalton | 68 |
| 6.5.6 | OpenBabel | 68 |
| 6.5.7 | CFOUR | 68 |
| 6.5.7.1 | Paramters | 69 |
| 6.6 | Meta (wrapping) Calculators | 70 |
| 6.6.1 | ExternalPotential | 70 |
| 6.6.2 | Restraint | 70 |
| 6.6.3 | HarmonicSphere | 71 |
| 6.6.4 | LogFermi | 71 |
| 6.6.5 | RMSD | 71 |
| 6.6.6 | DFT-D3 | 71 |
| 6.6.7 | AFIR | 72 |
| 6.6.8 | ONIOM | 74 |
| 6.6.9 | Dimer | 76 |
| 6.7 | Pure Python calculators | 78 |
| 6.7.1 | Sympy 2D Potentials | 78 |
| 6.7.2 | Lennard-Jones | 79 |
| 6.7.3 | FakeASE | 79 |
| 6.7.4 | TIP3P | 79 |
| 7 | Minimization | 81 |
| 7.1 | YAML Example | 82 |
| 7.2 | Convergence Thresholds | 83 |
| 7.3 | Available Optimizers | 84 |

| | | |
|-----------|--|------------|
| 8 | Minimization with Microiterations | 91 |
| 8.1 | YAML input | 92 |
| 8.2 | Example using pysisyphus' ONIOM | 93 |
| 8.3 | Example with sockets & i-PI-protocol | 95 |
| 9 | Relaxed Scans | 99 |
| 10 | Excited State Optimization | 101 |
| 10.1 | YAML Example | 101 |
| 10.2 | Optimization of Conical Intersections | 103 |
| 11 | Chain Of States Methods | 105 |
| 11.1 | String method | 105 |
| 11.2 | Nudged Elastic Band | 105 |
| 11.3 | General remarks | 106 |
| 11.4 | YAML example(s) | 106 |
| 11.5 | General advice for COS optimizations | 108 |
| 11.6 | Chain Of States base class | 108 |
| 11.7 | Chain Of State Methods | 110 |
| 11.7.1 | Nudged Elastic Band (NEB) | 110 |
| 11.7.2 | Adaptive NEB | 111 |
| 11.7.3 | Free-End NEB | 112 |
| 11.7.4 | Simple Zero-Temperature String | 112 |
| 11.8 | Growing Chain Of States base class | 112 |
| 11.9 | Growing Chain Of State Methods | 113 |
| 11.9.1 | Growing String Method | 113 |
| 12 | Transition State Optimization | 115 |
| 12.1 | Hessian Based TS Optimization | 115 |
| 12.1.1 | YAML example | 115 |
| 12.2 | Dimer Method | 117 |
| 12.2.1 | YAML example | 117 |
| 12.3 | General advice for TS optimizations | 118 |
| 12.4 | TSHessianOptimizer base class | 118 |
| 12.5 | TS-Optimizer using hessian information | 120 |
| 12.5.1 | Restricted-Step Partitioned-RFO | 120 |
| 12.5.2 | Restricted-Step Image-Function-RFO | 120 |
| 12.5.3 | Trust-Region Image-Function Method | 120 |
| 12.6 | TS-Optimization without hessian information | 121 |
| 12.6.1 | Dimer method | 121 |
| 13 | Intrinsic Reaction Coordinate (IRC) | 123 |
| 13.1 | YAML example(s) | 123 |
| 13.2 | IRC base class | 124 |
| 13.3 | IRC Integrators | 126 |
| 13.3.1 | Damped-Velocity-Verlet integrator | 126 |
| 13.3.2 | Euler integrator | 127 |
| 13.3.3 | Euler-Predictor-Corrector integrator | 127 |
| 13.3.4 | Gonzales-Schlegel-2 integrator | 127 |
| 13.3.5 | Local-Quadratic-Approximation integrator | 128 |
| 13.3.6 | Modified-Ishida-Morokuma-Komornicki integrator | 128 |
| 13.3.7 | Runge-Kutta-4 integrator | 128 |
| 14 | Diabatization | 129 |
| 14.1 | General Algorithm | 129 |

| | | |
|-----------|--|------------|
| 14.2 | Example | 131 |
| 14.3 | YAML input | 132 |
| 15 | Plotting | 135 |
| 15.1 | Example - Diels-Alder reaction | 135 |
| 15.1.1 | Plotting optimization progress | 136 |
| 15.1.2 | Plotting COS optimization progress | 136 |
| 15.1.3 | Plotting TS-optimization progress | 142 |
| 15.1.4 | Plotting the Intrinsic Reaction Coordinate | 142 |
| 15.2 | Example - AFIR | 144 |
| 15.3 | Example - Excited State Tracking | 145 |
| 16 | Working with Geometries | 149 |
| 17 | pysisyphus | 151 |
| 17.1 | pysisyphus package | 151 |
| 17.1.1 | Subpackages | 151 |
| 17.1.1.1 | pysisyphus.benchmarks package | 151 |
| 17.1.1.2 | pysisyphus.calculators package | 152 |
| 17.1.1.3 | pysisyphus.cos package | 214 |
| 17.1.1.4 | pysisyphus.db package | 221 |
| 17.1.1.5 | pysisyphus.drivers package | 222 |
| 17.1.1.6 | pysisyphus.dynamics package | 231 |
| 17.1.1.7 | pysisyphus.intcoords package | 237 |
| 17.1.1.8 | pysisyphus.interpolate package | 263 |
| 17.1.1.9 | pysisyphus.io package | 264 |
| 17.1.1.10 | pysisyphus.irc package | 267 |
| 17.1.1.11 | pysisyphus.line_searches package | 278 |
| 17.1.1.12 | pysisyphus.modelfollow package | 282 |
| 17.1.1.13 | pysisyphus.optimizers package | 284 |
| 17.1.1.14 | pysisyphus.plotters package | 302 |
| 17.1.1.15 | pysisyphus.stochastic package | 302 |
| 17.1.1.16 | pysisyphus.tests package | 305 |
| 17.1.1.17 | pysisyphus.tsoptimizers package | 305 |
| 17.1.1.18 | pysisyphus.wrapper package | 308 |
| 17.1.2 | Submodules | 309 |
| 17.1.3 | pysisyphus.Geometry module | 309 |
| 17.1.4 | pysisyphus.TableFormatter module | 317 |
| 17.1.5 | pysisyphus.TablePrinter module | 317 |
| 17.1.6 | pysisyphus.color module | 318 |
| 17.1.7 | pysisyphus.config module | 318 |
| 17.1.8 | pysisyphus.constants module | 318 |
| 17.1.9 | pysisyphus.elem_data module | 318 |
| 17.1.10 | pysisyphus.exceptions module | 318 |
| 17.1.11 | pysisyphus.filtertrj module | 318 |
| 17.1.12 | pysisyphus.helpers module | 318 |
| 17.1.13 | pysisyphus.helpers_pure module | 320 |
| 17.1.14 | pysisyphus.init_logging module | 322 |
| 17.1.15 | pysisyphus.linalg module | 323 |
| 17.1.16 | pysisyphus.pack module | 325 |
| 17.1.17 | pysisyphus.peakdetect module | 325 |
| 17.1.18 | pysisyphus.plot module | 328 |
| 17.1.19 | pysisyphus.run module | 329 |
| 17.1.20 | pysisyphus.socket_helper module | 331 |

| | | |
|-----------|---------------------------------------|------------|
| 17.1.21 | pysisyphus.testing module | 332 |
| 17.1.22 | pysisyphus.thermo module | 332 |
| 17.1.23 | pysisyphus.trj module | 332 |
| 17.1.24 | pysisyphus.version module | 333 |
| 17.1.25 | pysisyphus.xyzloader module | 333 |
| 17.1.25.1 | Paramters | 333 |
| 17.1.26 | pysisyphus.yaml_mods module | 334 |
| 17.1.27 | Module contents | 334 |
| 18 | Indices and tables | 335 |
| | Python Module Index | 337 |
| | Index | 341 |

pysisyphus is a software-suite for the exploration of potential energy surfaces in ground- and **excited states**. It implements several methods to search for stationary points (minima and first order saddle points) and the calculation of minimum energy paths by means of IRC and Chain of States methods like Nudged Elastic Band and Growing String. Furthermore it provides tools to easily analyze & modify geometries (aligning, translating, **interpolating**, ...) and to visualize the calculation results/progress.

The required energies, gradients and hessians are calculated by calling external quantum chemistry codes. *pysisyphus* can also be used as a library to implement custom quantum chemistry workflows.

If any issues arise please open an [issue](#) and I'll see if it can be fixed and my time permits it. Contrubtions are welcome, so feel free to submit a PR.

This software is still work in progress. Use at your own risk. Also take a look at the [license](#)

OVERVIEW OF PYSISYPHUS

pysisyphus is a software-suite for the exploration of potential energy surfaces in ground- and **excited states**. User input is read from YAML files, but it can also be used as a python library to set up custom workflows.

Below you can find a screencast of the transition state (TS) search for the famous alanine dipeptide isomerization at the *xtb* level of theory. It starts with pre-optimizations of the initial and final geometry, a subsequent growing string calculation in delocalized internal coordinates to generate a guess for the final TS optimization and with which it concludes.

```
14      0.000398      0.000109      0.008266      0.002214      0.1
Converged!

Final summary:
  max(forces,internal): 0.000398 hartree/(bohr,rad)
  rms(forces,internal): 0.000109 hartree/(bohr,rad)
  max(forces,cartesian): 0.000590 hartree/bohr
  rms(forces,cartesian): 0.000190 hartree/bohr
  energy: -32.97379138 hartree
Wrote final, hopefully optimized, geometry to 'last_prefinal_geometry.xyz'
Preoptimization of last geometry converged!
Saved final preoptimized structure to 'last_preopt.xyz'.

Read 2 geometries.
#####
# RUNNING GROWINGSTRING #
#####

Path with 2 moving images.
Spent 0.0 s preparing the first cycle.
cycle  max(force)    rms(force)    max(step)    rms(step)    s/cycle
  0      0.020646      0.004532      0.016517      0.003626      0.3
    String=2+2 HEI=03/04 (E_max-E_0)=15.3 kJ/mol
  1      0.054177      0.008406      0.043341      0.006724      0.5
    String=3+3 HEI=04/06 (E_max-E_0)=76.1 kJ/mol
  2      0.091448      0.011890      0.073159      0.009512      0.6
    String=4+4 HEI=04/08 (E_max-E_0)=161.8 kJ/mol
```

Besides being able to track excited states over the course of an optimization the idea of *pysisyphus* is to provide tools for handling the whole process of calculating reaction paths, starting from pre-optimizing the given input geometries for chain of states methods and ending with the hessian calculation on the optimized IRC endpoints.

1.1 Entry points

pysisyphus provides several entry points that can be called from the shell (command line). The available commands of the entry points can be queried with the `-h` or `--help` arguments:

```
# Run calculations (Minima optimization, TS search, IRC, NEB, GS, ...)
pysis
# Plotting of path-energies, optimization progress, IRC progress, etc ...
pysisplot
# Manipulate a .trj file or multiple .xyz files
pysistrj
```

1.1.1 pysis

Called with a YAML-input file. Simple and more complex examples can be found in the *examples* directory of this repository. Some common arguments are given below:

```
usage: pysis [-h] [--clean] --cp CP]
           [yaml]

positional arguments:
  yaml                Start pysisyphus with input from a YAML file.

optional arguments:
  --clean             Ask for confirmation before cleaning.
  --cp CP, --copy CP Copy .yaml file and corresponding geometries to a new
                    directory. Similar to TURBOMOLEs cpc command.
```

1.1.2 pysisplot

Visualization/plotting of the pysisyphus calculations. Some common arguments are given below:

```
usage: pysisplot (--cosforces | --cosens | --all_energies | --afir | --opt | --irc | --
↪overlaps)

optional arguments:
  --cosforces, --cf  Plot image forces along a COS.
  --cosens, --ce     Plot COS energies.
  --all_energies, -a Plot ground and excited state energies from 'overlap_data.h5'.
  --afir            Plot AFIR and true -energies and -forces from an AFIR calculation.
  --opt             Plot optimization progress.
  --irc            Plot IRC progress.
  --overlaps, -o
```

1.1.3 pysistrj

Please see *pysistrj --help* for a list of available arguments.

1.2 Available calculators

1.2.1 Excited state capabilities

| Program | Gradient | Hessian | Exc. states | Version |
|------------|----------|---------|--------------------|--|
| ORCA | y | y | TD-DFT, TDA | 4.2.x |
| Turbomole | y | y | TD-DFT, TDA, ricc2 | 7.2-7.4 |
| Gaussian16 | y | y | tested TD-DFT, TDA | |
| PySCF | y | y | tested TD-DFT | 1.7.5 |
| OpenMOLCAS | y | n | &rasscf | Not yet derived from OverlapCalculator |

1.2.2 Ground states capabilities

| Program | Gradient | Hessian | Version |
|-----------|----------|---------|-----------|
| MOPAC2016 | y | y | |
| XTB | y | y | 6.3.2 |
| QCEngine | y | n | >= 0.16.0 |
| Psi4 | y | y | |

1.2.3 Pure python calculators & Wrappers

| Program | Gradient | Hessian | Comment |
|---------------|----------|---------|---|
| Sympy 2D | y | y | Many analytical potentials (LEPS, Rosenbrock, Cerjan-Miller, Muller-Brown, ...) |
| Lennard-Jones | y | n | No periodic boundary conditions |
| AFIR | y | n | |
| ONIOM | y | n | Arbitrary number of layers with multicenter-support in the highest layer. |
| FakeASE | y | n | Wraps <i>pysisyphus</i> calculators so they can be used with <i>ase</i> . |

1.3 Available algorithms

1.3.1 Chain Of States Methods

| Algorithm | Coordinates | Comment |
|--------------------------------|------------------------|--|
| Nudged Elastic Band (NEB) | Cartesian, DLC planned | Climbing Image variants, Doubly nudged variant |
| Adaptive NEB | Cartesian | Not well tested |
| Free-End NEB | Cartesian | Not well tested |
| Simple Zero-Temperature-String | Cartesian | Equal spacing, energy-dependent spacing |
| Growing String Method | Cartesian, DLC | |

1.3.2 Chain Of States Optimizer

| Algorithm | Comment | Links |
|--------------------|----------------------|--------------------------------|
| Steepest Descent | Backtracking variant | NEB-Optimizers |
| Conjugate Gradient | Backtracking variant | NEB-Optimizers |
| QuickMin | | NEB-Optimizers |
| FIRE | | NEB-Optimizers |
| BFGS | | NEB-Optimizers |

1.3.3 Transition state optimization

| Algorithm | Comment | Links |
|--------------|---------|--|
| RS-P-RFO | default | RFO-Paper , RS-Paper |
| RS-I-RFO | | RFO-Paper , RS-Paper |
| TRIM | | TRIM-Paper |
| Dimer method | | |

1.3.4 Intrinsic Reaction Coordinate integrators

| Algorithm | Comment | Links |
|-------------------------------|-----------------|--|
| Damped-Velocity-Verlet | | DVV-Paper |
| Euler | Not recommended | |
| EulerPC | default | Kaestner-PC , Euler-PC |
| Gonzales-Schlegel 2 | | GS2-Paper |
| Local Quadratic Approximation | | LQA-Paper |
| Modified IMK | | IMK-Paper |
| Runge-Kutta-4 | Not recommended | |

1.4 Additional remarks

pysisyphus uses the *tempfile* module from the python stdlib. The location of the temporary directories can be controlled by setting the **\$TMPDIR** environment variable before executing *pysis*.

```
export TMPDIR=[tmpdir]
```


INSTALLATION

2.1 Preparing an environment

It is good idea to install *pysisyphus* into a separate python environment, whether it is an [Anaconda](#) environment or a [virtualenv](#) environment, derived from the system python installation or a [pyenv](#) installation (preferred).

```
# Optional: Create separate Anaconda environment
conda create -n pysis-env python=3.9
# Activate conda environment
conda activate pysis-env

# Optional: Create separate virtual environment
python -m venv pysis-env
# On some linux distributions 'python3' is used instead of 'python', e.g. Ubuntu
# python3 -m venv pysis-env
# Activate virtual environment. Preferably, an absolute path pointing to the
# 'activate' script should be used.
source pysis-env/bin/activate
```

2.2 Installation from PyPI with pip

This installs the latest stable release as published on PyPI. If you don't want to do any development work this is the preferred way of installing *pysisyphus*.

```
python -m pip install pysisyphus
# Installation of additional optional dependencies is also possible
# python -m pip install pysisyphus[test]
```

If you run into any problems please make sure that your pip version is recent enough. A pip upgrade is achieved via:

```
python -m pip install --upgrade pip
```

2.3 Installation from source

Decide on an `$install_dir` and clone the repository from github. If you want to change the code after installation (develop) do an editable (-e) installation with pip.

```
git clone https://github.com/eljost/pysisyphus.git $install_dir
cd $install_dir
# Install with -e if you want an editable installation
python -m pip install [-e] .
# Installation of extras is also possible. 'sphinx' is only needed if you
# want to build the documentation.
# python -m pip install [-e] .[test]
```

With an editable installation it is also easy to use other branches, besides the *master* branch, e.g., the *dev* branch to test out a new feature.

```
git switch dev
```

2.4 Setting up .pysisyphusrc

pysisyphus interfaces several quantum chemistry codes and related software (Multiwfn, Jmol). Software available to *pysisyphus* is registered in `$HOME/.pysisyphusrc`, so *pysisyphus* knows which command to execute an/or where to find the binaries.

Depending on the software different choices were made how it is registered. An example `.pysisyphusrc` is given below, with a short comment for each software.

```
# Excited state calculators

[gaussian09]
# Cmd to execute. Please ensure that g09 is on your $PATH.
cmd=g09
formchk_cmd=formchk
unfchk_cmd=unfchk

[gaussian16]
# Cmds to execute. Please ensure that the binaries are found in your $PATH.
cmd=g16
formchk=formchk
unfchk=unfchk
rwfdump=rwfdump

[openmolcas]
# Cmd to execute. Please ensure that pymolcas is on your $PATH.
cmd=pymolcas

[orca]
# ORCA needs the full path to its binary, so please provide the full path.
cmd=/scratch/programme/orca_4_2_0_linux_x86-64_openmpi314/orca

#[pyscf]
```

(continues on next page)

(continued from previous page)

```

# pyscf must not have an explicit entry in the .pysisyphusrc. pysisyphus uses
# the python API of pyscf, so it is mandatory that is installed in the same environment
# as pysisyphus.

#[turbomole]
# Turbomole must not have an explicit entry in the .pysisyphusrc. The user has to take
# care that everything is set up correctly, e.g. TM-binaries are on the PATH etc...
# The respective commands are hardcoded into pysisyphus (dscf, ridft, ricc2, ...)

# Ground state calculators

[mopac]
# Similar to Psi4. An example is given below.
cmd=/user/johannes/bin/runmopac.sh

[psi4]
# As the Psi4 installation without conda is, to put it slightly, tricky it was
# decided to allow the installation of Psi4 into a separate conda environment.
# pysisyphus then creates a Psi4 input and sends it to the (bash)-script given below
# that accepts/expects one argument. It is the responsibility of the scrip to activate
# the appropriate conda environment and submit the Psi4 input. An example runpsi4.sh
# script is given below.
cmd=/user/johannes/bin/runpsi4.sh

[qcengine]
# QCEngine must not have an entry explicit entry in the .pysisyphusrc. It is used
# via its python interface and can be installed as an extra with pip (see above).
# The user is referenced to the QCEngine-documentation for any further questions.

[xtb]
# Cmd to execute. Please ensure that xtb is on your $PATH.
cmd=xtb

[openmolcas]
# Make sure that the MOLCAS variable is set.
cmd=pymolcas

[cfour]
# Parallelism is managed by the user by setting environment variables in the runscript,
# as the name of the environment variable for MPI parallelism depends on how CFOUR_MPI_
→ CORES
# was compiled. The GENBAS file must also be symlinked by the runscript.
# A sample is provided below.
cmd=/home/ghjones/bin/runcfour.sh

# Utilities

[wfoverlap]
# Cmd to execute. Please ensure that wfoverlap is on your $PATH. The binary/source
# can be obtained from https://github.com/sharc-md/sharc/tree/master/bin
cmd=/scratch/wfoverlap_1.0/bin/wfoverlap.x

```

(continues on next page)

(continued from previous page)

```
[mwfn]
# Cmd to execute. Please ensure that Multiwfn is on your $PATH. Otherwise put an
# absolute path here. By default pysisyphus looks up "Multiwfn", so if you would
# put a relative path here you don't have to, as this is already covered by the
# defaults.
cmd=Multiwfn

[jmol]
# Cmd to execute. The same arguments apply for jmol as for Multiwfn. "jmol" is
# already the default.
cmd=jmol
```

When the specified path/cmd is not absolute, but relative (e.g. for xtb, g16, ...) the corresponding binaries have to be available on the **\$PATH** and all other environment variables have to be set up correctly by the user.

2.5 Example runpsi4.sh

```
#!/bin/bash

# Afaik this doesn't work in non-interactive shells ...
# See https://github.com/conda/conda/issues/8072
# conda activate psi4
source /scratch/programme/anaconda3/bin/activate psi4
#conda activate psi4
psi4 -o stdout $1
```

2.6 Example runmopac.sh

```
#!/bin/bash

module purge
module load mopac

MOPAC2016.exe $1
```

2.7 Example runcfour.sh

```
#!/bin/bash

module purge
module load cfour
export OMP_NUM_THREADS=4
export CFOUR_MPI_CORES=1 # The name of this environment variable depends on how you
↳ compiled CFour
[ ! -f GENBAS ] && ln -s /software/cfour/basis/GENBAS
xcfour > out.log 2>&1
```

2.8 Verifying Installation

By executing `pytest -v --pyargs pysisyphus.tests` a series of quick tests can be executed, verifying successful calculator setup. Running these tests requires *pyscf* and *pytest* to be present (*pip install pyscf pytest*).

INSTALLATION VIA NIX

An easy and reliable way to get a complete pysisyphus installation, including additional quantum chemistry (QC) codes, is possible through the [Nix package manager](#). Required Nix files for the QC codes and their dependencies (linear algebra & MPI libraries etc.) are provided by the [NixOS-QChem](#) overlay.

Nix installations are fully reproducible and extremely simple to accomplish for any potential pysisyphus user, as most of the necessary configuration is already described in the respective Nix files.

The [Nix Pills](#) series provides an informative introduction to the general concepts behind Nix.

Recent stable versions of pysisyphus are also directly included in [NixOS-QChem](#). If you do not require a development version of pysisyphus, using the [NixOS-QChem](#) overlay is the preferred option. Otherwise, continue with the instructions below.

3.1 Prerequisites

Please follow the installation instructions in the [nix manual](#) to set up a working Nix installation on your system. You may want to enable the `flakes` and `nix-command` features for Nix.

3.2 Caching

The build time can be reduced drastically, if the quantum chemistry codes are fetched from a binary cache.

Pysisyphus provides a binary cache on Cachix.

Nix $\geq 2.6.0$ with flakes directly supports the binary cache without further action required.

Alternatively, if you are allowed to add binary caches in nix, you may simply execute:

```
nix run nixpkgs#cachix use pysisyphus
```

3.3 Usage

You may use pysisyphus directly via Flakes without cloning any repository or local installation. Some examples how to use pysisyphus with nix:

Inspect the structure of pysisyphus' flake:

```
nix flake show github:eljost/pysisyphus
```

Directly run the pysis command on an input :

```
nix run github:eljost/pysisyphus input.yml
```

Start an interactive shell with a pysisyphus installation

```
nix shell github:eljost/pysisyphus
```

For developing and hacking pysisyphus and for legacy Nix commands, first clone the pysisyphus repository

```
git clone https://github.com/eljost/pysisyphus.git && cd pysisyphus
```

You may obtain a development shell for pysisyphus via

```
nix develop
```

build pysisyphus in the legacy nix style

```
nix-build ./nix/default.nix
```

or obtain a development legacy-style dev-shell

```
nix-shell ./nix/shell.nix
```

The Flake also offers options to build pysisyphus with proprietary components such as ORCA or Turbomole, e.g.

```
nix run .#pysisyphusOrca
```

or build singularity or docker containers:

```
nix bundle --bundler .#toSingularityImage
```


WORKED EXAMPLE

This page provides a general overview of pysisyphus's capabilities for optimizing ground state reaction paths for the example of a tris-pericyclic reaction between a tropone derivative and dimethylfulvene.

You can find a short discussion of the reaction on Steven M. Bacharach's [blog](#). Here we focus on how to obtain TS1. Instead of using DFT as in the original [publication](#) we'll employ the semi-empirical tight binding code `xtb`. It allows us to run all necessary calculations on common desktop hardware in little time (2 min, using 6 physical CPU cores).

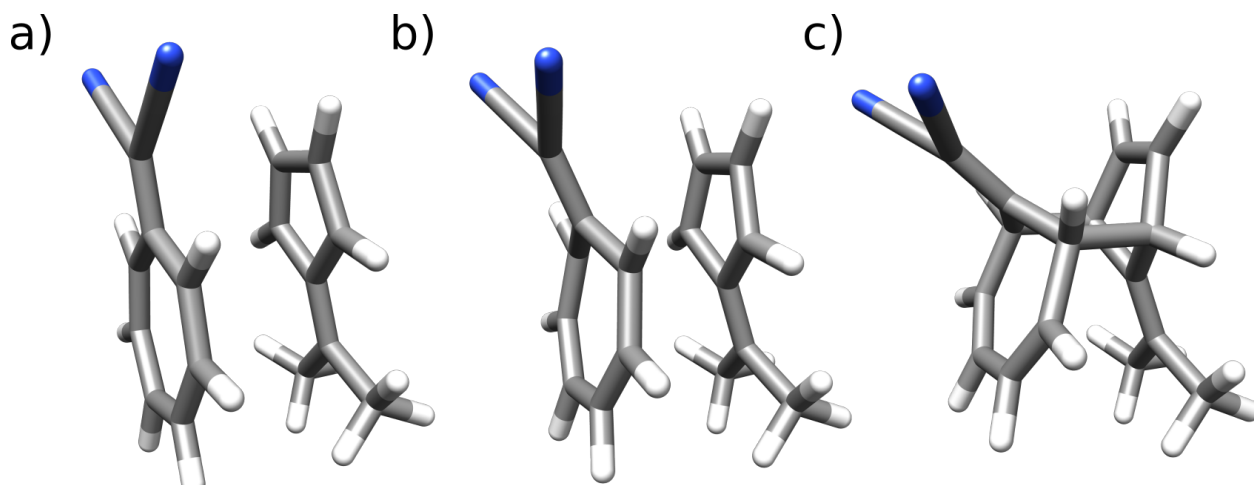


Fig. 4.1: Educts, TS and products of the tris-pericyclic reaction between a tropone derivative and dimethylfulvene.

At first, we have to create appropriate educt and product geometries using the molecular editor of our choice, e.g., [avogadro](#) or [TmoleX](#). Given a reasonable educt geometry a suitable product geometry can be constructed by decreasing the distance between the two ring systems and optimizing the result, e.g., by `xtb`. Sometimes it may be easier to start with the product geometry, and obtain the educt from it. **Consistent atom ordering in both geometries is mandatory!** Usually it's a bad idea to construct both geometries independently, as it is easy to mess up the atom ordering.

Given educts (*min_xtbopt.xyz*) and product (*prod_xtbopt.xyz*) we can create the YAML input. Our goal is to obtain the barrier heights for the reaction shown in [Fig. 4.1](#) by means of growing string (GS) calculation, transition state (TS) optimization, intrinsic reaction coordinate (IRC) calculation and subsequent optimization of the IRC endpoints. The full input is shown below. We'll go over it step by step.

```
geom:
  type: dlc
  fn: [min_xtbopt.xyz, prod_xtbopt.xyz]
preopt:
cos:
  type: gs
```

(continues on next page)

(continued from previous page)

```
max_nodes: 18
climb: True
opt:
  type: string
  align: False
  max_cycles: 20
tsopt:
  type: rsirfo
  do_hess: True
  thresh: gau
  hessian_recalc: 3
irc:
  type: eulerpc
  rms_grad_thresh: 0.0005
endopt:
calc:
  type: xtb
pal: 6
```

The desired coordinate system and the file names of the input geometries are given in the geom block.

```
geom:
  type: dlc
  fn: [min_xtbopt.xyz, prod_xtbopt.xyz]
```

Here we chose delocalized internal coordinates (DLC) for our GS, which is the preferred way. Alternatively Cartesian coordinates could be used by `type: cart`. If DLCs fail, Cartesian coordinates should be tried.

```
preopt:
```

The `preopt` block is given without any additional keywords, so sane defaults will be used for preoptimizing educt and product geometries (Rational function optimization (RFO) in redundant internal coordinates). Strictly, in our case pre-optimization is not necessary, as we already preoptimized the geometries using `xtb`. But in general it is advised to span chain of states (COS) like GS or nudged elastic band (NEB) between stationary points on the potential energy surface. If the educts and products are NOT covalently bound it may be a good idea to restrict the number of preoptimization cycles to a small number (`max_cycles: 10`), as these optimizations are sometimes hard to converge. Please see [Optimization of Minima](#) for a list of possible keywords in the `preopt` block.

```
cos:
  type: gs
  max_nodes: 18
  climb: True
```

The `cos` block configures COS calculations. Here we request a GS (gs) with 18 nodes (images) between educt and product, resulting in a total string length of 20 nodes. By default first and last node (educt and product) are kept fixed throughout the optimization. We enable a climbing image (CI) to obtain a better TS guess. Please see [Chain Of States Methods](#) for further information on COS methods.

```
opt:
  type: string
  align: False
  max_cycles: 20
```

COS/GS optimization is controlled via the `opt` block. For GS optimization one should always use `type: string`.

In internal coordinates we disable automated geometry alignment, as it is not needed. We also restrict the number of optimization cycles to 20 (default 50). The chosen optimizer will do steepest descent (SD) steps when the string grew in the previous cycle, otherwise conjugate gradient (CG) steps are used. When the GS is fully grown/connected the optimizer will use limited-memory Broyden-FletcherGoldfarb-Shanno (L-BFGS) to determine more sophisticated steps.

```
tsopt:
  type: rsirfo
  do_hess: True
  thresh: gau
  hessian_recalc: 3
```

After GS convergence the highest energy image (HEI) is determined by cubic splining and used as guess for a classical TS optimization using restricted step image RFO (RSIRFO). `do_hess: True` requests a frequency calculation after the TS optimization. The Hessian is recalculated every 3th step. When the Hessian for the chosen computational method is reasonably cheap it is a good idea to recalculate it periodically. Between recalculations it's updated using the Bofill-update. Convergence criteria are tightened from the default `thresh: gau_loose` to `thresh: gau`.

```
irc:
  type: eulerpc
  rms_grad_thresh: 0.0005
```

IRC integration is controlled in the `irc` block. By default the Euler-predictor-corrector (EulerPC) integrator is used. Integration is terminated when the root-mean-square (RMS) of the gradient is equal to or less than 0.0005 au. Possible inputs are given in *Intrinsic Reaction Coordinate (IRC)*.

```
endopt:
```

Similar to `preopt` the `endopt` will be executed with default arguments. It is used to optimize the IRC endpoints to stationary points and enables printing of additional information like RMS deviation of atomic positions (RMSD) between optimized endpoints and initial geometries. The RMSD values help in deciding if the obtained TS actually connects presumed educts and products.

```
calc:
  type: xtb
  pal: 6
```

The `calc` block configures the level of theory used in energy/gradient/Hessian calculations. Here we chose `xtb` and requested 6 CPU cores. Additional inputs for `xtb` can be found in the *xtb module documentation*

With everything set up we are ready to actually execute pysisyphus! Assuming the above YAML is saved to `01_pericyclic.yaml` just run

```
pysis 01_pericyclic.yaml | tee pysis.log
```

By default pysisyphus prints to STDOUT so you have to capture STDOUT explicitly. We use `tee` so everything is logged to a file and printed simulatenously. A lot of files and output will be produced so we will go over everything slowly.

```

                                d8b      888
                                Y8P      888
                                         888
888888b.  888 888 .d88888b 888 .d88888b 888 888 888888b. 888888b. 888 888 .d88888b
888 "88b 888 888 88K      888 88K      888 888 888 "88b 888 "88b 888 888 88K
888 888 888 888 "Y88888b. 888 "Y88888b. 888 888 888 888 888 888 888 "Y88888b.
```

(continues on next page)

(continued from previous page)

```

888 d88P Y88b 888      X88 888      X88 Y88b 888 888 d88P 888 888 Y88b 888      X88
888888P"  "Y88888 888888P' 888 888888P' "Y88888 888888P" 888 888 "Y88888 888888P'
888      888      888 888
888      Y8b d88P      Y8b d88P 888
888      "Y88P"      "Y88P" 888

```

Version 0.5.0.post1+450.g2c1654d3 (Python 3.8.5, NumPy 1.19.2, SciPy 1.5.2)

Git commit 2c1654d35d69e7b48ac4e9b00d38afd58a8bedd4

Executed at Tue Oct 6 10:08:30 2020 on 'your fancy hostname'

If pysisyphus benefitted your research please cite:

<https://doi.org/10.1002/qua.26390>

Good luck!

You will be greeted by a banner and some information about your current installation, which hopefully aids in reproducing your results later on, if needed. Then your input is repeated, including default values that you did not explicitly set. There you can also see the default values chosen for `preopt` and `endopt`.

```

{'calc': {'pal': 6, 'type': 'xtb'},
'cos': {'climb': True, 'fix_ends': True, 'max_nodes': 18, 'type': 'gs'},
'endopt': {'dump': True,
           'fragments': False,
           'max_cycles': 100,
           'overachieve_factor': 3,
           'thresh': 'gau',
           'type': 'rfo'},
'geom': {'fn': ['min_xtbopt.xyz', 'prod_xtbopt.xyz'], 'type': 'dlc'},
'interpol': {'between': 0, 'type': None},
'irc': {'rms_grad_thresh': 0.0005, 'type': 'eulerpc'},
'opt': {'align': False, 'dump': True, 'max_cycles': 30, 'type': 'string'},
'preopt': {'dump': True,
           'max_cycles': 100,
           'overachieve_factor': 3,
           'preopt': 'both',
           'strict': False,
           'thresh': 'gau_loose',
           'trust_max': 0.3,
           'type': 'rfo'},
'tsopt': {'do_hess': True,
          'dump': True,
          'h5_group_name': 'tsopt',
          'hessian_recalc': 3,
          'overachieve_factor': 3,
          'thresh': 'gau',
          'type': 'rsirfo'}}

```

The whole run starts with preoptimizations of educt and product. Both optimizations converge quickly, as the geometries are already preoptimized.

```
#####
```

(continues on next page)

(continued from previous page)

```

# RUNNING FIRST PREOPTIMIZATION #
#####
Spent 0.0 s preparing the first cycle.
cycle  max(force)  rms(force)  max(step)  rms(step)  s/cycle
    0      0.000080   0.000010   0.027005   0.003934   0.1
Converged!

Final summary:
    max(forces, internal): 0.000080 hartree/(bohr,rad)
    rms(forces, internal): 0.000010 hartree/(bohr,rad)
    max(forces, cartesian): 0.000063 hartree/bohr
    rms(forces, cartesian): 0.000015 hartree/bohr
    energy: -52.35197394 hartree
Wrote final, hopefully optimized, geometry to 'first_pre_final_geometry.xyz'

Preoptimization of first geometry converged!
Saved final preoptimized structure to 'first_preopt.xyz'.
RMSD with initial geometry: 0.000000 au

#####
# RUNNING LAST PREOPTIMIZATION #
#####
Spent 0.0 s preparing the first cycle.
cycle  max(force)  rms(force)  max(step)  rms(step)  s/cycle
    0      0.000188   0.000025   0.002310   0.000592   0.1
Converged!

Final summary:
    max(forces, internal): 0.000188 hartree/(bohr,rad)
    rms(forces, internal): 0.000025 hartree/(bohr,rad)
    max(forces, cartesian): 0.000182 hartree/bohr
    rms(forces, cartesian): 0.000032 hartree/bohr
    energy: -52.39960456 hartree
Wrote final, hopefully optimized, geometry to 'last_pre_final_geometry.xyz'

Preoptimization of last geometry converged!
Saved final preoptimized structure to 'last_preopt.xyz'.
RMSD with initial geometry: 0.000000 au

```

After an optimization remaining RMS and max of the forces are reported for internal and Cartesian coordinates. If the internal force is zero, but a substantial Cartesian force remains something went wrong, e.g., the generated coordinate system is lacking important coordinates. In such cases the generated coordinates can be examined manually `pysistrj [geom file] --internals` to determine important missing coordinates.

Preoptimizations are followed by the GS optimization.

```

#####
# RUNNING GROWINGSTRING #
#####
Spent 0.0 s preparing the first cycle.
cycle  max(force)  rms(force)  max(step)  rms(step)  s/cycle
    0      0.006403   0.001545   0.006403   0.001545   0.2
String=2+2 HEI=02/04 (E_max-E_0)=1.4 kJ/mol

```

(continues on next page)

(continued from previous page)

```

1      0.009367      0.001734      0.009367      0.001734      0.3
String=3+3 HEI=03/06 (E_max-E_0)=4.1 kJ/mol
2      0.010394      0.001807      0.010394      0.001807      0.6
String=4+4 HEI=04/08 (E_max-E_0)=8.0 kJ/mol
3      0.011152      0.001790      0.011152      0.001790      1.1
String=5+5 HEI=05/10 (E_max-E_0)=12.5 kJ/mol
4      0.010687      0.001722      0.010687      0.001722      1.0
String=6+6 HEI=06/12 (E_max-E_0)=17.6 kJ/mol
5      0.008751      0.001620      0.008751      0.001620      1.1
String=7+7 HEI=07/14 (E_max-E_0)=22.8 kJ/mol
6      0.008007      0.001517      0.008007      0.001517      1.3
String=8+8 HEI=08/16 (E_max-E_0)=27.6 kJ/mol
7      0.006895      0.001423      0.006895      0.001423      1.4
String=9+9 HEI=09/18 (E_max-E_0)=31.3 kJ/mol
Starting to climb in next iteration.
8      0.006253      0.001313      0.006253      0.001314      1.6
String=Full HEI=10/20 (E_max-E_0)=32.8 kJ/mol
9      0.006134      0.001169      0.0059438      0.011487      1.7
String=Full HEI=10/20 (E_max-E_0)=32.0 kJ/mol
10     0.009555      0.001504      0.022529      0.003827      1.6
String=Full HEI=10/20 (E_max-E_0)=31.6 kJ/mol
11     0.009792      0.001583      0.100000      0.011614      1.7
String=Full HEI=10/20 (E_max-E_0)=31.6 kJ/mol
12     0.008636      0.001534      0.100000      0.011518      1.7
String=Full HEI=10/20 (E_max-E_0)=30.7 kJ/mol
13     0.008708      0.001448      0.100000      0.011463      1.6
String=Full HEI=10/20 (E_max-E_0)=30.0 kJ/mol
14     0.008217      0.001363      0.100000      0.005660      1.7
String=Full HEI=10/20 (E_max-E_0)=29.4 kJ/mol
15     0.007743      0.001335      0.100000      0.005297      1.7
String=Full HEI=10/20 (E_max-E_0)=29.1 kJ/mol
16     0.007243      0.001307      0.100000      0.005143      1.6
String=Full HEI=10/20 (E_max-E_0)=28.8 kJ/mol
17     0.006897      0.001285      0.100000      0.004938      1.6
String=Full HEI=10/20 (E_max-E_0)=28.6 kJ/mol
18     0.006772      0.001277      0.100000      0.004992      1.6
String=Full HEI=10/20 (E_max-E_0)=28.5 kJ/mol
19     0.006714      0.001273      0.100000      0.005006      1.6
String=Full HEI=10/20 (E_max-E_0)=28.5 kJ/mol
Found sign 'converged'. Ending run.
Operator indicated convergence!
Wrote final, hopefully optimized, geometry to 'final_geometries.trj'
Splined HEI is at 8.07/19.00, between image 8 and 9 (0-based indexing).
Wrote splined HEI to 'splined_hei.xyz'

```

The string grows quickly and is fully grown in cycle 8. String size and barrier height between the first and HEI are reported in every cycle. From cycle 8 on, a CI is employed. The final HEI index is printed at the end. As we interpolate the HEI, the index may be a fractional number. The COS optimization is followed by a TS optimization.

```

#####
# RUNNING TS-OPTIMIZATION FROM COS #
#####

```

(continues on next page)

(continued from previous page)

Creating mixed HEI tangent, using tangents at images (8, 9).

Overlap of splined HEI tangent with these tangents:

08: 0.982763

09: 0.149474

Index of splined highest energy image (HEI) is 8.07.

Wrote animated HEI tangent to cart_hei_tangent.trj

Splined HEI (TS guess)

[xyz file printed here; removed for clarity]

Splined Cartesian HEI tangent

[xyz file printed here; removed for clarity]

Wrote splined HEI coordinates to 'splined_hei.xyz'

Calculating Hessian at splined TS guess.

Negative eigenvalues at splined HEI:

[-0.005069 -0.000317]

Overlaps between HEI tangent and imaginary modes:

00: 0.856746

01: 0.009826

Imaginary mode 0 has highest overlap with splined HEI tangent.

Spent 0.3 s preparing the first cycle.

| cycle | max(force) | rms(force) | max(step) | rms(step) | s/cycle |
|-------|------------|------------|-----------|-----------|---------|
| 0 | 0.009004 | 0.001343 | 0.056873 | 0.017930 | 0.2 |
| 1 | 0.000754 | 0.000227 | 0.158373 | 0.035866 | 0.2 |
| 2 | 0.001142 | 0.000156 | 0.316150 | 0.051562 | 3.6 |
| 3 | 0.000483 | 0.000082 | 0.339089 | 0.052646 | 0.2 |
| 4 | 0.000370 | 0.000067 | 0.321142 | 0.049527 | 0.2 |
| 5 | 0.000362 | 0.000050 | 0.243173 | 0.040395 | 3.6 |
| 6 | 0.000152 | 0.000023 | 0.174987 | 0.032910 | 0.2 |
| 7 | 0.000503 | 0.000063 | 0.092082 | 0.019454 | 0.2 |
| 8 | 0.000199 | 0.000031 | 0.121285 | 0.024502 | 3.5 |
| 9 | 0.000136 | 0.000019 | 0.061690 | 0.012096 | 0.2 |

Converged!

Final summary:

max(forces, internal): 0.000136 hartree/(bohr,rad)

rms(forces, internal): 0.000019 hartree/(bohr,rad)

max(forces, cartesian): 0.000164 hartree/bohr

rms(forces, cartesian): 0.000043 hartree/bohr

energy: -52.34823514 hartree

Wrote final, hopefully optimized, geometry to 'ts_final_geometry.xyz'

Optimized TS coords:

[xyz file printed here; removed for clarity]

Wrote TS geometry to 'ts_opt.xyz'

| HESSIAN AT FINAL GEOMETRY |

(continues on next page)

(continued from previous page)

```

... mass-weighting cartesian hessian
... doing Eckart-projection

First 10 eigenvalues [-2.4303e-03 -3.9622e-17 -8.1858e-18 -7.8671e-19  2.1865e-18  5.
↪1589e-18
  2.6247e-17  1.3733e-05  7.3736e-05  1.2735e-04]
Imaginary frequencies: [-253.24] cm1

Wrote final (not mass-weighted) hessian to 'calculated_final_cart_hessian'.
Wrote HD5 Hessian to 'final_hessian.h5'.
Wrote imaginary mode with =-253.24 cm1 to 'imaginary_mode_000.trj'

Barrier between TS and first COS image: 9.8 kJ mol1
Barrier between TS and last COS image: 134.9 kJ mol1

```

The initial HEI TS guess features only two sizable imaginary frequencies, confirming that it is a suitable TS guess. Root 0 has the highest overlap (85%) with the HEI tangent and is chosen for maximization in the TS optimization, whereas the energy will be minimized along the remaining modes. The optimization converged quickly in 10 cycles. A final Hessian is computed at the optimized TS as we used `do_hess: True`. Only one imaginary frequency remains, which is the desired result for a first-order saddle point. All sizable imaginary modes are written to `.trj` files and can be viewed by tools like `jmol`.

Barrier heights between the TS and first and last COS image (no thermochemistry is included!) are reported. In this case the energy difference between the first COS image and the TS is very small, indicating an early TS, similar to the educts. This is also confirmed by examining Fig. 4.1.

```

#####
# RUNNING IRC #
#####

Calculating energy and gradient at the TS
IRC length in mw. coords, max(|grad|) and rms(grad) in unweighted coordinates.
Norm of initial displacement step: 0.1974

```

```

#####
# IRC - FORWARD #
#####

Step   IRC length   dE / au   max(|grad|)   rms(grad)
-----
0      0.298147     -0.000333   0.001413     0.000394
Integrator indicated convergence!

```

```

#####
# IRC - BACKWARD #
#####

Step   IRC length   dE / au   max(|grad|)   rms(grad)
-----
0      0.332823     -0.000938   0.004607     0.001154
1      0.662398     -0.001367   0.006116     0.001608
2      0.991868     -0.001878   0.008670     0.002208
3      1.320005     -0.002558   0.012581     0.002968
4      1.644982     -0.003249   0.015719     0.003595

```

(continues on next page)

(continued from previous page)

```

5      1.966058      -0.003700      0.017023      0.003976
6      2.285006      -0.003972      0.016598      0.004230
7      2.596325      -0.004126      0.019112      0.004415
8      2.898296      -0.004206      0.021118      0.004439
9      3.186446      -0.004052      0.021182      0.004240
10     3.462246      -0.003697      0.019162      0.003793
11     3.723777      -0.003087      0.015042      0.003078
12     3.966373      -0.002263      0.009324      0.002161
13     4.173308      -0.001387      0.003822      0.001310
14     4.321559      -0.000781      0.002226      0.000838
15     4.420391      -0.000523      0.001616      0.000642
16     4.498118      -0.000418      0.001299      0.000548
17     4.571712      -0.000353      0.001967      0.000569
18     4.647481      -0.000298      0.004516      0.000754
19     4.722445      -0.000308      0.003469      0.000670
20     4.808189      -0.000290      0.003426      0.000682
21     4.881593      -0.000209      0.004155      0.000683

```

Integrator indicated convergence!

The imaginary mode is used to displace the TS towards educts and product. As the TS is very similar to the educt, forward IRC integration already terminates after one cycle. Maybe, further integration steps could be forced by tightening the threshold in the `irc:` block. Backward integration terminates after 22 cycles. At first, the gradient increases and after the inflection point is passed, falls off again as a stationary point is approached. In the end both IRC endpoints are fully optimized to stationary points.

```

#####
# RUNNING FORWARD_END OPTIMIZATION #
#####
Spent 0.0 s preparing the first cycle.
cycle  max(force)  rms(force)  max(step)  rms(step)  s/cycle
0      0.000491     0.000126   0.099615   0.030780   0.1
1      0.001769     0.000333   0.121277   0.033041   0.2
2      0.002986     0.000525   0.228218   0.054622   0.2
3      0.002455     0.000475   0.396872   0.096049   0.2
4      0.003106     0.000502   0.061397   0.012734   0.2
5      0.000676     0.000187   0.040565   0.007988   0.2
6      0.000498     0.000144   0.048609   0.012294   0.2
7      0.000908     0.000142   0.055895   0.016806   0.2
8      0.000715     0.000102   0.037441   0.011352   0.2
9      0.000226     0.000047   0.034467   0.006157   0.2
10     0.000119     0.000032   0.034646   0.006157   0.1
Converged!

Final summary:
    max(forces, internal): 0.000119 hartree/(bohr,rad)
    rms(forces, internal): 0.000032 hartree/(bohr,rad)
    max(forces, cartesian): 0.000241 hartree/bohr
    rms(forces, cartesian): 0.000073 hartree/bohr
    energy: -52.35194671 hartree
Wrote final, hopefully optimized, geometry to 'forward_end_final_geometry.xyz'

Moved 'forward_end_final_geometry.xyz' to 'forward_end_opt.xyz'.

```

(continues on next page)

(continued from previous page)

```
#####
# RUNNING BACKWARD_END OPTIMIZATION #
#####
Spent 0.0 s preparing the first cycle.
cycle   max(force)   rms(force)   max(step)   rms(step)   s/cycle
  0      0.004485    0.001075    0.168005    0.031009    0.1
  1      0.003805    0.000731    0.158516    0.028598    0.2
  2      0.003345    0.000610    0.047413    0.008626    0.2
  3      0.002849    0.000531    0.093820    0.017252    0.2
  4      0.001571    0.000332    0.154150    0.028740    0.2
  5      0.000752    0.000148    0.034367    0.007031    0.2
  6      0.000630    0.000107    0.017362    0.004137    0.2
  7      0.000218    0.000047    0.019096    0.003624    0.1
  8      0.000165    0.000041    0.027156    0.004814    0.2
  9      0.000400    0.000065    0.044195    0.007618    0.2
 10      0.000460    0.000076    0.039724    0.006652    0.1
 11      0.000265    0.000059    0.021454    0.003444    0.2
 12      0.000194    0.000034    0.009649    0.001594    0.1
 13      0.000146    0.000025    0.009711    0.001835    0.1
Converged!

Final summary:
    max(forces, internal): 0.000146 hartree/(bohr,rad)
    rms(forces, internal): 0.000025 hartree/(bohr,rad)
    max(forces, cartesian): 0.000183 hartree/bohr
    rms(forces, cartesian): 0.000047 hartree/bohr
    energy: -52.39959356 hartree
Wrote final, hopefully optimized, geometry to 'backward_end_final_geometry.xyz'

Moved 'backward_end_final_geometry.xyz' to 'backward_end_opt.xyz'.
```

Both optimizations converge quickly without any problems. Finally, optimized endpoint geometries are compared to the initial geometries and final barrier heights are reported. Thermochemistry is not (yet) included. Even though both endpoints are reported as dissimilar to the initial geometries they are still very similar, confirming, that the obtained TS indeed connects presumed educts and product.

```
#####
# RMSDS AFTER END OPTIMIZATIONS #
#####
start geom 0 (first_preopt.xyz/min_xtbopt.xyz)
end geom 0 ( forward_end_opt.xyz): RMSD=0.133727 au
end geom 1 (backward_end_opt.xyz): RMSD=1.126059 au
Optimized end geometries are dissimilar to 'first_preopt.xyz/min_xtbopt.xyz'!
start geom 1 (last_preopt.xyz/prod_xtbopt.xyz)
end geom 0 ( forward_end_opt.xyz): RMSD=1.144102 au
end geom 1 (backward_end_opt.xyz): RMSD=0.093044 au
Optimized end geometries are dissimilar to 'last_preopt.xyz/prod_xtbopt.xyz'!

#####
# BARRIER HEIGHTS AFTER END OPTIMIZATIONS #
#####
```

(continues on next page)

(continued from previous page)

```
Thermochemical corrections are NOT included!
```

```
Minimum energy of 0.0 kJ mol1 at 'backward_end_opt.xyz'.
```

```
    forward_end_opt.xyz: 125.10 kJ mol1  
                        TS: 134.84 kJ mol1  
    backward_end_opt.xyz: 0.00 kJ mol1
```

```
Wrote optimized end-geometries and TS to 'end_geoms_and_ts.trj'
```

Visualization/plotting of running optimizations and IRC integrations is also possible. Please see [Plotting](#) for further information.

All inputs can be found in the *examples/complex/08_trispericyclic* directory on github.

COORDINATE SYSTEMS

Choice of coordinate system is often crucial, for the successful outcome of geometry optimizations. Pysisyphus supports different coordinate systems. By default, molecular optimizations are carried out in redundant internal coordinates (RIC).

5.1 Supported Coordinate Systems

5.1.1 Cartesian Coordinates

- Strong coupling
- Unambiguously defined, if translation and rotation (TR) are removed
- Redundant set, if TR are not removed
- `type: cart`

5.1.2 Redundant Internal Coordinates

- Comprising bond stretches, (linear) bends, (improper) torsions and other primitives
- Less coupling, compared to Cartesians
- Easy estimation of diagonal model Hessians
- **Support constraints**
- Require sophisticated setup algorithm
- Iterative internal-Cartesian backtransformation, which may fail
- Usually highly redundant set
- `type: redund`

5.1.3 Delocalized Internal Coordinates (DLC)

- Complicated linear combinations of primitive internals
- No coupling, coordinates are orthogonal to each other, at least at the geometrey they were defined at
- Non redundant set
- More efficient compared to RIC for bigger systems (if initial DLC generation is feasible)
- Same comments apply, as for RICs
- `type: dlc`

5.1.4 Translation & Rotation Internal Coordinates (TRIC)

- Especially suited to optimize solvated/non-covalently bound systems
- Translation and rotation coordinates are assigned to every fragment
- Avoids error-prone assignment of interfragment coordinates
- See [10.1063/1.4952956](#) for a full discussion
- By default the B-Matrix is recalculated in every step of the internal-Cartesian backtransformation when TRIC is enabled
- `type: tric`

5.2 Supported File Formats

All formats can be read, at least to a certain extent. Coordinate files are expected to be in , if not otherwise dictated by the respective format.

| Suffix | Write | Comment |
|---------|-------|-------------------------------------|
| .xyz | ✓ | Plain XYZ, single geometry. |
| .trj | ✓ | Plain XYZ, multiple geometries. |
| .pdb | ✓ | Protein Data Bank. |
| .molden | | Restricted to [Geometries] block. |
| .zmat | | Z-Matrix, see below for an example. |
| .cjson | | As saved by Avogadro. |
| .crd | | CHARMM card format |
| .sdf | | Structure-data file |

5.2.1 Z-Matrix example

```
C
N 1 1.35
H 1 1.0 2 105.0
H 2 1.4 1 105.0 3 150.0
H 2 1.4 1 110.0 3 -160.0
```

Indexing in Z-matrices is 1-based, so the first atom has index 1. Variable substitution is not supported in the current Z-matrix parser.

5.3 YAML Input

See below for an explanation of possible inputs in the *geom* section. Input related to internal coordinates is given mostly in the *coord_kwargs* subgroup, whereas atom-related input (*isotops*, *freeze_atoms*) is given one level above. See the example below:

```
geom:
  type: cart           # Coordinate system (cart/redund/dlc/tric)
  fn: [input]         # File name or inline input
  union: False        # Define same set of primitives at multiple geometries
  isotops: null       # Specify different isotopes
  freeze_atoms: null  # Freeze Cartesians of certain atoms
  coord_kwargs:      # Keywords that are passed to the internal coordinate_
  ↪ class
    define_prims: null # Additionally define these primitives
    constrain_prims: null # Primitive internals to be constrained
    freeze_atoms_exclude: False # Whether to set up internal coordinates for frozen atoms
  preopt:
    geom:             # geom block in preopt takes same keywords as above
    ...              # no 'fn' key here!
  endopt:
    geom:             # geom block in endopt takes same keywords as above
    ...              # no 'fn' key here!
```

Employed coordinates and coordinate systems in *preopt* and *endopt* are similarly controlled by a *geom* block. Same keywords are supported, as for the *geom* block, at the top level, except the *fn* key.

5.3.1 Inline input

Inline coordinates are supported for XYZ format, and expected in `.`. Take care of proper indentation. The example below would yield RIC for the hydrogen molecule (1 bond).

```
geom:
  type: redund
  fn: |
    2

    H 0.0 0.0 0.0
    H 0.0 0.0 0.7
```

5.3.2 Types of Primitive Coordinates

Pysisyphus implements many different (primitive) internal coordinates. Every coordinate is defined by its type and a set of atom indices, e.g., 2 indices for a bond, 3 indices for a bend and 4 indices for a dihedral.

Specification of a type is necessary, as there are many different kinds of bonds, bends and dihedrals/out-of-plane. One can't just assume, that a coordinate comprised of 3 atom indices is always a regular bend, as it may also be a linear bend or a translational coordinate (TRANSLATION_X, 14), describin the mean Cartesian X coordinate of 3 atoms.

Atom indices start at 0!

```

# Primitive types
BOND = 0
AUX_BOND = 1
HYDROGEN_BOND = 2
INTERFRAG_BOND = 3
AUX_INTERFRAG_BOND = 4
BEND = 5
LINEAR_BEND = 6
LINEAR_BEND_COMPLEMENT = 7
PROPER_DIHEDRAL = 8
IMPROPER_DIHEDRAL = 9
OUT_OF_PLANE = 10
LINEAR_DISPLACEMENT = 11
LINEAR_DISPLACEMENT_COMPLEMENT = 12
# TRANSLATION = 13 # Dummy coordinate
TRANSLATION_X = 14
TRANSLATION_Y = 15
TRANSLATION_Z = 16
# ROTATION = 17 # Dummy coordinate
ROTATION_A = 18
ROTATION_B = 19
ROTATION_C = 20
# CARTESIAN = 21 # Dummy coordinate
CARTESIAN_X = 22
CARTESIAN_Y = 23
CARTESIAN_Z = 24
BONDED_FRAGMENT = 25
DUMMY_TORSION = 26
DISTANCE_FUNCTION = 27
# atan2 based coordinates
BEND2 = 28
TORSION2 = 29

```

As some of these types are quite unwieldy, several shortcuts are supported, that can be used in place of the types above.

```

# Additional shortcuts
# Using Cartesians in the framework of internal coordinates is mainly
# useful if one wants to constrain certain atoms.
"X": [PT.CARTESIAN_X],
"Y": [PT.CARTESIAN_Y],
"Z": [PT.CARTESIAN_Z],
"XY": [PT.CARTESIAN_X, PT.CARTESIAN_Y],
"XZ": [PT.CARTESIAN_X, PT.CARTESIAN_Z],
"YZ": [PT.CARTESIAN_Y, PT.CARTESIAN_Z],
"XYZ": [PT.CARTESIAN_X, PT.CARTESIAN_Y, PT.CARTESIAN_Z],
"ATOM": [PT.CARTESIAN_X, PT.CARTESIAN_Y, PT.CARTESIAN_Z],
# Primitive aliases
"B": [PT.BOND],
"A": [PT.BEND],
"A2": [PT.BEND2],
"D": [PT.PROPER_DIHEDRAL],
"DIHEDRAL": [PT.PROPER_DIHEDRAL],
"TORSION": [PT.PROPER_DIHEDRAL],

```

(continues on next page)

(continued from previous page)

```
"D2": [PT.PROPER_DIHEDRAL2],
"DIHEDRAL2": [PT.PROPER_DIHEDRAL2],
"TORSION2": [PT.PROPER_DIHEDRAL2],
# Translation & Rotation coordinates
"TRANSLATION": [PT.TRANSLATION_X, PT.TRANSLATION_Y, PT.TRANSLATION_Z],
"ROTATION": [PT.ROTATION_A, PT.ROTATION_B, PT.ROTATION_C],
# Miscellaneous
"DIST_FUNC": [PT.DISTANCE_FUNCTION]
```

5.3.3 Define Additional Primitives

Pysisyphus tries its best, to automatically come up with a reasonable set of internal coordinates, but sometimes the algorithm misses an important one. Especially at transition state guesses, where increased atom distances are common, bonds may be missed.

In such cases, additional coordinates can be requested explicitly. If additional coordinates are requested, **a nested list is expected** `[[coord0], [coord1], ...]`.

```
# General structure (list of coordinate lists)
define_prims: [[PrimType or Shortcut], [*atom indices], ...]

# Examples

# Additional bond between atoms 4 and 7 (0-based indexing).
# All three lines below result in the same bond; the latter two use shortcuts.
define_prims: [[0, 4, 7]]
define_prims: [[B, 4, 7]]
define_prims: [[BOND, 4, 7]]

# Wrong specification (forgot outer list/brackets):
define_prims: [0, 4, 7]

# Also define an additional dihedral, beside the bond
define_prims: [[0, 4, 7], ["D", 0, 1, 2, 3]]
```

5.3.4 Freeze Atoms

All three Cartesian coordinates (X, Y, Z) of certain atoms can be frozen, so they always remain at their initial value. By setting `freeze_atoms_exclude` in `coord_kwargs`, frozen atoms can be excluded from the internal coordinate setup. By default frozen atoms are included.

```
geom:
  type: [type]
  fn: [fn]
  freeze_atoms: [*atom indices]
  coord_kwargs:
    freeze_atoms_exclude: False

# Example; fully freeze Cartesians of first and second atom.
geom:
```

(continues on next page)

(continued from previous page)

```

type: cart
fn: input.xyz
freeze_atoms: [0, 1]
coord_kwargs:
  freeze_atoms: True

```

5.3.5 Constraints

Constraints beyond frozen atoms are currently only supported in conjunction with RIC (`type: redund`). It is not (yet) possible to modify the value of the specified coordinate via YAML input; the internal coordinate is constrained at its initial value. The same syntax as for `define_prims` is used. If the coordinate of the requested constraint is not already defined, it will be defined subsequently. There is no need to also add the constrained coordinate to `define_prims`.

```

# General structure (nested list of coordinates)
constrain_prims: [[[PrimType or Shortcut], *[atom indices]], ...]

# Examples

# Constrain Cartesian coordinate of atom 0.
# Both lines result in the same constraint.
constrain_prims: [[XYZ, 0]]
constrain_prims: [[ATOM, 0]]

# Constrain only Cartesian X and Y component of atom 0.
constrain_prims: [[XY, 0]]

# Constraint bond between atoms 4 and 7 (0-based indexing).
# All three lines below result in the same constraint; the latter two use shortcuts.
constrain_prims: [[0, 4, 7]]
constrain_prims: [[B, 4, 7]]
constrain_prims: [[BOND, 4, 7]]

```

Constraining the Cartesian coordinates (X, Y and Z) of one atom does not affect the final energy of an optimization. **But constraining more than one atome does.**

Harmonic restraints to selected primitive internals can be specified in the `calc:` section (see the [Restraint](#) documentation).

5.3.6 Isotopes

Different isotope masses can be requested. The system works similar to Gaussians system. A list of pairs is expected, where the first number specifies the atom and the second number is either an integer or a float. If it is an integer, the isotope mass closest to this integer is looked up in an internal database. Floats are used as is.

```

# General structure (nested list of coordinates)
isotopes: [[[atom index], [new mass, integer/float]], ...]

# Modify the mass of atom with index 2 (hydrogen in this case)
# Both lines give identical results (deuterium).
# In the second line, the mass is given directly.

```

(continues on next page)

(continued from previous page)

```
isotopes: [[2, 2]]
isotopes: [[2, 2.014101778]]
```

Different isotope masses affect calculated frequencies and IRCs. Atoms can be fixed in IRC calculations by specifying a very high mass. **YAML does not recognize 1e9 as float**, take care to add a dot (**1.e9**).

```
# Fix atom 0 in IRC calculation.
isotopes: [[0, 1.e9]]
```

5.4 Geometry & RedundantCoords

The central class in pysisyphus, handling coordinates and delegating calculations to (external QC) codes, is the *Geometry* class, similar to ASE's *Atoms*.

```
class pysisyphus.Geometry.Geometry(atoms, coords, fragments=None, coord_type='cart',
                                   coord_kwargs=None, isotopes=None, freeze_atoms=None,
                                   comment="", name="")

__init__(atoms, coords, fragments=None, coord_type='cart', coord_kwargs=None, isotopes=None,
         freeze_atoms=None, comment="", name="")
```

Object representing atoms in a coordinate system.

The Geometry represents atoms and their positions in coordinate system. By default cartesian coordinates are used, but internal coordinates are also possible.

Parameters

- **atoms** (*iterable*) -- Iterable of length N, containing element symbols.
- **coords** (*1d iterable*) -- 1d iterable of length 3N, containing the cartesian coordinates of N atoms.
- **fragments** (*dict, optional*) -- Dict with different keys denoting different fragments. The values contain lists of atom indices.
- **coord_type** (*{ "cart", "redund" }, optional*) -- Type of coordinate system to use. Right now cartesian (cart) and redundant (redund) are supported.
- **coord_kwargs** (*dict, optional*) -- Dictionary containing additional arguments that get passed to the constructor of the internal coordinate class.
- **isotopes** (*iterable of pairs, optional*) -- Iterable of pairs consisting of 0-based atom index and either an integer or a float. If an integer is given the closest isotope mass will be selected. Given a float, this float will be directly used as mass.
- **freeze_atoms** (*iterable of integers*) -- Specifies which atoms should remain fixed at their initial positions.
- **comment** (*str, optional*) -- Comment string.
- **name** (*str, optional*) -- Verbose name of the geometry, e.g. methanal or water. Used for printing

align_principal_axes()

Align the principal axes to the cartesian axes.

<https://math.stackexchange.com/questions/145023>

property all_energies

Return energies of all states that were calculated.

This will also set `self.energy`, which may NOT be the ground state, but the state corresponding to the 'root' attribute of the calculator.

approximate_radius()

Approximate molecule radius from the biggest atom distance along an axis.

as_ase_atoms()**as_g98_list()**

Returns data for fake Gaussian98 standard orientation output.

Returns

g98_list -- List with one row per atom. Every row contains [center number, atomic number, atomic type (always 0 for now), X Y Z coordinates in Angstrom.

Return type

list

as_xyz(comment="", atoms=None, cart_coords=None)

Current geometry as a string in XYZ-format.

Parameters

- **comment** (*str*, *optional*) -- Will be written in the second line (comment line) of the XYZ-string.
- **cart_coords** (*np.array*, *1d*, *shape* ($3 * \text{atoms.size}$,)) -- Cartesians for dumping instead of `self._coords`.

Returns

xyz_str -- Current geometry as string in XYZ-format.

Return type

str

assert_cart_coords(coords)**assert_compatibility(other)**

Assert that two Geometries can be subtracted from each other.

Parameters

other (*Geometry*) -- Geometry for comparison.

atom_indices()

Dict with atom types as key and corresponding indices as values.

Returns

inds_dict -- Unique atom types as keys, corresponding indices as values.

Return type

dict

property atom_types**atom_xyz_iter()****property atomic_numbers**

property `bond_sets`

`calc_double_ao_overlap(geom2)`

`calc_energy_and_forces()`

Force a calculation of the current energy and forces.

property `cart_coords`

property `cart_forces`

property `cart_gradient`

property `cart_hessian`

`center()`

property `center_of_mass`

Returns the center of mass.

Returns

R -- Center of mass.

Return type

np.array, shape(3,)

`center_of_mass_at(coords3d)`

Returns the center of mass at given coords3d.

Parameters

coords3d (np.array, shape(*N*, 3)) -- Cartesian coordinates.

Returns

R -- Center of mass.

Return type

np.array, shape(3,)

property `centroid`

Geometric center of the Geometry.

Returns

R -- Geometric center of the Geometry.

Return type

np.array, shape(3,)

`clear()`

Reset the object state.

property `comment`

```
coord_types = {'cart': None, 'cartesian': <class
'pysisyphus.intcoords.CartesianCoords.CartesianCoords'>, 'dlc': <class
'pysisyphus.intcoords.DLC.DLC'>, 'hdlc': <class 'pysisyphus.intcoords.DLC.HDLC'>,
'hredund': <class 'pysisyphus.intcoords.RedundantCoords.HybridRedundantCoords'>,
'mwcartesian': <class 'pysisyphus.intcoords.CartesianCoords.MWCartesianCoords'>,
'redund': <class 'pysisyphus.intcoords.RedundantCoords.RedundantCoords'>, 'tmtric':
<class 'pysisyphus.intcoords.RedundantCoords.TMTRIC'>, 'tric': <class
'pysisyphus.intcoords.RedundantCoords.TRIC'>}
```

property coords

1d vector of atomic coordinates.

Returns

coords -- 1d array holding the current coordinates.

Return type

np.array

property coords3d

Coordinates in 3d.

Returns

coords3d -- Coordinates of the Geometry as 2D array.

Return type

np.array

property coords_by_type

Coordinates in 3d by atom type and their corresponding indices.

Returns

- **cbt** (*dict*) -- Dictionary with the unique atom types of the Geometry as keys. It's values are the 3d coordinates of the corresponding atom type.
- **inds** (*dict*) -- Dictionary with the unique atom types of the Geometry as keys. It's values are the original indices of the 3d coordinates in the whole coords3d array.

copy(*coord_type=None, coord_kwargs=None*)

Returns a new Geometry object with same atoms and coordinates.

Parameters

- **coord_type** (*str*) -- Desired coord_type, defaults to current coord_type.
- **coord_kwargs** (*dict, optional*) -- Any desired coord_kwargs that will be passed to the RedundantCoords object.

Returns

geom -- New Geometry object with the same atoms and coordinates.

Return type

Geometry

copy_all(*coord_type=None, coord_kwargs=None*)

property covalent_radii

del_atoms(*inds, **kwargs*)

describe()

dump_xyz(*fn, cart_coords=None, **kwargs*)

eckart_projection(*mw_hessian, return_P=False, full=False*)

property energy

Energy of the current atomic configuration.

Returns

energy -- Energy of the current atomic configuration.

Return type

float

fd_coords3d_gen(*step_size=0.001*)

Iterator returning 3d Cartesians for finite-differences.

property forces

Energy of the current atomic configuration.

Returns**force** -- 1d array containing the forces acting on the atoms. Negative of the gradient.**Return type**

np.array

get_energy_and_cart_forces_at(*cart_coords*)**get_energy_and_cart_hessian_at**(*cart_coords*)**get_energy_and_forces_at**(*coords*)

Calculate forces and energies at the given coordinates.

The results are not saved in the Geometry object.

get_energy_at(*coords*)**get_energy_at_cart_coords**(*cart_coords*)**get_fragments**(*regex*)**get_imag_frequencies**(*hessian=None, thresh=1e-06*)**get_normal_modes**(*cart_hessian=None, full=False*)

Normal mode wavenumbers, eigenvalues and Cartesian displacements Hessian.

get_restart_info()**get_sphere_radius**(*offset=4*)**get_subgeom**(*indices, coord_type='cart', sort=False*)

Return a Geometry containing a subset of the current Geometry.

Parameters

- **indices** (*iterable of ints*) -- Atomic indices that the define the subset of the current Geometry.
- **coord_type** (*str, ("cart", "redund"), optional*) -- Coordinate system of the new Geometry.

Returns**sub_geom** -- Subset of the current Geometry.**Return type***Geometry***get_subgeom_without**(*indices, **kwargs*)**get_temporary_coords**(*coords*)**get_thermoanalysis**(*energy=None, cart_hessian=None, T=298.15, p=101325, point_group='c1'*)

get_trans_rot_projector(*full=False*)

property gradient

Negative of the force.

Returns

gradient -- 1d array containing the negative of the current forces.

Return type

np.array

property hessian

Matrix of second derivatives of the energy in respect to atomic displacements.

Returns

hessian -- 2d array containing the second derivatives of the energy with respect to atomic/coordinate displacements depending on the type of coordiante system.

Return type

np.array

property inertia_tensor

property is_analytical_2d

jmol(*atoms=None, cart_coords=None*)

Show geometry in jmol.

property layers

mass_weigh_hessian(*hessian*)

property masses

property masses_rep

property mm_inv

Inverted mass matrix.

Returns a diagonal matrix containing the inverted atomic masses.

property mm_sqrt_inv

Inverted square root of the mass matrix.

modes3d()

property moving_atoms

moving_atoms_jmol()

property mw_coords

Mass-weighted coordinates.

Returns

mw_coords -- 1d array containing the mass-weighted cartesian coordiantes.

Return type

np.array

property mw_gradient

Mass-weighted gradient.

Returns

mw_gradient -- Returns the mass-weighted gradient.

Return type

np.array

property mw_hessian

Mass-weighted hessian.

Returns

mw_hessian -- 2d array containing the mass-weighted hessian $M^{(-1/2)} H M^{(-1/2)}$.

Return type

np.array

principal_axes_are_aligned()

Check if the principal axes are aligned with the cartesian axes.

Returns

aligned -- Whether the principal axes are aligned or not.

Return type

bool

reparametrize()**reset_coords**(*new_typed_prims=None*)**rmsd**(*geom*)**rotate**(*copy=False, rng=None*)**set_calculator**(*calculator, clear=True*)

Reset the object and set a calculator.

set_coord(*ind, coord*)

Set a coordinate by index.

Parameters

- **ind** (*int*) -- Index in of the coordinate to set in the self.coords array.
- **coord** (*float*) -- Coordinate value.

set_coords(*coords, cartesian=False, update_constraints=False*)**set_h5_hessian**(*fn*)**set_restart_info**(*restart_info*)**set_results**(*results*)

Save the results from a dictionary.

Parameters

results (*dict*) -- The keys in this dict will be set as attributes in the current object, with the corresponding item as value.

standard_orientation()

property `sum_formula`

tmp_xyz_handle(*atoms=None, cart_coords=None*)

property `total_mass`

unweight_mw_hessian(*mw_hessian*)

Unweight a mass-weighted hessian.

Parameters

mw_hessian (*np.array*) -- Mass-weighted hessian to be unweighted.

Returns

hessian -- 2d array containing the hessian.

Return type

np.array

property `vdw_radii`

vdw_volume(***kwargs*)

without_hydrogens()

zero_frozen_forces(*cart_forces*)

`pysisyphus.Geometry.get_trans_rot_projector`(*cart_coords, masses, full=False*)

`pysisyphus.Geometry.get_trans_rot_vectors`(*cart_coords, masses, rot_thresh=1e-06*)

Vectors describing translation and rotation.

These vectors are used for the Eckart projection by constructing a projector from them.

See Martin J. Field - A Partial Introduction to the simulation of Molecular Systems, 2007, Cambridge University Press, Eq. (8.23), (8.24) and (8.26) for the actual projection.

See also <https://chemistry.stackexchange.com/a/74923>.

Parameters

- **cart_coords** (*np.array, 1d, shape (3 * atoms.size,)*) -- Atomic masses in amu.
- **masses** (*iterable, 1d, shape (atoms.size,)*) -- Atomic masses in amu.

Returns

ortho_vecs -- 2d array containing row vectors describing translations and rotations.

Return type

*np.array(6, 3*atoms.size)*

`pysisyphus.Geometry.inertia_tensor`(*coords3d, masses*)

Inertia tensor.

$x^2 \quad xy \quad xz \mid$

$(x \ y \ z)^T \cdot (x \ y \ z) = \mid xy \ y^2 \ yz \mid$

$xz \ yz \ z^2 \mid$

Basic infrastructure to work with internal coordinates is provided by *RedundantCoords*.

```

class pysisyphus.intcoords.RedundantCoords.HybridRedundantCoords(*args, **kwargs)

class pysisyphus.intcoords.RedundantCoords.RedundantCoords(atoms, coords3d, masses=None,
                                                             bond_factor=1.3, typed_prims=None,
                                                             define_prims=None,
                                                             constrain_prims=None,
                                                             freeze_atoms=None,
                                                             freeze_atoms_exclude=False,
                                                             internals_with_frozen=False,
                                                             define_for=None, bonds_only=False,
                                                             check_bends=True, rebuild=True,
                                                             bend_min_deg=15,
                                                             dihed_max_deg=175,
                                                             lb_min_deg=175, weighted=False,
                                                             min_weight=0.3,
                                                             svd_inv_thresh=0.000316,
                                                             recalc_B=False, tric=False,
                                                             hybrid=False, hbond_angles=False,
                                                             rm_for_frag=None)

    property B
        Wilson B-Matrix

    property B_inv
        Generalized inverse of the Wilson B-Matrix.

    property B_inv_prim
        Generalized inverse of the primitive Wilson B-Matrix.

    property B_prim
        Wilson B-Matrix

    property Bt_inv
        Transposed generalized inverse of the Wilson B-Matrix.

    property Bt_inv_prim
        Transposed generalized inverse of the primitive Wilson B-Matrix.

    property C
        Diagonal matrix. Entries for constraints are set to one.

    property P
        Projection matrix onto B. See [1] Eq. (4).

    backtransform_hessian(redund_hessian, int_gradient=None)
        Transform Hessian in internal coordinates to Cartesians.

    property bend_atom_indices

    property bend_indices

    property bond_atom_indices

    property bond_indices

    property bond_typed_prims

```

property cartesian_indices
clear()
property constrained_indices
property coords
property coords3d
property dihedral_atom_indices
property dihedral_indices
eval(*coords3d*, *attr=None*)
get_K_matrix(*int_gradient=None*)
get_index_of_typed_prim(*typed_prim*)
Index in self.typed_prims for the supplied typed_prim.
get_prim_internals_by_indices(*indices*)
inv_B(*B*)
inv_Bt(*B*)
property linear_bend_indices
log(*message*)
log_int_grad_msg(*int_gradient*)
property outofplane_indices
property prim_coords
property prim_indices_set
property prim_internals
property primitives
print_typed_prims()
project_hessian(*H*, *shift=1000*)
Expects a hessian in internal coordinates. See Eq. (11) in [1].
project_vector(*vector*)
Project supplied vector onto range of B.
return_inds(*slice_*)
property rotation_indices
set_inds_from_typed_prims(*typed_prims*)
set_primitive_indices(*atoms*, *coords3d*)
transform_forces(*cart_forces*)
Combination of Eq. (9) and (11) in [1].

transform_hessian(*cart_hessian*, *int_gradient=None*)

Transform Cartesian Hessian to internal coordinates.

transform_int_step(*int_step*, *update_constraints=False*, *pure=False*)

property translation_indices

property typed_prims

class pysisyphus.intcoords.RedundantCoords.**TMTRIC**(*atoms*, **args*, ***kwargs*)

class pysisyphus.intcoords.RedundantCoords.**TRIC**(**args*, ***kwargs*)

5.5 Related Literature

1. The efficient optimization of molecular geometries using redundant internal coordinates
2. The generation and use of delocalized internal coordinates in geometry optimization
3. Geometry optimization in redundant internal coordinates
4. Geometry optimization made simple with translation and rotation coordinates

CALCULATORS

Exploring potential energy surfaces (PESs) requires calculation of energies and its derivatives (gradients, Hessian matrices).

For testing purposes and method development, pysisyphus implements various analytical 2d potentials, as they allow fast evaluations of the aforementioned quantities. Actual (production) calculations are carried out by wrapping existing quantum chemistry codes (ORCA, TURBOMOLE, Gaussian etc.) and delegating the required calculations to them. Pysisyphus generates all necessary inputs, executes the QC code and parses their output for the requested quantities.

Furthermore pysisyphus provides several "meta"-calculators which wrap other (actual) calculators, to modify calculated energies and forces. Examples for this are the *Dimer* calculator, used for carrying out transition state searches or the *ONIOM* calculator, allowing multi-level calculations comprising different levels of theory.

External forces, e.g. a restraining spherical potential or harmonic restraints on primitive internal coordinates (stretches, bends, torsion) can be applied with *ExternalPotential*.

6.1 YAML input

Possible keywords for the YAML input can be derived from inspecting the possible arguments of the *Calculator* baseclass (see below) and the possible arguments of the respective calculator, e.g. ORCA or XTB.

The most commonly used keywords, derived from the *Calculator* baseclass are *mem*, handling the requested memory per core in MB, *pal*, handling the number of requested CPU cores, *charge*, the total charge of the system and *mult*, the systems multiplicity.

For (excited state, ES) calculations carried out with calculators derived from *OverlapCalculator* additional keywords are possible. The most common keywords controlling ES calculations are *track*, activating ES tracking, *ovlp_type*, selecting the tracking algorithm and *ovlp_with*, handling the selection of the reference cycle.

An example input highlighting the most important keywords is shown below.

```
geom:
  [... omitted ...]
calc:
  type: orca                # Calculator type, e.g. g09/g16/openmolcas/
                           # orca/pyscf/turbomole/dftb+/mopac/psi4/xtb

  pal: 1                    # Number of CPU cores
  mem: 1000                 # Memory per core
  charge: 0                 # Charge
  mult: 1                   # Multiplicity
  # Keywords for ES calculations
  track: False              # Activate ES tracking
```

(continues on next page)

(continued from previous page)

```

ovlp_type: tden                # Tracking algorithm
ovlp_with: previous            # Reference cycle selection
# Additional calculator specific keywords
[... omitted ...]
opt:
[... omitted ...]

```

6.2 Calculator base classes

```

class pysisyphus.calculators.Calculator.Calculator(calc_number=0, charge=0, mult=1,
                                                       base_name='calculator', pal=1, mem=1000,
                                                       keep_kind='all', check_mem=True, retry_calc=0,
                                                       last_calc_cycle=None, clean_after=True,
                                                       out_dir='qm_calcs', force_num_hess=False,
                                                       num_hess_kwargs=None)

__init__(calc_number=0, charge=0, mult=1, base_name='calculator', pal=1, mem=1000, keep_kind='all',
          check_mem=True, retry_calc=0, last_calc_cycle=None, clean_after=True, out_dir='qm_calcs',
          force_num_hess=False, num_hess_kwargs=None)

```

Base-class of all calculators.

Meant to be extended.

Parameters

- **calc_number** (*int*, *default=0*) -- Identifier of the Calculator. Used in distinguishing it from other Calculators, e.g. in ChainOfStates calculations. Also used in the creation of filenames.
- **charge** (*int*, *default=0*) -- Molecular charge.
- **mult** (*int*, *default=1*) -- Molecular multiplicity (1 = singlet, 2 = doublet, ...)
- **base_name** (*str*, *default=calculator*) -- Generated filenames will start with this string.
- **pal** (*int*, *default=1*) -- Positive integer that gives the number of physical cores to use on 1 node.
- **mem** (*int*, *default=1000*) -- Memory per core in MB. The total amount of memory is given as `mem*pal`.
- **check_mem** (*bool*, *default=True*) -- Whether to adjust the requested memory if too much is requested.
- **retry_calc** (*int*, *default=0*) -- Number of additional retries when calculation failed.
- **last_calc_cycle** (*int*) -- Internal variable used in restarts.
- **clean_after** (*bool*) -- Delete temporary directory were calculations were executed after a calculation.
- **out_dir** (*str*) -- Path that is prepended to generated filenames.
- **force_hess_kwargs** (*bool*, *default False*) -- Force numerical Hessians.
- **num_hess_kwargs** (*dict*) -- Keyword arguments for finite difference Hessian calculation.

apply_keep_kind()

apply_set_plans(*kept_fns*, *set_plans=None*)

build_set_plans(*_set_plans=None*)

clean(*path*)

Delete the temporary directory.

Parameters

path (*Path*) -- Directory to delete.

conf_key = *None*

force_num_hessian()

Always calculate numerical Hessians.

classmethod geom_from_fn(*fn*, ***kwargs*)

get_cmd(*key='cmd'*)

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

get_num_hessian(*atoms*, *coords*, ***prepare_kwargs*)

get_restart_info()

Return a dict containing chkfiles.

Returns

restart_info -- Dictionary holding the calculator state. Used for restoring calculators in restarted calculations.

Return type

dict

keep(*path*)

Backup calculation results.

Parameters

path (*Path*) -- Temporary directory of the calculation.

Returns

kept_fns -- Dictionary holding the filenames that were backed up. The keys correspond to the type of file.

Return type

dict

log(*message=""*)

Write a log message.

Wraps the logger variable.

Parameters

message (*str*) -- Message to be logged.

make_fn(*name*, *counter=None*, *return_str=False*)

Make a full filename.

Return a full filename including the calculator name and the current counter given a suffix.

Parameters

- **name** (*str*) -- Suffix of the filename.
- **counter** (*int*, *optional*) -- If not given use the current `calc_counter`.
- **return_str** (*int*, *optional*) -- Return a string instead of a Path when True.

Returns

fn -- Filename.

Return type

`str`

property name

popen(*cmd*, *cwd=None*)

prepare(*inp*)

Prepare a temporary directory and write input.

Similar to `prepare_path`, but the input is also written into the prepared directory.

6.2.1 Paramters

inp

[*str*] Input to be written into the file `self.inp_fn` in the prepared directory.

returns**path: Path**

Prepared directory.

prepare_coords(*atoms*, *coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

`np.array`, 3d

prepare_input(*atoms*, *coords*, *calc_type*)

Meant to be extended.

prepare_path(*use_in_run=False*)

Get a temporary directory handle.

Create a temporary directory that can later be used in a calculation.

Parameters

use_in_run (*bool, option*) -- Sets the internal variable `self.path_already_prepared` that is later read by `self.run()`. No new temporary directory will be created in `self.run()`.

Returns

path: `Path`

Prepared directory.

prepare_pattern(*raw_pat*)

Prepare globs.

Transforms an entry of `self.to_keep` into a glob and a key suitable for the use in `self.keep()`.

Parameters

raw_pat (*str*) -- Entry of `self.to_keep`

Returns

- **pattern** (*str*) -- Glob that can be used in `Path.glob()`
- **multi** (*bool*) -- Flag if glob may match multiple files.
- **key** (*str*) -- A key to be used in the `kept_fns` dict.

prepare_turbo_coords(*atoms, coords*)

Get a Turbomole coords string.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- String holding coordinates in Turbomole coords format.

Return type

`str`

prepare_xyz_string(*atoms, coords*)

Returns a xyz string in Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

xyz_str -- Coordinates in .xyz format.

Return type

`string`

print_capabilities()

print_out_fn(*path*)

Print calculation output.

Prints the output of a calculator after a calculation.

Parameters

path (*Path*) -- Temporary directory of the calculation.

restore_org_hessian()

Restore original 'get_hessian' method, which may also fallback to numerical Hessians, if not implemented.

run(*inp, calc, add_args=None, env=None, shell=False, hold=False, keep=True, cmd=None, inc_counter=True, run_after=True, parser_kwargs=None, symlink=True*)

Run a calculation.

The bread-and-butter method to actually run an external quantum chemistry code.

Parameters

- **inp** (*str*) -- Input for the external program that is written to the temp-dir.
- **calc** (*str, hashable*) -- Key (and more or less type of calculation) to select the right parsing function from `self.parser_funcs`.
- **add_args** (*iterable, optional*) -- Additional arguments that will be appended to the program call.
- **env** (*Environment, optional*) -- A potentially modified environment for the subprocess call.
- **shell** (*bool, optional*) -- Use a shell to execute the program call. Need for Turbomole were we chain program calls like dscf; escf.
- **hold** (*bool, optional*) -- Wether to remove the temporary directory after the calculation.
- **keep** (*bool, optional*) -- Wether to backup files as specified in `self.to_keep()`. Usually you want this.
- **cmd** (*str or iterable, optional*) -- Overwrites `self.base_cmd`.
- **inc_counter** (*bool, optional*) -- Wether to increment the counter after a calculation.

Returns

results -- Dictionary holding all applicable results of the calculations like the energy, a forces vector and/or excited state energies from TDDFT.

Return type

dict

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like formchk or ricctools.

set_restart_info(*restart_info*)

Sets restart information (chkfiles etc.) on the calculator.

Parameters

restart_info (*dict*) -- Dictionary holding the calculator state. Used for restoring calculators in restarted calculations.

verify_chkfiles(*chkfiles*)

Checks if given chkfiles exist and return them as Paths

Parameters

chkfiles (*dict*) -- Dictionary holding the chkfiles. The keys correspond to the attribute names, the values are strs holding the (potentially full) filename (path).

Returns

paths -- Dictionary of Paths.

Return type

dict

class pysisyphus.calculators.Calculator.**HessKind**(*value*)

An enumeration.

NUMERICAL = 2

ORG = 1

class pysisyphus.calculators.Calculator.**KeepKind**(*value*)

An enumeration.

ALL = 1

LATEST = 2

NONE = 3

class pysisyphus.calculators.Calculator.**SetPlan**(*key*, *name=None*, *condition=<function SetPlan.<lambda>>*, *fail=None*)

condition()

fail: Optional[Callable] = None

key: str

name: Optional[str] = None

6.3 OverlapCalculator base class

class pysisyphus.calculators.OverlapCalculator.**NTOs**(*ntos*, *lambdas*)

Bases: tuple

lambdas

Alias for field number 1

ntos

Alias for field number 0

```
class pysisyphus.calculators.OverlapCalculator.OverlapCalculator(*args, track=False,
                                                                ovlp_type='tden',
                                                                double_mol=False,
                                                                ovlp_with='previous',
                                                                adapt_args=(0.5, 0.3, 0.6),
                                                                cdds=None, orient="",
                                                                dump_fn='overlap_data.h5',
                                                                h5_dump=False,
                                                                conf_thresh=0.001,
                                                                mos_ref='cur',
                                                                mos_renorm=True, **kwargs)
```

Bases: [Calculator](#)

```
H5_MAP = {'Ca': 'Ca_list', 'Cb': 'Cb_list', 'Xa': 'Xa_list', 'Xb': 'Xb_list',
          'Ya': 'Ya_list', 'Yb': 'Yb_list', 'all_energies': 'all_energies_list', 'coords':
          'coords_list', 'ref_roots': 'reference_roots', 'roots': 'roots_list'}
```

```
OVLP_TYPE_VERBOSE = {'nto': 'natural transition orbital overlap', 'nto_org':
                     'original natural transition orbital overlap', 'tden': 'transition density matrix
                     overlap', 'top': 'transition orbital pair overlap', 'wf': 'wavefunction overlap'}
```

```
VALID_CDDS = (None, 'calc', 'render')
```

```
VALID_KEYS = ['wf', 'tden', 'nto', 'nto_org', 'top']
```

```
VALID_XY = ('X', 'X+Y', 'X-Y')
```

```
calc_cdd_cube(root, cycle=-1)
```

conf_thresh

```
self.dyn_roots = int(dyn_roots) if self.dyn_roots != 0:
```

```
    self.dyn_roots = 0 self.log("dyn_roots = 0 is hardcoded right now")
```

property data_model

dump_overlap_data()

static from_overlap_data(h5_fn, set_wfow=False)

get_ci_coeffs_for(ind)

get_h5_group()

get_indices(indices=None)

A new root is determined by selecting the overlap matrix row corresponding to the reference root and checking for the root with the highest overlap (at the current geometry).

The overlap matrix is usually formed by a double loop like:

```
overlap_matrix = np.empty((ref_states, cur_states)) for i, ref_state in enumerate(ref_states):
```

```
    for j, cur_state in enumerate(cur_states):
```

```
        overlap_matrix[i, j] = make_overlap(ref_state, cur_state)
```

So the reference states run along the rows. Thats why the ref_state index comes first in the 'indices' tuple.

static get_mo_norms(C, S_AO)

get_orbital_matrices(*indices=None, S_AO=None*)

Return MO coefficients and AO overlaps for the given indices.

If not provided, a AO overlap matrix is constructed from one of the MO coefficient matrices (controlled by `self.mos_ref`). Also, if requested one of the two MO coefficient matrices is re-normalized.

get_ref_mos(*C_ref, C_cur*)

static get_sao_from_mo_coeffs(*C*)

Recover AO overlaps from given MO coefficients.

For MOs in the columns of `mo_coeffs`:

$$S_{AO} = C^{\dagger} C S_{AO} C = C^{\dagger} (S_{AO} C)^T = C^{\dagger} C^T S_{AO}^T = C^{\dagger} C^T S_{AO} C = I$$

get_tden_overlaps(*indices=None, S_AO=None*)

get_top_differences(*indices=None, S_AO=None*)

Transition orbital projection.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like `mos`, a MO coefficient array and a CI coefficient array.

render_cdd_cube()

static renorm_mos(*C, S_AO*)

store_overlap_data(*atoms, coords, path=None, overlap_data=None*)

property stored_calculations

track_root(*ovlp_type=None*)

Check if a root flip occurred compared to the previous cycle by calculating the overlap matrix wrt. a reference cycle.

`pysisyphus.calculators.OverlapCalculator.get_data_model(exc_state_num, occ_a, virt_a, occ_b, virt_b, ovlp_type, atoms, max_cycles)`

6.4 Calculators with Excited state capabilities

6.4.1 Gaussian09

class `pysisyphus.calculators.Gaussian09.Gaussian09`(**args, **kwargs*)

`conf_key = 'gaussian09'`

6.4.2 Gaussian16

```
class pysisyphus.calculators.Gaussian16.Gaussian16(route, gbs="", gen="", keep_chk=False, stable="",  
                                                fchk=None, **kwargs)
```

Bases: [OverlapCalculator](#)

conf_key = 'gaussian16'

get_chkfiles()

get_energy(atoms, coords, **prepare_kwargs)

Meant to be extended.

get_forces(atoms, coords, **prepare_kwargs)

Meant to be extended.

get_hessian(atoms, coords, **prepare_kwargs)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

make_exc_str()

make_fchk(path)

make_gbs_str()

parse_635r_dump(dump_path, roots, nmos)

parse_all_energies(fchk=None)

parse_charges(path=None)

parse_double_mol(path, out_fn=None)

static parse_fchk(fchk_path, keys)

parse_force(path)

parse_hessian(path)

parse_keyword(text)

parse_log(*args, **kwargs)

parse_stable(path)

parse_tddft(path)

prepare_input(atoms, coords, calc_type, did_stable=False, point_charges=None)

Meant to be extended.

prepare_overlap_data(path)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reuse_data(path)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_double_mol_calculation(*atoms*, *coords1*, *coords2*)

run_rwf_dump(*path*, *rwf_index*, *chk_path=None*)

run_stable(*atoms*, *coords*, ***prepare_kwargs*)

set_chkfiles(*chkfiles*)

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

6.4.3 OpenMolcas

Pysisyphus currently supports energy and gradient calculations utilizing the `&rasscf` and/or the `&mcpdft` sections. Neither analytical nor numerical Hessians are yet implemented for the OpenMolcas-calculator.

Two keywords are always required: `inorb` and `basis`, with the former pointing to a `.RasOrb` file and the latter containing the selected atomic orbital basis, e.g., `ano-rcc-vdzp`. Additional input for the `&gateway`, `&rasscf` and `&mcpdft` sections can be given via the respective keyword(s).

Due to restrictions of the current design, simple keywords that don't take further arguments as `cmsi` in `&rasscf` or `grad` and `mspdft` in `&mcpdft` still must be given with a trailing colon. See below for an example.

Listing 6.1: Optimization using compressed-multi-state PDFT.

```
geom:
  type: redund
  fn: |
    4

    C      -1.0398336639      0.0      0.0
    S      0.6002429216      0.0      0.0
    H -1.6321592382      -0.94139864      0.0
    H      -1.6321592382      0.94139864      0.0
calc:
  type: openmolcas
  basis: cc-pvdz
  charge: 0
  mult: 1
  inorb: /home/johannes/tmp/359_cmcpdft/359_cmcpdft.RasOrb
  rasscf:
    ciroot: 3 3 1
    mdrlxroot: 2
  cmsi:
  mcpdft:
    ksdf: t:pbe
  grad:
  mspdf:
```

(continues on next page)

(continued from previous page)

```
opt:  
  thresh: gau
```

```
class pysisyphus.calculators.OpenMolcas.OpenMolcas(basis, inporb, rasscf=None, gateway=None,  
                                                    mcpdft=None, track=True, **kwargs)
```

Bases: [Calculator](#)

build_gateway_str()

build_mcpdft_str()

build_rasscf_str()

build_rassi_str()

build_str_from_dict(*dct*)

conf_key = 'openmolcas'

get_energy(*atoms, coords*)

Meant to be extended.

get_forces(*atoms, coords*)

Meant to be extended.

get_pal_env()

get_root()

parse_energies(*text*)

parse_energy(*path*)

parse_gradient(*path*)

parse_rassi_track(*path*)

prepare_coords(*atoms, coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms, coords, calc_type*)

Meant to be extended.

reattach(*last_calc_cycle*)

run_calculation(*atoms, coords, calc_type='energy'*)

6.4.4 ORCA 4.2.1 / 5.0.1

```
class pysisyphus.calculators.ORCA.ORCA(keywords, blocks="", gbw=None, do_stable=False,
                                       numfreq=False, json_dump=True, **kwargs)
```

Bases: [OverlapCalculator](#)

```
__init__(keywords, blocks="", gbw=None, do_stable=False, numfreq=False, json_dump=True, **kwargs)
```

ORCA calculator.

Wrapper for creating ORCA input files for energy, gradient and Hessian calculations. The PAL and memory inputs must not be given in the keywords and/or blocks, as they are handled by the 'pal' and 'memory' arguments.

Parameters

- **keywords** (*str*) -- Keyword line, as normally given in ORCA, excluding the leading "!".
- **blocks** (*str*, *optional*) -- ORCA block input(s), e.g. for TD-DFT calculations (%tddft ... end). As the blocks start with a leading "%", wrapping the input in quotes (") is required, otherwise the parsing will fail.
- **gbw** (*str*, *optional*) -- Path to an input gbw file, which will be used as initial guess for the first calculation. Will be overridden later, with the path to the gbw file of a previous calculation.
- **do_stable** (*bool*, *optional*) -- Run stability analysis until a stable wavefunction is obtained, before every calculation.
- **numfreq** (*bool*, *optional*) -- Use numerical frequencies instead of analytical ones.
- **json_dump** (*bool*, *optional*) -- Whether to dump the wavefunction to JSON via orca_2json. The JSON can become very large in calculations comprising many basis functions.

```
check_termination(*args, **kwargs)
```

```
clean_tmp(path)
```

```
conf_key = 'orca'
```

```
get_block_str()
```

```
get_chkfiles()
```

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
get_moinp_str(gbw)
```

```
get_stable_wavefunction(atoms, coords)
```

```
parse_all_energies(text=None, triplets=None)
```

static parse_atoms_coords(*inp*, *args, **kwargs)

static parse_cis(*cis*)

Simple wrapper of external function.

Currently, only returns X and Y.

parse_energy(*path*)

parse_engrad(*path*)

static parse_engrad_info(*inp*, *args, **kwargs)

static parse_gbw(*gbw_fn*)

static parse_hess_file(*inp*, *args, **kwargs)

parse_hessian(*path*)

parse_mo_numbers(*out_fn*)

parse_stable(*path*)

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges*=None, *do_stable*=False)

Meant to be extended.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reattach(*last_calc_cycle*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

Basically some kind of dummy method that can be called to execute ORCA with the stored cmd of this calculator.

set_chkfiles(*chkfiles*)

set_mo_coeffs(*mo_coeffs*=None, *gbw*=None)

static set_mo_coeffs_in_gbw(*in_gbw_fn*, *out_gbw_fn*, *mo_coeffs*)

See `self.parse_gbw`.

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

`pysisyphus.calculators.ORCA.get_exc_ens_fosc`(*wf_fn*, *cis_fn*, *log_fn*)

`pysisyphus.calculators.ORCA.get_name`(*text*)

Return string that comes before first character & offset.

`pysisyphus.calculators.ORCA.make_sym_mat`(*table_block*)

`pysisyphus.calculators.ORCA.parse_orca_cis(cis_fn)`

Read binary CI vector file from ORCA.

Loosly based on TheoDORE 1.7.1, Authors: S. Mai, F. Plasser <https://sourceforge.net/p/theodore-qc>

`pysisyphus.calculators.ORCA.parse_orca_gbw(gbw_fn)`

Adapted from <https://orcaforum.kofo.mpg.de/viewtopic.php?f=8&t=3299>

The first 5 long int values represent pointers into the file:

Pointer @+0: Internal ORCA data structures Pointer @+8: Geometry Pointer @+16: BasisSet Pointer @+24: Orbitals Pointer @+32: ECP data

`pysisyphus.calculators.ORCA.save_orca_pc_file(point_charges, pc_fn, hardness=None)`

6.4.5 PySCF 1.7.6

6.4.6 Turbomole 7.x

Pysisyphus does not implement a wrapper for *define*, so the user has to manually prepare a directory containing a valid control file. An automated *define* wrapper, restricted to ground state functionality, is available via the [QCEngine project](#), to which I contributed the Turbomole harness.

Care should be taken to include only the minimum amount of necessary files in the *control_path* directory, e.g., (*auxbasis*, *basis*, *control*, *coord*, *mos*) for a closed-shell calculation using RI. **A gradient file must not be present in control_path, as well as other subdirectories and files with .out extension.** The *coord* file, while not strictly required, should be kept too, to facilitate testing of the setup with standalone Turbomole.

It may be a good idea to pre-converge the calculation in *control_path*, to see if the setup is correct and actually works. Resulting files like *energy*, *statistics* can be deleted; *mos* should be kept, as the converged MOs are reused in pysisyphus.

If an excited-state optimization is desired, care has to be taken, to include `$exopt [n]` for TD-DFT/TDA or the `geoopt state=([n])` (ricc2)! Tracking of excited states is currently possible for closed shell *egrad* and *ricc2* calculations.

The current implementation was tested against Turbomole 7.4.1 and QCEngine 0.19.0. Please see [examples/complex/11_turbomole_gs_tsopt](#) for a full example where Turbomole is utilized in a growing string calculation. The same example, using QCEngine, is found in [examples/complex/12_qcengine_turbomole_gs_tsopt](#). MOs are not reused with the QCEngine calculator, so the native pysisyphus calculator is probably faster.

```
class pysisyphus.calculators.Turbomole.Turbomole(control_path=None, simple_input=None,
                                                root=None, double_mol_path=None,
                                                cosmo_kwargs=None, **kwargs)
```

Bases: [OverlapCalculator](#)

```
append_control(to_append, log_msg="", **kwargs)
```

```
conf_key = 'turbomole'
```

```
get_chkfiles()
```

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, cmd=None, **prepare_kwargs)
```

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

get_pal_env()

get_ricc2_root(*text*)

static make_molden(*path*)

parse_all_energies()

parse_cc2_vectors(*ccre*)

parse_ci_coeffs()

parse_double_mol(*path*)

Parse a double molecule overlap matrix from Turbomole output to be used with WFOWrapper.

parse_energy(*path*)

parse_force(*path*)

parse_gs_energy()

Several places are possible: \$subenergy from control file total energy from turbomole.out Final MP2 energy from turbomole.out with ADC(2) Final CC2 energy from turbomole.out with CC(2)

parse_hessian(*path*, *fn=None*)

parse_mos()

parse_td_vectors(*text*)

For TDA calculations only the X vector is present in the *ciss_a/etc.* file. In TDDFT calculations there are twice as much items compared with TDA. The first half corresponds to (X+Y) and the second half to (X-Y). X can be calculated as $X = ((X+Y)+(X-Y))/2$. Y is then given as $Y = (X+Y)-X$. The normalization can then be checked as

$$\text{np.concatenate}((X, Y)).\text{dot}(\text{np.concatenate}((X, -Y)))$$

and should be 1.

static parse_tddft_tden(*inp*, **args*, ***kwargs*)

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges=None*)

To rectify this we have to construct the basemcd dynamically and construct it ad hoc. We could set a RI flag in the beginning and select the correct scf binary here from it. Then we select the following binary on demand, e.g. aoforce or rdgrad or egrad etc.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

prepare_point_charges(*point_charges*)

\$point_charges <x> <y> <z> <q>

prepare_td(*text*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`.
Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_double_mol_calculation(*atoms*, *coords1*, *coords2*)

set_chkfiles(*chkfiles*)

set_occ_and_mo_nums(*text*)

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

sub_control(*pattern*, *repl*, *log_msg*="", ***kwargs*)

`pysisyphus.calculators.Turbomole.control_from_simple_input`(*simple_inp*, *charge*, *mult*,
cosmo_kwargs=None)

Create control file from 'simple input'.

See examples/opt/26_turbomole_simple_input/ for an example.

`pysisyphus.calculators.Turbomole.get_cosmo_data_groups`(*atoms*, *epsilon*, *rsolv*=None,
dcosmo_rs=None)

`pysisyphus.calculators.Turbomole.index_strs_for_atoms`(*atoms*)

`pysisyphus.calculators.Turbomole.render_data_groups`(*raw_data_groups*)

6.4.7 DFTB+ 20.x

class `pysisyphus.calculators.DFTBp.DFTBp`(*parameter*, **args*, *slakos*=None, *root*=None, ***kwargs*)

Bases: [OverlapCalculator](#)

conf_key = 'dftbp'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

static get_excited_state_str(*root*, *forces*=False)

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

static get_gen_str(*atoms*, *coords*)

hubbard_derivs = {'3ob': {'Br': -0.0573, 'C': -0.1492, 'Ca': -0.034, 'Cl':
-0.0697, 'F': -0.1623, 'H': -0.1857, 'I': -0.0433, 'K': -0.0339, 'Mg': -0.02, 'N':
-0.1535, 'Na': -0.0454, 'O': -0.1575, 'P': -0.14, 'S': -0.11, 'Zn': -0.03}}

max_ang_moms = {'3ob': {'Br': 'd', 'C': 'p', 'Ca': 'p', 'Cl': 'd', 'F': 'p',
'H': 's', 'I': 'd', 'K': 'p', 'Mg': 'p', 'N': 'p', 'Na': 'p', 'O': 'p', 'P': 'd',
'S': 'd', 'Zn': 'd'}, 'mio-ext': {'C': 'p', 'H': 's', 'N': 'p', 'O': 'p'}}

parse_all_energies(*out_fn*=None, *exc_dat*=None)

`parse_energy(path)`

`static parse_exc_dat(text)`

`parse_forces(path)`

`parse_total_energy(text)`

`prepare_input(atoms, coords, calc_type)`

Meant to be extended.

`prepare_overlap_data(path)`

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

`run_calculation(atoms, coords, **prepare_kwargs)`

`store_and_track(results, func, atoms, coords, **prepare_kwargs)`

`pysisyphus.calculators.DFTBp.parse_mo(eigvec)`

`pysisyphus.calculators.DFTBp.parse_xplusy(text)`

6.5 Calculators with Ground state capabilities

6.5.1 MOPAC 2016

`class pysisyphus.calculators.MOPAC.MOPAC(method='PM7', **kwargs)`

Bases: [Calculator](#)

<http://openmopac.net/manual/>

`CALC_TYPES = {'energy': '1SCF', 'gradient': '1SCF GRADIENTS', 'hessian': 'DFORCE FORCE LET'}`

`METHODS = ['am1', 'pm3', 'pm6', 'pm6-dh2', 'pm6-d3', 'pm6-dh+', 'pm6-dh2', 'pm6-dh2x', 'pm6-d3h4', 'pm6-d3h4x', 'pm7', 'pm7-ts']`

`MULT_STRS = {1: 'SINGLET', 2: 'DOUBLET', 3: 'TRIPLET', 4: 'QUARTET', 5: 'QUINTET', 6: 'SEXTET', 7: 'SEPTET', 8: 'OCTET'}`

`base_cmd`

Do only SCF AUX: Creates a checkpoint file NOREO: Dont reorient geometry

`Type`

1SCF

`conf_key = 'mopac'`

`get_energy(atoms, coords, **prepare_kwargs)`

Meant to be extended.

`get_forces(atoms, coords, **prepare_kwargs)`

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

parse_energy(*path*)

static parse_energy_from_aux(*inp*, **args*, ***kwargs*)

parse_grad(*path*)

parse_hessian(*path*)

static parse_hessian_from_aux(*inp*, **args*, ***kwargs*)

prepare_coords(*atoms*, *coords*, *opt=False*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*, *opt=False*)

Meant to be extended.

read_aux(*path*)

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

6.5.2 Psi4

class pysisyphus.calculators.Psi4.**Psi4**(*method*, *basis*, *to_set=None*, *to_import=None*, *pcm='iefpcm'*, *solvent=None*, *write_fchk=False*, ***kwargs*)

Bases: [Calculator](#)

conf_key = 'psi4'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_fchk_str()

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
parse_energy(path)
parse_grad(path)
parse_hessian(path)
prepare_input(atoms, coords, calc_type)
    Meant to be extended.
run_calculation(atoms, coords, **prepare_kwargs)
```

6.5.3 QCEngine

6.5.4 XTB 6.x

```
class pysisyphus.calculators.XTB.OptResult(opt_geom, opt_log)
```

Bases: tuple

opt_geom

Alias for field number 0

opt_log

Alias for field number 1

```
class pysisyphus.calculators.XTB.XTB(gbsa="", alpb="", gfn=2, acc=1.0, iterations=250, etemp=None,
    retry_etemp=None, restart=False, topo=None, topo_update=None,
    quiet=False, **kwargs)
```

Bases: [Calculator](#)

```
__init__(gbsa="", alpb="", gfn=2, acc=1.0, iterations=250, etemp=None, retry_etemp=None, restart=False,
    topo=None, topo_update=None, quiet=False, **kwargs)
```

XTB calculator.

Wrapper for running energy, gradient and Hessian calculations by XTB.

Parameters

- **gbsa** (*str*, *optional*) -- Solvent for GBSA calculation, by default no solvent model is used.
- **alpb** (*str*, *optional*) -- Solvent for ALPB calculation, by default no solvent model is used.
- **gfn** (*int* or *str*, *must be* (0, 1, 2, or "ff")) -- Hamiltonian for the XTB calculation (GFN0, GFN1, GFN2, or GFNFF).
- **acc** (*float*, *optional*) -- Accuracy control of the calculation, the lower the tighter several numerical thresholds are chosen.
- **iterations** (*int*, *optional*) -- The number of iterations in SCC calculation.
- **topo** (*str*, *optional*) -- Path the a GFNFF-topolgy file. As setting up the topology may take some time for sizable systems, it may be desired to reuse the file.
- **topo_update** (*int*) -- Integer controlling the update interval of the GFNFF topology update. If supplied, the topology will be recreated every N-th calculation.
- **mem** (*int*) -- Mememory per core in MB.
- **quiet** (*bool*, *optional*) -- Suppress creation of log files.

```

static check_termination(inp, *args, **kwargs)

conf_key = 'xtb'

get_energy(atoms, coords, **prepare_kwargs)
    Meant to be extended.

get_forces(atoms, coords, **prepare_kwargs)
    Meant to be extended.

get_hessian(atoms, coords, **prepare_kwargs)
    Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
    Preferably, this method should provide an analytical Hessian.

get_mdrestart_str(coords, velocities)
    coords and velocities have to given in au!

get_pal_env()

get_retry_args()

parse_charges(fn=None)

parse_charges_from_json(fn=None)

parse_energy(path)

parse_gradient(path)

parse_hessian(path)

parse_md(path)

parse_opt(path, keep_log=False)

parse_topo(path)

prepare_add_args(xcontrol=None)

prepare_coords(atoms, coords)
    Get 3d coords in Angstrom.
    Reshape internal 1d coords to 3d and convert to Angstrom.

    Parameters
    • atoms (iterable) -- Atom descriptors (element symbols).
    • coords (np.array, 1d) -- 1D-array holding coordinates in Bohr.

    Returns
    coords -- 3D-array holding coordinates in Angstrom.

    Return type
    np.array, 3d

prepare_input(atoms, coords, calc_type, point_charges=None)
    Meant to be extended.

reattach(last_calc_cycle)

```

```
run_calculation(atoms, coords, **prepare_kwargs)

run_md(atoms, coords, t, dt, velocities=None, dump=1)
    Expecting t and dt in fs, even though xtb wants t in ps!

run_opt(atoms, coords, keep=True, keep_log=False)

run_topo(atoms, coords)

write_mdrestart(path, mdrestart_str)
```

6.5.5 Dalton

```
class pysisyphus.calculators.Dalton.Dalton(basis, method='hf', **kwargs)
    Bases: Calculator

    compute(mol, prop)

    conf_key = 'dalton'

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    prepare_input(atoms, coords)
        Meant to be extended.
```

6.5.6 OpenBabel

```
class pysisyphus.calculators.OBabel.OBabel(ff='gaff', mol=None, **kwargs)
    Bases: Calculator

    conv_dict = {'kcal/mol': 627.5094740630558, 'kJ/mol': 2625.4996394798254}

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    setup(atoms, coords)
```

6.5.7 CFOUR

```
class pysisyphus.calculators.CFOUR.CFOUR(cfour_input, wavefunction_dump=True, initden_file=None,
                                         **kwargs)

    Bases: Calculator
```

__init__(*cfour_input*, *wavefunction_dump=True*, *initden_file=None*, ***kwargs*)

CFOUR calculator.

Wrapper handling CFOUR ground state energy and gradient calculations.

Parameters

- **cfour_input** (*dict*) -- CFOUR keywords and values. Note: "on" must be encapsulated in quotes to avoid being translated to True by YAML.
- **wavefunction_dump** (*bool*, *optional*) -- Whether or not to keep ground state SCF orbitals for each geometry step.
- **initden_file** (*str*, *optional*) -- Path to an input initden file for use as a guess SCF density.

conf_key = 'cfour'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

keep(*path*)

Backup calculation results.

Parameters

path (*Path*) -- Temporary directory of the calculation.

Returns

kept_fns -- Dictionary holding the filenames that were backed up. The keys correspond to the type of file.

Return type

dict

parse_energy(*path*)

parse_gradient(*path*)

prepare(*inp*)

Prepare a temporary directory and write input.

Similar to `prepare_path`, but the input is also written into the prepared directory.

6.5.7.1 Paramters

inp

[str] Input to be written into the file `self.inp_fn` in the prepared directory.

returns

path: Path

Prepared directory.

prepare_coords(*atoms*, *coords*)

Get 3d coords in Bohr

Reshape internal 1d coords to 3d.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Bohr.

Return type

np.array, 3d

prepare_input(*atoms, coords, calc_type*)

Meant to be extended.

run_calculation(*atoms, coords, calc_type, **prepare_kwargs*)

pysisyphus.calculators.CFOUR.calc_centroid(coords_3d)

pysisyphus.calculators.CFOUR.calc_rot_matrix(source_coords_3d, target_coords_3d, source_centroid, target_centroid)

pysisyphus.calculators.CFOUR.parse_cfour_energy(out_fn)

pysisyphus.calculators.CFOUR.parse_cfour_gradient(grd_fn, out_fn, coords_pysis_frame_3d)

pysisyphus.calculators.CFOUR.read_cfour_geom(out_fn)

pysisyphus.calculators.CFOUR.rotate_gradient(gradient, coords_pysis_frame_3d, coords_comp_frame_3d)

6.6 Meta (wrapping) Calculators

6.6.1 ExternalPotential

6.6.2 Restraint

```
# General input structure for restraints
calc:
  type: ext
  # Multiple potentials could be specified here as a list
  potentials:
    # Primitive internal coordinate restraint
    - type: restraint
      # List of restraints; could also be multiple restraints. Each restraint is given as
      # list of 2 or 3 items.
      #
      # The first item always specifies an internal coordinate,
      # whereas the second argument is a force constant (in atomic units; actual units
      # depend on the coordinate). Optionally a reference value (third argument) can be
      # given. If omitted, the initial coordinate value is used as reference value.
      restraints: [[[BOND, 0, 1], 10, 3.0]]
      # The commented out input below would restrain the bond at its initial value.
      #restraints: [[[BOND, 0, 1], 10]]
      # Multiple restraints are specified as given below.
```

(continues on next page)

(continued from previous page)

```

    #restraints: [[[BOND, 0, 1], 10], [[BEND, 0, 1, 2], 1.0]]
calc:
  type: [actual calculator that is wrapped by ExternalPotential]

```

6.6.3 HarmonicSphere

6.6.4 LogFermi

6.6.5 RMSD

6.6.6 DFT-D3

Method to add DFT-D3 dispersion corrections as an external potential via the program developed by the Grimme group <https://www.chemie.uni-bonn.de/grimme/de/software/dft-d3/get_dft-d3>.

This is for use with calculators that do not natively provide D3 corrections (e.g. OpenMolcas). Usage mirrors that of other external potentials, with an example given below.

```

# General input structure for restraints
calc:
  type: ext
  # Multiple potentials could be specified here as a list
  potentials:
    # Add atom-pairwise D3 dispersion correction as a differentiable, external potential
    - type: d3
      # Functional is specified in TURBOMOLE format, all lower case.
      functional: pbe
      # Optional Becke-Johnson damping, default false, recommended true
      bjdamping: true

calc:
  type: [actual calculator that is wrapped by ExternalPotential]

```

```
class pysisyphus.calculators.DFTD3.DFTD3(geom,functional,bjdamping=False,**kwargs)
```

```
    calc(coords3d, gradient=False)
```

```
    conf_key = 'dftd3'
```

```
    parse_energy(path)
```

```
    parse_gradient(path)
```

6.6.7 AFIR

```
class pysisyphus.calculators.AFIR.AFIR(calculator, fragment_indices, gamma, rho=1, p=6,
                                       ignore_hydrogen=False, zero_hydrogen=True,
                                       complete_fragments=True, dump=True, h5_fn='afir.h5',
                                       h5_group_name='afir', **kwargs)
```

Bases: [Calculator](#)

```
__init__(calculator, fragment_indices, gamma, rho=1, p=6, ignore_hydrogen=False, zero_hydrogen=True,
         complete_fragments=True, dump=True, h5_fn='afir.h5', h5_group_name='afir', **kwargs)
```

Artificial Force Induced Reaction calculator.

Currently, there are no automated drivers to run large-scale AFIR calculations with many different initial orientations and/or increasing collision energy parameter. Nonetheless, selected AFIR calculations can be carried out by hand. After convergence, artificial potential & forces, as well as real energies and forces can be plotted with 'pysisplot --afir'. The highest energy point along the AFIR path can then be selected for a subsequent TS-optimization, e.g. via 'pysistrj --get [index] optimization.trj'.

Future versions of pysisyphus may provide drivers for more automated AFIR calculations.

Parameters

- **calculator** ([Calculator](#)) -- Actual QC calculator that provides energies and its derivatives, that are modified by the AFIR calculator, e.g., ORCA or Psi4.
- **fragment_indices** (List[List[int]]) -- List of lists of integers, specifying the separate fragments. If the indices in these lists don't comprise all atoms in the molecule, the remaining indices will be added as a separate fragment. If a AFIR calculation is carried out with 2 fragments and 'complete_fragments' is True (see below) it is enough to specify only the indices of one fragment, e.g., for a system of 10 atoms 'fragment_indices=[[0,1,2,3]]' is enough. The second system will be set up automatically with indices [4,5,6,7,8,9].
- **gamma** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) -- Collision energy parameter in au. For 2 fragments it can be a single integer, while for > 2 fragments a list of gammas must be given, specifying the pair-wise collision energy parameters. For 3 fragments 3 gammas must be given [_01, _02, _12], for 4 fragments 6 gammas would be required [_01, _02, _03, _12, _13, _23] and so on.
- **rho** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]], default: 1) -- Direction of the artificial force, either 1 or -1. The same comments as for gamma apply. For 2 fragments a single integer is enough, for > 2 fragments a list of rhos must be given (see above). For rho=1 fragments are pushed together, for rho=-1 fragments are pulled apart.
- **p** (int, default: 6) -- Exponent p used in the calculation of the weight function. Defaults to 6 and probably does not have to be changed.
- **ignore_hydrogen** (bool, default: False) -- Whether hydrogens are ignored in the calculation of the artificial force. All weights between atom pairs containing hydrogen will be set to 0.
- **zero_hydrogen** (bool, default: True) -- Whether to use 0.0 as covalent radius for hydrogen in the weight function. Compared to 'ignore_hydrogen', which results in zero weights for all atom pairs involving hydrogen, 'zero_hydrogen' may be non-zero, depending on the covalent radius of the second atom in the pair.

- **complete_fragments** (bool, default: True) -- Whether an incomplete specification in 'fragment_indices' is automatically completed.
- **dump** (bool, default: True) -- Whether an HDF5 file is created.
- **h5_fn** (str, default: 'afir.h5') -- Filename of the HDF5 file used for dumping.
- **h5_group_name** (str, default: 'afir') -- HDF5 group name used for dumping.
- ****kwargs** -- Keyword arguments passed to the Calculator baseclass.

afir_fd_hessian_wrapper(*coords3d*, *afir_grad_func*)

property charge

dump_h5(*atoms*, *coords*, *results*)

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

init_h5_group(*atoms*, *max_cycles=None*)

log_fragments()

property mult

set_atoms_and_funcs(*atoms*, *coords*)

Initially atoms was also an argument to the constructor of AFIR. I removed it so creation becomes easier. The first time a calculation is requested with a proper atom set everything is set up (cov. radii, afir function and corresponding gradient). Afterwards there is only a check if atoms != None and it is expected that all functions are properly set.

fragment_indices can also be incomplete w.r.t. to the number of atoms. If the sum of the specified fragment atoms is less than the number of atoms present then all remaining unspecified atoms will be gathered in one fragment.

write_fragment_geoms(*atoms*, *coords*)

exception pysisyphus.calculators.AFIR.CovRadiiSumZero

Bases: Exception

pysisyphus.calculators.AFIR.**afir_closure**(*fragment_indices*, *cov_radii*, *gamma*, *rho=1*, *p=6*, *prefactor=1.0*, *logger=None*)

rho=1 pushes fragments together, rho=-1 pulls fragments apart.

pysisyphus.calculators.AFIR.**get_data_model**(*atoms*, *max_cycles*)

6.6.8 ONIOM

```
class pysisyphus.calculators.ONIOMv2.LayerCalc(models, total_size, parent_layer_calc=None)
```

Bases: object

property charge

do_parent(with_parent)

static energy_from_results(model_energies, parent_energy=None)

get_energy(atoms, coords, with_parent=True)

get_forces(atoms, coords, with_parent=True)

get_hessian(atoms, coords, with_parent=True)

property mult

run_calculations(atoms, coords, method)

```
class pysisyphus.calculators.ONIOMv2.Link(ind, parent_ind, atom, g)
```

Bases: tuple

atom

Alias for field number 2

g

Alias for field number 3

ind

Alias for field number 0

parent_ind

Alias for field number 1

```
class pysisyphus.calculators.ONIOMv2.Model(name, calc_level, calc, parent_name, parent_calc_level,  
                                           parent_calc, atom_inds, parent_atom_inds,  
                                           use_link_atoms=True)
```

Bases: object

as_calculator(cap=False)

as_geom(all_atoms, all_coords)

capped_atoms_coords(all_atoms, all_coords)

create_bond_vec_getters(atoms)

create_links(atoms, coords, debug=False)

get_energy(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_forces(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_hessian(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_jacobian()

`get_sparse_jacobian()`

`log(message="")`

`parse_charges()`

`class pysisyphus.calculators.ONIOMv2.ModelDummyCalc(model, cap=False)`

Bases: `object`

`get_energy(atoms, coords)`

`get_forces(atoms, coords)`

`class pysisyphus.calculators.ONIOMv2.ONIOM(calcs, models, geom, layers=None, embedding="",
real_key='real', use_link_atoms=True, *args, **kwargs)`

Bases: `Calculator`

`__init__(calcs, models, geom, layers=None, embedding="", real_key='real', use_link_atoms=True, *args,
**kwargs)`

layer: list of models

`len(layer) == 1`: normal ONIOM, `len(layer) >= 1`: multicenter ONIOM.

model:

(sub)set of all atoms that resides in a certain layer and has a certain calculator.

`atom_inds_in_layer(index, exclude_inner=False)`

Returns list of atom indices in layer at index.

Atoms that also appear in inner layer can be excluded on request.

Parameters

- **index** (`int`) -- pasd
- **exclude_inner** (`bool`, `default=False`, `optional`) -- Whether to exclude atom indices that also appear in inner layers.

Returns

atom_indices -- List containing the atom indices in the selected layer.

Return type

list

`calc_layer(atoms, coords, index, parent_correction=True)`

property charge

```
embeddings = {'': '', 'electronic': 'Electronic embedding', 'electronic_rc':  
'Electronic embedding with redistributed charges', 'electronic_rcd': 'Electronic  
embedding with redistributed charges and dipoles'}
```

`get_energy(atoms, coords)`

Meant to be extended.

`get_forces(atoms, coords)`

Meant to be extended.

`get_hessian(atoms, coords)`

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
get_layer_calc(layer_ind)

property model_iter

property mult

run_calculation(atoms, coords)

run_calculations(atoms, coords, method)
```

```
pysisyphus.calculators.ONIOMv2.atom_inds_to_cart_inds(atom_inds)

pysisyphus.calculators.ONIOMv2.cap_fragment(atoms, coords, fragment, link_atom='H', g=None)

pysisyphus.calculators.ONIOMv2.get_embedding_charges(embedding, layer, parent_layer, coords3d)

pysisyphus.calculators.ONIOMv2.get_g_value(atom, parent_atom, link_atom)
```

6.6.9 Dimer

```
class pysisyphus.calculators.Dimer.Dimer(calculator, *args, N_raw=None, length=0.0189,
                                          rotation_max_cycles=15, rotation_method='fourier',
                                          rotation_thresh=0.0001, rotation_tol=1,
                                          rotation_max_element=0.001, rotation_interpolate=True,
                                          rotation_disable=False, rotation_disable_pos_curv=True,
                                          rotation_remove_trans=True, trans_force_f_perp=True,
                                          bonds=None, N_hessian=None, bias_rotation=False,
                                          bias_translation=False, bias_gaussian_dot=0.1, seed=None,
                                          write_orientations=True, forward_hessian=True, **kwargs)
```

Bases: [Calculator](#)

property C

Shortcut for the curvature.

property N

add_gaussian(atoms, center, N, height=0.1, std=0.0529, max_cycles=50, dot_ref=None)

property can_bias_f0

property can_bias_f1

property coords0

property coords1

curvature(f1, f2, N)

Curvature of the mode represented by the dimer.

direct_rotation(optimizer, prev_step)

do_dimer_rotations(rotation_thresh=None)

property energy0

property f0

property f1

property f1_bias

property f2

Never calculated explicitly, but estimated from f0 and f1.

fourier_rotation(*optimizer, prev_step*)

get_N_raw_from_hessian(*h5_fn, root=0*)

get_forces(*atoms, coords*)

Meant to be extended.

get_gaussian_energies(*coords, sum_=True*)

get_gaussian_forces(*coords, sum_=True*)

get_hessian(*atoms, coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

remove_translation(*displacement*)

property rot_force

rotate_coords1(*rad, theta*)

Rotate dimer and produce new coords1.

set_N_raw(*coords*)

property should_bias_f0

May lead to calculation of f0 and/or f1 if present!

property should_bias_f1

May lead to calculation of f0 and/or f1 if present!

update_orientation(*coords*)

class pysisyphus.calculators.Dimer.**Gaussian**(*height, center, std, N*)

Bases: object

energy(*R, height=None*)

forces(*R, height=None*)

exception pysisyphus.calculators.Dimer.**RotationConverged**

Bases: Exception

6.7 Pure Python calculators

6.7.1 Sympy 2D Potentials

```
class pysisyphus.calculators.AnaPotBase.AnaPotBase(V_str, scale=1.0, xlim=(-1, 1), ylim=(-1, 1),
                                                    levels=None, use_sympify=True, minima=None,
                                                    saddles=None)
```

Bases: [Calculator](#)

```
anim_coords(coords, interval=50, show=False, title_func=None)
```

```
anim_opt(opt, energy_profile=False, colorbar=False, figsize=(8, 6), show=False)
```

```
get_energy(atoms, coords)
```

Meant to be extended.

```
get_forces(atoms, coords)
```

Meant to be extended.

```
classmethod get_geom(coords, atoms=('X'), V_str=None, calc_kwargs=None, geom_kwargs=None)
```

```
get_geoms_from_stored_coords(coords_list, i=None, calc_kwargs=None, geom_kwargs=None)
```

```
get_hessian(atoms, coords)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
get_minima(i=None, calc_kwargs=None, geom_kwargs=None)
```

```
get_path(num, minima_inds=None)
```

```
get_saddles(i=None, calc_kwargs=None, geom_kwargs=None)
```

```
plot(levels=None, show=False, **figkwargs)
```

```
plot3d(levels=None, show=False, zlim=None, vmin=None, vmax=None, resolution=100, rcount=50,
        ccount=50, nan_above=None, init_view=None, colorbar=False, **figkwargs)
```

```
plot_coords(xs, ys, enum=True, show=False, title=None)
```

```
plot_eigenvalue_structure(grid=50, levels=None, show=False)
```

```
plot_geoms(geoms, **kwargs)
```

```
plot_irc(irc, *args, **kwargs)
```

```
plot_opt(opt, *args, **kwargs)
```

```
statistics()
```

6.7.2 Lennard-Jones

```
class pysisyphus.calculators.LennardJones.LennardJones(sigma=1.8897261251, epsilon=1, rc=None)
    Bases: Calculator
    calculate(coords3d)

    get_energy(atoms, coords)
        Meant to be extended.
    get_forces(atoms, coords)
        Meant to be extended.
```

6.7.3 FakeASE

```
class pysisyphus.calculators.FakeASE.FakeASE(calc)
    Bases: object
    get_atoms_coords(atoms)
    get_forces(atoms=None)
    get_potential_energy(atoms=None)
```

6.7.4 TIP3P

```
class pysisyphus.calculators.TIP3P.TIP3P(rc=9.44863062728914)
    Bases: Calculator
    Transferable Intermolecular Potential 3 Point
    aHOH = 104.52
    calculate(coords3d)

    charges

        coulomb_energy = (multiple of elem. charge * multiple of elem. charge)
            / (distance in bohr) * 1 / (4 * pi * vacuum permittivity)

        coulomb_prefactor converts everything to atomic units and it is ... drum roll ... 1. from scipy.constants
        import value as pcval self.coulomb_prefactor = (1 / (4 * np.pi) * pcval("elementary charge"))**2
            / pcval("Hartree energy") / pcval("Bohr radius") / pcval("vacuum electric permittivity")
    )
    coulomb(coords3d)

    epsilon = 0.0002423919586315716
    get_energy(atoms, coords)
        Meant to be extended.
    get_forces(atoms, coords)
        Meant to be extended.
```

$qH = 0.417$

$q0 = -0.834$

$r0H = 1.8088458464917874$

$\sigma = 5.953790025507198$

MINIMIZATION

Searching for minimum energy geometries of molecules is preferably done using second order methods that employ hessian information. Using the plain hessian H for step prediction through $p = -H^{-1}g$, with g being the gradient vector, may yield erroneous uphill steps when H has negative eigenvalues.

This can be understood by calculating the step in the basis of the hessian eigenvectors V .

$$\begin{aligned}\tilde{H} &= V^T H V \\ \tilde{g} &= V^T g \\ \tilde{p}_i &= -\frac{\tilde{g}_i}{\tilde{H}_{ii}}\end{aligned}\tag{7.4}$$

\tilde{H} , \tilde{g} and \tilde{p} are transformed into the eigenvector basis and the subscript i indicates the component belongs to the i -th eigenvalue (-vector) of H . As the gradient always points into the direction of greater function values dividing it by a negative eigenvalues \tilde{H}_{ii} will lead to a step in uphill direction along the i -th eigenvector. The step in the original basis is obtained by a simple back-transformation:

$$p = V\tilde{p}$$

A step in downhill direction can be ensured by introducing an appropriate shift parameter λ that must be smaller than the smallest eigenvalue of H :

$$\tilde{p}_i = -\frac{\tilde{g}_i}{\tilde{H}_{ii} - \lambda}$$

In *pysisyphus* the shift parameter λ is obtained by the Rational Function Optimization approach.

When the root-mean-square of the predicted step p drops below a certain threshold (default is 0.0025 au or rad) controlled GDIIS is tried. If GDIIS fails or is not yet possible a polynomial line-search is conducted. Using the latest two energies and the projection of the latest two gradients onto the last step the coefficients of a cubic polynomial are defined unambiguously. By requiring $d^2f(x)/dx^2 \geq 0$ and the equality holding at exactly one point also a quartic polynomial can be determined.

For now line-searches using a cubic polynomial are disabled as they seem to degrade the optimizer performance, so only the constrained quartic polynomial is tried. By also using projected hessian information the coefficients of a quintic polynomial can be determined, but this also seems to be inferior compared to the constrained quartic polynomial.

Convergence is indicated when the root-mean-square and the maximum value of the gradient and step drop below a certain threshold. It is not uncommon that the gradient convergence is achieved before step convergence. By using *overachieve_factor*: $[n, \text{float} > 1]$ in the YAML input convergence will be signalled, when gradient convergence is overachieved by factor n .

7.1 YAML Example

Below you can find an example YAML-input including the most important options that the user may want to modify when using the RFOptimizer.

```
opt:
  type: rfo                                # Optimization algorithm
  max_cycles: 50                          # Maximum number of optimization cycles

  overachieve_factor: 2                  # Indicate convergence, regardless of the
                                          # proposed step when max(grad) and rms(grad)
                                          # are overachieved by factor [n]

  do_hess: True                          # Calculate the hessian at the final geometry
                                          # after the optimization.

  #hessian_recalc: None                   # Recalculate exact hessian every n-th cycle

  #hessian_recalc_adapt: None             # Expects a float. Recalculate exact hessian
                                          # whenever the gradient norms drops below
                                          # 1/[n] of the gradient norm at the last hessian
                                          # recalculation.

  #hessian_init: fischer                  # Type of model hessian. Other options are: 'calc,
                                          # simple, xtb, lindh, swart, unit'

  #hessian_update: bfgs                   # Hessian-update. Other options are: 'flowchart,
                                          # damped_bfgs, bofill'. bofill is not recommended
                                          # for minimum searches.

  #small_eigval_thresh: 1e-8              # Neglect eigenvalues and corresponding eigenvectors
                                          # below this threshold.

  #max_micro_cycles: 50                   # No. of micro cycles for the RS-variants. Does not apply
                                          # to TRIM.

  #trust_radius: 0.3                      # Initial trust radius.
  #trust_max: 1.0                         # Max. trust radius
  #trust_min: 0.1                         # Min. trust radius

  #line_search: True                     # Do line search

  #gdiis_thresh: 0.0025                   # May do GDIIS if rms(step) falls below this threshold
  #gediis_thresh: 0.01                   # May do GEDIIS if rms(grad) falls below this threshold
  #gdiis: True                           # Do controlled GDIIS after 'gdiis_thresh' is reached
  #gediis: False                         # Do GEDIIS after 'gediis_thresh' is reached

calc:
  type: turbomole
  control_path: control_path_pbe0_def2svp_s1  # Path to the prepared calculation
  track: True                                # Activate excited state tracking
  ovlp_type: tden                            # Track with transition density matrix,
  ↪ overlaps
  charge: 0
```

(continues on next page)

(continued from previous page)

```

mult: 1
pal: 4
mem: 2000

geom:
  type: redund
  fn: cytosin.xyz

```

Further examples for optimizations from `.yaml` input can be found [here](#).

7.2 Convergence Thresholds

Optimization convergence is indicated when four criteria are met. Each cycle the root-mean-square and the absolute maximum of force and proposed step vectors are checked. The default criteria in pysisyphus correspond to the `opt=loose` in Gaussian. Different thresholds can easily be requested by supplying the respective keyword in the YAML input.

Convergence is also indicated, when `max(force)` and `rms(force)` are overachieved, by a certain factor (*overachieve_factor*, default=3), similar to ORCA's optimizer. Further keywords, controlling the convergence check are found below.

```

opt:
  ...
  thresh: gau_loose           # See table below for different thresholds.
  overachieve_factor: 3
  rms_force: null             # Derive four criteria from this value
  rms_force_only: False       # Only check rms(force)
  max_force_only: False       # Only check max(force)
  # Chain-of-States specific
  coord_diff_thresh: 0.01     # Terminate on insignificant coordinate change
  reparam_thresh: 0.001       # Terminate on insignificant coordinate change upon
  ↪ reparametrization
  ...

```

| Thresh- old | max(force) | rms(forces) | max(step) | rms(step) | Comment |
|-------------------|------------|-------------|-----------|-----------|---|
| <i>gau_loose</i> | 2.5e-3 | 1.7e-3 | 1.0e-2 | 6.7e-3 | default |
| <i>gau</i> | 4.5e-4 | 3.0e-4 | 1.8e-3 | 1.2e-3 | |
| <i>gau_tight</i> | 1.5e-5 | 1.0e-5 | 6.0e-5 | 4.0e-5 | |
| <i>gau_vtight</i> | 2.0e-6 | 1.0e-6 | 6.0e-6 | 4.0e-6 | |
| <i>baker</i> | 3.0e-4 | 2.0e-4 | 3.0e-4 | 2.0e-4 | energy difference and step are also checked |
| <i>never</i> | 2.0e-6 | 1.0e-6 | 6.0e-6 | 4.0e-6 | dummy thresholds; never converges |

7.3 Available Optimizers

Base class for optimizers that utilize Hessian information.

```
class pysisyphus.optimizers.Optimizer.ConvInfo(cur_cycle, energy_converged, max_force_converged,  
                                              rms_force_converged, max_step_converged,  
                                              rms_step_converged, desired_eigval_structure)
```

Bases: object

cur_cycle: int

desired_eigval_structure: bool

energy_converged: bool

get_convergence()

is_converged()

max_force_converged: bool

max_step_converged: bool

rms_force_converged: bool

rms_step_converged: bool

```
class pysisyphus.optimizers.Optimizer.Optimizer(geometry, thresh='gau_loose', max_step=0.04,  
                                              max_cycles=150, min_step_norm=1e-08,  
                                              assert_min_step=True, rms_force=None,  
                                              rms_force_only=False, max_force_only=False,  
                                              force_only=False,  
                                              converge_to_geom_rms_thresh=0.05, align=False,  
                                              align_factor=1.0, dump=False, dump_restart=False,  
                                              print_every=1, prefix="", reparam_thresh=0.001,  
                                              reparam_check_rms=True, reparam_when='after',  
                                              overachieve_factor=0.0,  
                                              check_eigval_structure=False, restart_info=None,  
                                              check_coord_diffs=True, coord_diff_thresh=0.01,  
                                              fragments=None, monitor_frag_dists=0, out_dir='.',  
                                              h5_fn='optimization.h5', h5_group_name='opt')
```

Bases: object

```
__init__(geometry, thresh='gau_loose', max_step=0.04, max_cycles=150, min_step_norm=1e-08,  
        assert_min_step=True, rms_force=None, rms_force_only=False, max_force_only=False,  
        force_only=False, converge_to_geom_rms_thresh=0.05, align=False, align_factor=1.0,  
        dump=False, dump_restart=False, print_every=1, prefix="", reparam_thresh=0.001,  
        reparam_check_rms=True, reparam_when='after', overachieve_factor=0.0,  
        check_eigval_structure=False, restart_info=None, check_coord_diffs=True,  
        coord_diff_thresh=0.01, fragments=None, monitor_frag_dists=0, out_dir='.',  
        h5_fn='optimization.h5', h5_group_name='opt')
```

Optimizer baseclass. Meant to be subclassed.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.

- **thresh** (Literal['gau_loose', 'gau', 'gau_tight', 'gau_vtight', 'baker', 'never'], default: 'gau_loose') -- Convergence threshold.
- **max_step** (float, default: 0.04) -- Maximum absolute component of the allowed step vector. Utilized in optimizers that don't support a trust region or line search.
- **max_cycles** (int, default: 150) -- Maximum number of allowed optimization cycles.
- **min_step_norm** (float, default: 1e-08) -- Minimum norm of an allowed step. If the step norm drops below this value a ZeroStepLength-exception is raised. The unit depends on the coordinate system of the supplied geometry.
- **assert_min_step** (bool, default: True) -- Flag that controls whether the norm of the proposed step is check for being too small.
- **rms_force** (Optional[float], default: None) -- Root-mean-square of the force from which user-defined thresholds are derived. When 'rms_force' is given 'thresh' is ignored.
- **rms_force_only** (bool, default: False) -- When set, convergence is signalled only based on rms(forces).
- **max_force_only** (bool, default: False) -- When set, convergence is signalled only based on max(|forces|).
- **force_only** (bool, default: False) -- When set, convergence is signalled only based on max(|forces|) and rms(forces).
- **converge_to_geom_rms_thresh** (float, default: 0.05) -- Threshold for the RMSD with another geometry. When the RMSD drops below this threshold convergence is signalled. Only used with Growing Newton trajectories.
- **align** (bool, default: False) -- Flag that controls whether the geometry is aligned in every step onto the coordinates of the previous step. Must not be used with internal coordinates.
- **align_factor** (float, default: 1.0) -- Factor that controls the strength of the alignment. 1.0 means full alignment, 0.0 means no alignment. The factor mixes the rotation matrix of the alignment with the identity matrix.
- **dump** (bool, default: False) -- Flag to control dumping/writing of optimization progress to the filesystem
- **dump_restart** (bool, default: False) -- Flag to control whether restart information is dumped to the filesystem.
- **print_every** (int, default: 1) -- Report optimization progress every nth cycle.
- **prefix** (str, default: '') -- Short string that is prepended to several files created by the optimizer. Allows distinguishing several optimizations carried out in the same directory.
- **reparam_thresh** (float, default: 0.001) -- Controls the minimal allowed similarity between coordinates after two successive reparametrizations. Convergence is signalled if the coordinates did not change significantly.
- **reparam_check_rms** (bool, default: True) -- Whether to check for (too) similar coordinates after reparametrization.
- **reparam_when** (Optional[Literal['before', 'after']], default: 'after') -- Reparametrize before or after calculating the step. Can also be turned off by setting it to None.
- **overachieve_factor** (float, default: 0.0) -- Signal convergence when max(forces) and rms(forces) fall below the chosen threshold, divided by this factor. Convergence of max(step) and rms(step) is ignored.

- **check_eigval_structure** (bool, default: False) -- Check the eigenvalues of the modes we maximize along. Convergence requires them to be negative. Useful if TS searches are started from geometries close to a minimum.
- **restart_info** (default: None) -- Restart information. Undocumented.
- **check_coord_diffs** (bool, default: True) -- Whether coordinates of chain-of-sates images are checked for being too similar.
- **coord_diff_thresh** (float, default: 0.01) -- Unitless threshold for similary checking of COS image coordinates. The first image is assigned 0, the last image is assigned to 1.
- **fragments** (Optional[Tuple], default: None) -- Tuple of lists containing atom indices, defining two fragments.
- **monitor_frag_dists** (int, default: 0) -- Monitor fragment distances for N cycles. The optimization is terminated when the interfragment distances falls below the initial value after N cycles.
- **out_dir** (str, default: ' . ') -- String poiting to a directory where optimization progress is dumped.
- **h5_fn** (str, default: 'optimization.h5') -- Basename of the HDF5 file used for dumping.
- **h5_group_name** (str, default: 'opt') -- Groupname used for dumping of this optimization.

check_convergence(*step=None, multiple=1.0, overachieve_factor=None*)

Check if the current convergence of the optimization is equal to or below the required thresholds, or a multiple thereof. The latter may be used in initiating the climbing image.

dump_restart_info()

final_summary()

fit_rigid(*, *vectors=None, vector_lists=None, hessian=None*)

get_path_for_fn(*fn, with_prefix=True*)

get_restart_info()

log(*message, level=50*)

make_conv_dict(*key, rms_force=None, rms_force_only=False, max_force_only=False, force_only=False*)

abstract optimize()

postprocess_opt()

prepare_opt()

print_opt_progress(*conv_info*)

procrustes()

Wrapper for procrustes that passes additional arguments along.

report_conv_thresholds()

run()

```

scale_by_max_step(steps)

set_restart_info(restart_info)

write_cycle_to_file()

write_image_trjs()

write_results()

write_to_out_dir(out_fn, content, mode='w')

```

```
pysisyphus.optimizers.Optimizer.get_data_model(geometry, is_cos, max_cycles)
```

```

class pysisyphus.optimizers.HessianOptimizer.HessianOptimizer(geometry, trust_radius=0.5,
                                                             trust_update=True, trust_min=0.1,
                                                             trust_max=1,
                                                             max_energy_incr=None,
                                                             hessian_update='bfgs',
                                                             hessian_init='fischer',
                                                             hessian_recalc=None,
                                                             hessian_recalc_adapt=None,
                                                             hessian_xtb=False,
                                                             hessian_recalc_reset=False,
                                                             small_eigval_thresh=1e-08,
                                                             line_search=False, alpha0=1.0,
                                                             max_micro_cycles=25,
                                                             rfo_overlaps=False, **kwargs)

```

Bases: [Optimizer](#)

```

__init__(geometry, trust_radius=0.5, trust_update=True, trust_min=0.1, trust_max=1,
         max_energy_incr=None, hessian_update='bfgs', hessian_init='fischer', hessian_recalc=None,
         hessian_recalc_adapt=None, hessian_xtb=False, hessian_recalc_reset=False,
         small_eigval_thresh=1e-08, line_search=False, alpha0=1.0, max_micro_cycles=25,
         rfo_overlaps=False, **kwargs)

```

Baseclass for optimizers utilizing Hessian information.

Parameters

- **geometry** ([Geometry](#)) -- Geometry to be optimized.
- **trust_radius** (float, default: 0.5) -- Initial trust radius in whatever unit the optimization is carried out.
- **trust_update** (bool, default: True) -- Whether to update the trust radius throughout the optimization.
- **trust_min** (float, default: 0.1) -- Minimum trust radius.
- **trust_max** (float, default: 1) -- Maximum trust radius.
- **max_energy_incr** (Optional[float], default: None) -- Maximum allowed energy increased after a faulty step. Optimization is aborted when the threshold is exceeded.
- **hessian_update** (Literal['none', None, False, 'bfgs', 'damped_bfgs', 'flowchart', 'bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev'], default: 'bfgs') -- Type of Hessian update. Defaults to BFGS for minimizations and Bofill for saddle point searches.

- **hessian_init**(Literal['calc', 'unit', 'fischer', 'lindh', 'simple', 'swart', 'xtb', 'xtb1', 'xtbff'], default: 'fischer') -- Type of initial model Hessian.
- **hessian_recalc**(Optional[int], default: None) -- Recalculate exact Hessian every n-th cycle instead of updating it.
- **hessian_recalc_adapt**(Optional[float], default: None) -- Use a more flexible scheme to determine Hessian recalculation. Undocumented.
- **hessian_xtb**(bool, default: False) -- Recalculate the Hessian at the GFN2-XTB level of theory.
- **hessian_recalc_reset**(bool, default: False) -- Whether to skip Hessian recalculation after reset. Undocumented.
- **small_eigval_thresh**(float, default: 1e-08) -- Threshold for small eigenvalues. Eigenvectors belonging to eigenvalues below this threshold are discarded.
- **line_search**(bool, default: False) -- Whether to carry out a line search. Not implemented by a subclassing optimizers.
- **alpha0**(float, default: 1.0) -- Initial alpha for restricted-step (RS) procedure.
- **max_micro_cycles**(int, default: 25) -- Maximum number of RS iterations.
- **rfo_overlaps**(bool, default: False) -- Enable mode-following in RS procedure.
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

filter_small_eigvals(*eigvals*, *eigvecs*, *mask=False*)

get_alpha_step(*cur_alpha*, *rfo_eigval*, *step_norm*, *eigvals*, *gradient*)

get_augmented_hessian(*eigvals*, *gradient*, *alpha=1.0*)

static get_newton_step(*eigvals*, *eigvecs*, *gradient*)

get_newton_step_on_trust(*eigvals*, *eigvecs*, *gradient*, *transform=True*)

Step on trust-radius.

See Nocedal 4.3 Iterative solutions of the subproblem

get_rs_step(*eigvals*, *eigvecs*, *gradient*, *name='RS'*)

static get_shifted_step_trans(*eigvals*, *gradient_trans*, *shift*)

get_step_func(*eigvals*, *gradient*, *grad_rms_thresh=0.01*)

housekeeping()

Calculate gradient and energy. Update trust radius and hessian if needed. Return energy, gradient and hessian for the current cycle.

log_negative_eigenvalues(*eigvals*, *pre_str=""*)

prepare_opt(*hessian_init=None*)

property prev_eigvec_max

property prev_eigvec_min

static quadratic_model(*gradient*, *hessian*, *step*)

reset()


```

rfo_dict = {'max': (-1, 'max'), 'min': (0, 'min')}

static rfo_model(gradient, hessian, step)

save_hessian()

set_new_trust_radius(coeff, last_step_norm)

solve_rfo(rfo_mat, kind='min', prev_eigvec=None)

update_hessian()

update_trust_radius()

pysisyphus.optimizers.HessianOptimizer.dummy_hessian_update(H, dx, dg)

class pysisyphus.optimizers.RFOptimizer.RFOptimizer(geometry, line_search=True, gediis=False,
                                                    gdiis=True, gdiis_thresh=0.0025,
                                                    gediis_thresh=0.01, gdiis_test_direction=True,
                                                    max_micro_cycles=25, adapt_step_func=False,
                                                    **kwargs)

Bases: HessianOptimizer

__init__(geometry, line_search=True, gediis=False, gdiis=True, gdiis_thresh=0.0025, gediis_thresh=0.01,
        gdiis_test_direction=True, max_micro_cycles=25, adapt_step_func=False, **kwargs)

Rational function Optimizer.

Parameters
    • geometry (Geometry) -- Geometry to be optimized.
    • line_search (bool, default: True) -- Whether to carry out implicit line searches.
    • gediis (bool, default: False) -- Whether to enable GEDIIS.
    • gdiis (bool, default: True) -- Whether to enable GDIIS.
    • gdiis_thresh (float, default: 0.0025) -- Threshold for rms(forces) to enable GDIIS.
    • gediis_thresh (float, default: 0.01) -- Threshold for rms(step) to enable GEDIIS.
    • gdiis_test_direction (bool, default: True) -- Whether to the overlap of the RFO step
      and the GDIIS step.
    • max_micro_cycles (int, default: 25) -- Number of restricted-step microcycles. Disabled
      by default.
    • adapt_step_func (bool, default: False) -- Whether to switch between shifted Newton
      and RFO-steps.
    • **kwargs -- Keyword arguments passed to the Optimizer/HessianOptimizer baseclass.

optimize()

postprocess_opt()

```


MINIMIZATION WITH MICROITERATIONS

Pysisyphus allows layered optimizations using microiterations via the *LayerOpt* class. Required energies and its derivatives (gradients, Hessians) can either be calculated with pysisyphus' own ONIOM implementation or supplied through Unix sockets via the i-PI protocol.

In ONIOM calculations a small (model) system is treated using a high level of theory, while its surroundings (real system) are described at a lower, more economic level of theory. There may also be several (optional) intermediate layers. Please see the figure below.

Fig. 8.1: General structure of a n-layer ONIOM calculation in pysisyphus. Every box corresponds to a layer. Indices of the respective layers are given as red numbers in the top right corner of each layer.

Searching stationary points of such systems using classical optimization approaches with simultaneous relaxation of model and real system can be computationally quite costly. In cases where the model system converges before the real system, many unnecessary model system energy and gradients evaluations may be required to fully relax the total system, as one step requires gradient evaluations in all layers. Also, it may be prohibitively expensive to optimize the whole system using internal coordinates, so it would be desirable to treat only the model system using internal coordinates.

Optimization with microiterations offers a way to potentially reduce the number of costly calculations in the innermost layer. By fully relaxing the outer layers before taking a step in the innermost layer, superfluous gradient evaluations can be avoided. As the overall calculation time of an ONIOM energy/gradient is usually dominated by the innermost layer, great computational saving can be realized by decreasing the required number of such calculations.

In the present approach, outer layers are optimized in Cartesian coordinates with an economic optimizer like (pre-conditioned) limited memory BFGS (L-BFGS) or conjugate gradient (CG). As L-BFGS avoids operations with steep scaling costs like matrix diagonalization, relaxation of large system comprising thousands of atoms becomes possible with negligible computational overhead. In contrast to the outer layers, the inner layer is usually optimized using internal coordinates and an optimizer that utilizes an explicit Hessian matrix, as this often allows for the most efficient optimizations.

As energy and gradient evaluations in the outer layers should be cheap the inferior performance of optimizations in Cartesian coordinates should not lead to an overall runtime increase.

8.1 YAML input

When pysisyphus' native ONIOM calculator is to be used, appropriate input has to be given in the `calc:` section, that is, layer composition and the respective calculators. If energies & gradients are sent via sockets and the i-PI-protocol `calc:` can be left empty but the appropriate socket addresses have to be specified later in the `opt:` section (see below). A separate socket/address is required for each layer.

Independent of the actual layer, pysisyphus always expects vectorial quantities with appropriate shapes of the total system. The expected size of a force vector for a system comprising N atoms is $3N$. Vector components not belonging to frozen atoms not belonging to a given layer are ignored and may be zero. **It is expected, that the full valid ONIOM forces for the total system are sent with the innermost layer.** These forces are then also used to check for optimization convergence.

A layered optimization is requested via `type: layer` in the `opt:` section. If no further input is given, appropriate defaults will be used. Microiterations for the outer layers 0 to $n-2$ will be carried out using Cartesian coordinates and regularized L-BFGS, while the innermost layer is optimized in internal coordinates (`type: tric`) using rational function optimization (`type: rfo`).

If a more fine grained control is desired each layer can be controlled independently by further keywords in the `layers:` list. When `layers:` is present in the input all layers must be specified, but empty list entries are possible (see the lines only containing a comment in the example below). **One must start with the outmost layer;** the last list item corresponds to the innermost layer. Geometry and/or optimizer setup of each layer is controlled by the `geom:` and `opt:` sections in the list of layers with the usual keywords.

The usual keywords to control the optimization are supported, e.g., `max_cycles`, `thresh` etc.

```
geom:
  type:
    cartesian
  fn: [filename of initial geometry]
calc:
  type:
    oniom:
      ...
opt:
  type:
    layer:
      [layers:
        - # dummy for layer 0 (real system)
        - # dummy for layer 1 layer
        - # ...
        # - # dummy for layer n-1 (model system)
        - geom:
            type:
              cartesian:
            opt:
              type:
                lbfgs:
          ]
      # thresh: gau_loose
      # max_cycles: 150
```

Below you can find a full example for the ONIOM2-optimization of Hexaphenylethane using pysisyphus' ONIOM implementation.

8.2 Example using pysisyphus' ONIOM

```
# 2-layer ONIOM optimization of Hexaphenylethan. The model
# system comprises the two carbons of the central ethan moiety.
# The model system is described GFN2-XTB while the real system is
# treated using GFN-FF.
```

```
geom:
```

```
  type: cart
```

```
  fn: |
```

```
    68
```

| | | | |
|---|---------|---------|---------|
| C | 1.2639 | 0.6380 | 0.4785 |
| C | 2.1638 | 1.9009 | 0.1349 |
| C | 2.4167 | 2.3403 | -1.2006 |
| C | 3.2096 | 3.4576 | -1.4755 |
| C | 2.7467 | 2.7036 | 1.1595 |
| C | 3.5368 | 3.8191 | 0.8739 |
| C | 3.7741 | 4.1922 | -0.4416 |
| H | 2.0100 | 1.8572 | -2.0700 |
| H | 3.3815 | 3.7580 | -2.5014 |
| H | 2.5944 | 2.5089 | 2.1979 |
| H | 3.9618 | 4.4009 | 1.6820 |
| H | 4.3848 | 5.0587 | -0.6598 |
| C | 0.7891 | 0.0052 | -0.8980 |
| C | -1.0360 | -0.5225 | -2.5243 |
| C | -0.0432 | -1.1375 | -3.4197 |
| C | 1.2448 | -1.1452 | -3.0690 |
| C | -0.6577 | 0.0144 | -1.3528 |
| H | -1.4492 | 0.4671 | -0.7765 |
| H | -2.0793 | -0.4955 | -2.8238 |
| H | -0.3570 | -1.5703 | -4.3642 |
| C | 1.6732 | -0.5526 | -1.7810 |
| H | 1.9838 | -1.5872 | -3.7297 |
| H | 2.7287 | -0.5813 | -1.6050 |
| C | -0.0387 | 1.1726 | 1.2081 |
| C | -0.2929 | 2.5599 | 1.4341 |
| C | -1.0635 | 0.2863 | 1.6524 |
| C | -2.2174 | 0.7426 | 2.2933 |
| C | -2.4082 | 2.0998 | 2.5118 |
| C | -1.4505 | 3.0060 | 2.0772 |
| H | 0.3724 | 3.3423 | 1.1178 |
| H | -1.0153 | -0.7691 | 1.5014 |
| H | -2.9711 | 0.0356 | 2.6161 |
| H | -1.6065 | 4.0659 | 2.2335 |
| H | -3.3046 | 2.4506 | 3.0066 |
| C | 2.1016 | -0.4883 | 1.4312 |
| C | 3.4003 | -1.0269 | 0.6933 |
| C | 4.4239 | -0.1425 | 0.2420 |
| C | 5.5730 | -0.6012 | -0.4056 |
| C | 3.6509 | -2.4147 | 0.4669 |
| C | 4.8038 | -2.8631 | -0.1829 |
| C | 5.7602 | -1.9589 | -0.6243 |

(continues on next page)

(continued from previous page)

```

H      4.3778    0.9134    0.3928
H      6.3259    0.1044   -0.7337
H      2.9864   -3.1958    0.7882
H      4.9572   -3.9235   -0.3393
H      6.6530   -2.3115   -1.1243
C      2.5704    0.1468    2.8077
C      4.3159    0.7126    4.4630
C      3.3714    1.2844    5.3047
C      2.0334    1.2840    4.9354
C      3.9303    0.1495    3.2436
H      4.7333   -0.2864    2.6778
H      5.3575    0.6988    4.7582
H      3.6749    1.7164    6.2495
C      1.6376    0.7248    3.7185
H      1.2932    1.7141    5.5981
H      0.5868    0.7426    3.5290
C      1.2016   -1.7498    1.7785
C      0.9537   -2.1874    3.1154
C      0.6140   -2.5534    0.7571
C     -0.1758   -3.6680    1.0474
C     -0.4081   -4.0392    2.3643
C      0.1611   -3.3038    3.3950
H      1.3648   -1.7032    3.9822
H      0.7614   -2.3600   -0.2822
H     -0.6047   -4.2504    0.2418
H     -0.0069   -3.6027    4.4220
H     -1.0186   -4.9050    2.5861
calc:
type: oniom
calcs:
real:
type: xtb
gfn: ff
pal: 6
high:
type: xtb
pal: 6
models:
# The "real" does not have to be described, only the "real" system.
high:
# Use ethan-carbons for the model system
inds: [0,34]
# Use appropriate calculator for model system.
calc: high
opt:
# Use LayerOpt
type: layers
thresh: gau
# When the 'layer' key is present, the user has to provide one
# entry per layer in the optimization. When the layer is empty default
# values are used. The layers are given from the outmost to the innermost
# layer.

```

(continues on next page)

(continued from previous page)

```
# In the example above no other arguments for the outmost layer (# real dummy),
# while redundant internal coordinates are used for the innermost layer.
#
# When pysisyphus own ONIOM implementation is used no additional atom indices
# must be given here.
layers:
- # real dummy
- geom:
    type: redund
do_hess: True
```

8.3 Example with sockets & i-PI-protocol

Whereas pysisyphus can figure out the layer composition for the microcycles when its own ONIOM calculator is used, the user has to specify it when using sockets and the i-PI-protocol. When covalent bonds between layers exist link atoms (LAs) must be introduced. Given a layer with index *n*, atoms in layer *n*-1 that are replaced by LAs in the given layer are called link atom hosts (LAH). A given layer and its LAHs in **higher** layers must be optimized together.

Fig. 8.2: Layer structure of a ONIOM2 optimization using microiterations with link atoms. A given layer as well as connected link atom hosts in **higher** layers must be optimized simultaneously.

A simple example for a ONIOM2 optimization with microiterations is found below. Here, ethanal is optimized, with the model system comprising the carbonyl group (atoms 4, 5 and 6). This introduces a link atom between the two carbons 4 and 0, with carbon 0 becoming a LAH.

Fig. 8.3: ONIOM2 optimization of ethanal. All atoms surrounded by the dashed blue line are optimized in the innermost layer. Hydrogens are shown white, carbons grey and oxygen red.

In contrast to the first example using the native ONIOM implementation the user can omit any input in the `calc:` section. Now the socket addresses have to be given for every layer, starting with the total system. For the total system the atom indices can be omitted, as it is assumed that it comprises all atoms. For the remaining layers, the indices of all moving atoms including LAHs in higher layers have to be given.

```
geom:
  type: cartesian
  fn: lib:acetaldehyd_oniom.xyz
calc:
opt:
  type: layers
  layers:
    # Atom indices of the total system / (first layer) don't have to be
    # specified.
    - address: ./layer0sock
    # Optimize atoms of the model system [4, 5, 6] and the link atom host 0.
    - indices: [0,4,5,6]
      address: ./layer1sock
  thresh: gau
# The assert: section can be left out in actual inputs,
```

(continues on next page)

(continued from previous page)

```
# it is here for testing purposes.
assert:
    opt_geom.energy: -115.53522653
```

Another sensible choice for optimizing outer layers besides (regularized) L-BFGS may be preconditioned L-BFGS (*type: plbfgs*).

```
class pysisyphus.optimizers.LayerOpt.LayerOpt(geometry, layers=None, **kwargs)
```

Bases: *Optimizer*

```
check_convergence(*args, **kwargs)
```

Check if we must use the model optimizer to signal convergence.

```
property layer_num: int
```

```
property model_opt
```

Return the persistent optimizer belonging to the model system. We don't have to supply any coordinates to the optimizer of the most expensive layer, as it is persistent, as well as the associated geometry.

```
optimize()
```

Return type

None

```
postprocess_opt()
```

Return type

None

```
print_opt_progress(*args, **kwargs)
```

Pick the correct method to report opt_progress.

When the model optimizer decides convergence we also report optimization progress using its data and not the data from LayerOpt, where the total ONIOM gradient is stored.

```
class pysisyphus.optimizers.LayerOpt.Layers(geometry, opt_thresh, layers=None)
```

Bases: object

```
classmethod from_oniom_calculator(geometry, oniom_calc=None, layers=None, **kwargs)
```

```
pysisyphus.optimizers.LayerOpt.get_geom_kwargs(layer_ind, layer_mask)
```

```
pysisyphus.optimizers.LayerOpt.get_opt_kwargs(opt_key, layer_ind, thresh)
```

```
class pysisyphus.optimizers.LBFGS.LBFGS(geometry, keep_last=7, beta=1, max_step=0.2,
                                          double_damp=True, gamma_mult=False, line_search=False,
                                          mu_reg=None, max_mu_reg_adaptions=10, control_step=True,
                                          **kwargs)
```

Bases: *Optimizer*

```
__init__(geometry, keep_last=7, beta=1, max_step=0.2, double_damp=True, gamma_mult=False,
          line_search=False, mu_reg=None, max_mu_reg_adaptions=10, control_step=True, **kwargs)
```

Limited-memory BFGS optimizer.

See [1] Nocedal, Wright - Numerical Optimization, 2006 for a general discussion of LBFGS. See pysisyphus.optimizers.hessian_updates for the references related to double damping and pysisyphus.optimizers.closures for references related to regularized LBFGS.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.
- **keep_last** (int, default: 7) -- History size. Keep last 'keep_last' steps and gradient differences.
- **beta** (float, default: 1) -- Force constant in $-(H + I)^{-1}g$.
- **max_step** (float, default: 0.2) -- Upper limit for the absolute component of the step vector in whatever unit the optimization is carried out.
- **double_damp** (bool, default: True) -- Use double damping procedure to modify steps s and gradient differences y to ensure $sy > 0$.
- **gamma_mult** (bool, default: False) -- Estimate γ from previous cycle. Eq. (7.20) in [1]. See 'beta' argument.
- **line_search** (bool, default: False) -- Enable implicit linesearches.
- **mu_reg** (Optional[float], default: None) -- Initial guess for regularization constant in regularized LBFGS.
- **max_mu_reg_adaptions** (int, default: 10) -- Maximum number of trial steps in regularized LBFGS.
- **control_step** (bool, default: True) -- Whether to scale down the proposed step its biggest absolute component is equal to or below 'max_step'
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

get_lbfgs_step(*forces*)

optimize()

postprocess_opt()

reset()

```
class pysisyphus.optimizers.PreconLBFGS.PreconLBFGS(geometry, alpha_init=1.0, history=7,
                                                    precon=True, precon_update=1,
                                                    precon_getter_update=None, precon_kind='full',
                                                    max_step_element=None, line_search='armijo',
                                                    c_stab=None, **kwargs)
```

Bases: *Optimizer*

```
__init__(geometry, alpha_init=1.0, history=7, precon=True, precon_update=1,
         precon_getter_update=None, precon_kind='full', max_step_element=None, line_search='armijo',
         c_stab=None, **kwargs)
```

Preconditioned limited-memory BFGS optimizer.

See pysisyphus.optimizers.precon for related references.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.
- **alpha_init** (float, default: 1.0) -- Initial scaling factor for the first trial step in the explicit line search.
- **history** (int, default: 7) -- History size. Keep last 'history' steps and gradient differences.
- **precon** (bool, default: True) -- Whether to use preconditioning or not.
- **precon_update** (int, default: 1) -- Recalculate preconditioner P in every n -th cycle with the same topology.

- **precon_getter_update** (Optional[int], default: None) -- Recalculate topology for preconditioner P in every n-th cycle. It is usually sufficient to only determine the topology once at the beginning.
- **precon_kind** (Literal['full', 'full_fast', 'bonds', 'bonds_bends'], default: 'full') -- What types of primitive internal coordinates to consider in the preconditioner.
- **max_step_element** (Optional[float], default: None) -- Maximum component of the absolute step vector when no line search is carried out.
- **line_search** (Literal['armijo', 'armijo_fg', 'strong_wolfe', 'hz', None, False], default: 'armijo') -- Whether to use explicit line searches and if so, which kind of line search.
- **c_stab** (Optional[float], default: None) -- Regularization constant c in $(H + cI)^1$ in atomic units.
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

get_precon_getter()

optimize()

prepare_opt()

scale_max_element(*step*, *max_step_element*)

RELAXED SCANS

Pysisyphus currently supports 1d-relaxed scans, where one internal coordinate is kept frozen, while the remaining ones are relaxed. A sample input is given below.

```
geom:
  type: redund
  fn: lib:h2o2_hf_321g_opt.xyz
calc:
  type: pyscf
  basis: 321g
  pal: 4
scan:
  type: BOND
  indices: [2, 3]
  # start is optional; if not specified the initial value of the primitive
  # internal coordinate is used.
  start: 3.0
  end: 5.0
  # steps is optional and can be omitted, but then step_size must be given
  steps: 5
  # step_size: 0.4
  # Enable symmetric scan in both directions, starting from the initial coordinates
  # symmetric: False
  #
  # By specifying an 'opt' block, the default optimization values, e.g.
  # convergence thresholds can be altered.
  # opt:
  #   thresh: gau
```

If no *start* value is specified in the *scan:* block, then the initial value is used. Specification of *steps* is mandatory, as it determines the number of optimizations to carry out. If not directly given (*step_size*), a suitable step size is derived from an *end* value. Either *end* or *step_size* must be given.

The values *step_size*, *start*, *end* have to be provided in atomic units or radians. When the appropriate YAML constructor is used, inputs can still be in Ångström or degrees. The constructor *!angstrom* converts input in Ångström to Bohr and *!deg* converts input from degrees to radians. See below for an example.

```
[...]
scan:
  type: DIHEDRAL
  indices: [0, 1, 2, 3]
  steps: 5
```

(continues on next page)

(continued from previous page)

```
step_size: !deg 10.  
[...]
```

In the example above, a relaxed scan around a torsion is carried out. In total, 5 steps of 10 degree each will be taken. By using the YAML constructor *!deg*, the input can be given in degrees, instead of radians. Internally, the degrees input is then converted to the appropriate unit, in this case radians.

Naive relaxed scan around the equilibrium value of a primitive internal may be affected by hysteresis, if the system under study is sufficiently complicated.

Relaxed scans are always carried out in redundant internal coordinates. Please see Optimization of Minima for the available optimization options. Supported primitive internals (*type*) are found in: [Types of Primitive Coordinates](#).

All geometries, including the initial one are written to *relaxed_scan.trj* in the current working directory.

EXCITED STATE OPTIMIZATION

pysisyphus offers excited state (ES) tracking, to follow diadiabatic states along an optimization. ES tracking is enabled by putting *track: True* in the *calc* section of your YAML input. In ES optimizations pysisyphus may use additional programs (wfoverlap, Multiwfn, jmol) if requested. Please see the [installation instructions](#) for information on how to set them up.

Please consider reading the relevant sections (2 and 3; 4 discusses examples) of the [pysisyphus paper](#). Additionally the user should think about the relevance between equilibrium/non-equilibrium solvation when calculating ES-gradients with implicit solvation. Several programs, e.g., Gaussian use equilibrium solvation when doing ES-optimization. It is in the responsibility of the user to add the relevant keywords when using pysisyphus, e.g., `EqSolv` for Gaussian.

10.1 YAML Example

A bare-bone input for the S_1 optimization of the 1H-amino-keto tautomer of cytosin at the TD-DFT/PBE0/def2-SVP level of theory using ORCA is shown below. The full example is found [here](#).

```
opt:
calc:
  type: orca
  keywords: pbe0 def2-svp rijcosx def2/J def2-svp/C
  # Calculate 2 ES by TD-DFT, follow the first one
  blocks: "%tddft nroots 2 iroot 1 tda false end"
  charge: 0
  mult: 1
  pal: 4
  mem: 2000
  # ES-tracking related keywords follow from here
  # Enable ES-tracking, this is important.
  track: True
  # Track ES by transition density overlaps
  ovlp_type: tden
geom:
  type: redund
  fn: cytosin.xyz
```

Additional keywords are possible in the *calc* section. The default values are shown below.

```
calc:
  # Controls calculation of charge-density-differences cubes and rendering
  # Cubes are calculated by Multiwfn, rendering is handled by jmol.
```

(continues on next page)

(continued from previous page)

```

#
# Possible values are: (None, calc, render).
# None: No cube calculation/rendering
# calc: Cube calculation by Multiwfn.
# render: Same as 'calc' and cubes are then rendered by jmol.
cdds: None
# Overlap type. Using 'wf' requires the external wfoverlap binary. The remaining
# options are implemented directly in pysisyphus.
#
# Possible values are (wf, tden, nto, nto_org).
# wf: Wavefunction overlaps using the external wfoverlap program.
# tden: Transition density matrix overlaps.
# nto: Natural transition orbital overlaps.
# nto_org: Natural transition orbital overlaps as described by Garcia.
# top: Transition orbital projection
ovlp_type: wf
# Controls the reference cycle that is used in the overlap calculation. The default
# 'adapt' is recommended.
#
# Possible values are (first, previous, adapt)
# first: Keep first calculation as reference cycle. Reliable when only minor
#         geometrical changes are expected.
# previous: Use previous cycle as reference.
# adapt: Use adaptive algorithm. Please see the pysisyphus paper for a discussion
#         at the end of section 3.
ovlp_with: adapt
# Thresholds controlling the update of the reference cycle.
# The first number specifies the minimum overlap that must be exceeded, for an update
# of the reference cycle. Assuming a value of 0.5 (50 %), the reference cycle update
# is skipped, if the overlaps between the current states and the reference state don't
# exceed 50 %.
# The last two numbers define an interval for the ratio between the second highest
# overlap, and the highest overlap. If the ratio is small, e.g., below 0.3, then both
# states are sufficiently different, and no reference cycle update is needed. If the
→ratio
# is bigger (> 0.6), then the states are quite similar, and an update is currently not
# advised.
# Possible values [three positive floats between 0. and 1.]
adapt_args: [0.5, 0.3, 0.6]
# Explicitly calculate the AO-overlap matrix in a double molecule calculation. Only
# supported by Turbomole and Gaussian calculators. If False, the approximate AO
# overlap matrix is reconstructed from inverting the MO-coefficient matrix.
#
# Possible values: (True, False)
double_mol: False
# Absolute CI-coefficients below this threshold are ignored in the overlap calculation.
#
# Possible values: positive float
conf_thresh: 0.0001
#
# nto/natural transition orbital specific
#

```

(continues on next page)

(continued from previous page)

```

# Number of NTOs to consider in the overlap calculation. Only relevant for 'nto'
# and 'nto_org' ovlp_types.
#
# Possible values: positive integer
use_ntos: 4
# Dynamically decide on number of NTOs according to their participation ratio. Only
# relevant for 'nto_org'
#
# Possible values: boolean
pr_nto: False
#
# wfoverlap/wavefunction overlaps specific
#
# Number of core orbitals to neglect in a wfoverlap calculation. Only relevant
# for the 'wf' ovlp_type. Must be >= 0.
#
# Possible values: positive integer
ncore: 0
#
# tden/transition density matrix specific
#
# Controls which set of MO coefficients (at current cycle, or the reference cycle)
# is used to recover the AO overlap matrix.
#
# Possible values: (ref, cur)
mos_ref: cur
# Controls whether the set of MO coefficients that was NOT used for recovering the AO
# overlap matrix is re-normalized, using the recovered AO overlap matrix. If set to
# True and mos_ref = cur, then the MO coefficients at the reference cycle will be re-
# normalized, and vice versa.
#
# Possible values: (True, False)
mos_renorm: True

```

By brief reasoning it would seem that `mos_ref: ref` and `mos_renorm: True` are more sensible choices, which is possibly true. Right now the present defaults are kept for legacy reasons, and I'll update them after testing out the alternatives.

Please also see Example - Excited State Tracking for possible visualizations when optimizing ES.

10.2 Optimization of Conical Intersections

pysisyphus implements the [projected gradient method](#) using an [updated branching plane](#), as developed by Maede, Ohno and Morokuma. Currently, CI-optimization is not enabled for YAML input. An illustrative example is found in `tests/test_conic_intersect`.

CHAIN OF STATES METHODS

A chain of states (COS) comprises a set of distinct states (images) and is usually spanned between two minima on a potential energy surface (PES).

When properly relaxed, a COS coincides with a minimum energy path (MEP), or is a good approximation to it. Tangents can be defined for every COS image and together they make up a discretized path that describes the reaction/chemical transformation. The tangents are also used to divide the COS gradient into perpendicular and parallel components, w.r.t the tangents. Relaxation (optimization) of the COS is achieved by minimizing the perpendicular component of the gradient, so only the parallel component remains.

pysisyphus offers different COS implementations, namely Nudged Elastic Band (NEB), including its Adaptive and Free-End (and Free-End-Adaptive) modifications and different flavors of String methods (Growing String Method, GSM, and Simple Zero Temperature String, SZTS). The GSM implementation is also available for with internal coordinates.

11.1 String method

(When discussing String methods the COS 'images' are usually called 'nodes'.)

In the string method the whole COS is periodically (every n -th cycle) reparametrized by fitting a spline through all nodes and redistributing the nodes then along the spline, according to a predefined parametrization. By choosing between different parametrizations equal node spacing or higher resolution around the highest energy image (HEI) can be achieved. The tangents needed for the gradient projection are obtained as first derivatives of the spline.

Reparametrization every n -th cycle impedes efficient string optimization, and prevents the use of optimizers with some kind of history Conjugate Gradient (CG). The optimizer history is reset after each reparametrization and a simple Steepest Descent (SD) step is done after reparametrization.

11.2 Nudged Elastic Band

No reparametrization takes place in the NEB method. The parallel gradient component along the tangent is projected out and replaced by an artificial spring force. In principle, optimizing NEBs should be easier as there is no reparametrization and more sophisticated optimizers beyond SD can and should be employed.

11.3 General remarks

Converged COS produce good guesses for subsequent TS searches, when the (splined) HEI is determined. In *pysisyphus* subsequent TS searches are easily started by including *tsopt*: in the YAML input. Knowledge of the initial and final images of the COS is used to construct a more complete set of internal coordinates for the TS guess and it is less likely that important coordinates are missed. The initial imaginary mode to follow uphill is selected as the one featuring the highest overlap with HEI tangent.

11.4 YAML example(s)

Below you can find an example YAML-input including the most important options that the user may want to modify when running a GSM optimization.

```
precontr:                                # Preconditioning of translation & rotation
preopt:                                  # Preoptimize initial and final geometry
cos:
  type: gs                               # Do a growing string
  max_nodes: 9                           # Total string will have 9 + 2 == 11 images
  climb: False                            # Enable climbing image (CI), usually a good
  ↪ idea.
  climb_rms: 0.005                        # rms(forces) threshold for enabling CI
  climb_lanczos: False                    # Use tangent obtained from Lanczos algorithm
  ↪ for CI.
  climb_lanczos_rms: 0.005                # rms(forces) threshold for enabling Lanczos
  ↪ algorithm.
  reparam_check: rms                      # Criterion for growing new frontier nodes (rms/
  ↪ norm).
  perp_thresh: 0.05                       # Threshold for growing new frontier nodes.
  reparam_every: 2                        # Reparametrize every n-th cycle when NOT fully
  ↪ grown.
  reparam_every_full: 3                   # Reparametrize every n-th cycle when fully
  ↪ grown.
opt:
  type: string                            # Optimizer for GrowingString
  stop_in_when_full: -1                   # Stop string optimization N cycles after fully
  ↪ grown
                                           # Disabled by -1. Usually it's a better idea to
                                           # further converge the string, after is fully
  ↪ grown.
  align: False                            # Disable Kabsch algorithm. Should be True with
                                           # type: cart
  scale_step: global                      # Scale down step as whole (global) or per image
                                           # (per_image)
tsopt:
  type: rsprfo                            # Continue with TS-optimization of highest
  ↪ energy images
                                           # (HEI) using the RS-P-RFO algorithm
  do_hess: True                           # Calculate hessian at optimized TS geometry
  trust_max: 0.3
  thresh: gau_loose
calc:
```

(continues on next page)

(continued from previous page)

```

type: orca
keywords: "b3lyp 6-31G* rijcosx"
pal: 4
charge: 0
mult: 1
geom:
  type: dlc
  fn: [first_preopt.xyz, last_preopt.xyz] # Run GrowingString in delocalized internal_
↪coordinates
                                     # (preferred).

```

For NEB optimizations a different optimizer (not `type: string`) should be used, e.g., QuickMin `type: qm` in the beginning, and `type: lbfgs` later on. It is not yet possible to specify two different optimizers that are used in stages, so if is desired it must be done manually.

```

# Taken from examples/complex/06_diels_alder...
geom:
  type: cart
  fn: diels_alder.trj
calc:
  type: xtb
  charge: 0
  mult: 1
  pal: 4
preopt:
  max_cycles: 5
interpol:
  type: redund
  between: 10
↪images,
↪== 23
                                     # In NEBs the whole path is interpolated beforehand.
                                     # Possible values: redund/idpp/lst/linear
                                     # Interpolate n-geometries between every pair of supplied
                                     # geometries. For two this yields 1 + 10 + 1 == 12_
                                     # for three geometries this yields 1 + 10 + 1 + 10 + 1_
                                     # geometries.

cos:
  type: neb
  climb: False
  align_fixed: True
↪along the path.
                                     # Align the fixed atoms of the initial and final images_

opt:
  type: lbfgs
  align: True
↪the previous step
  align_factor: 0.9
↪and 1
                                     # Align the image of the current step with the image of_
                                     # If full alignment is not desired, a factor between 0_
                                     # can be specified.

  rms_force: 0.01
  max_step: 0.04
tsopt:
  type: rsirfo
  do_hess: True
  max_cycles: 75

```

(continues on next page)

(continued from previous page)

```

thresh: gau_tight
hessian_recalc: 7
irc:
  type: eulerpc
  rms_grad_thresh: 0.0005
endopt:

```

Further examples for COS optimizations from `.yaml` input can be found [here](#).

11.5 General advice for COS optimizations

- Start from optimized geometries or use the `preopt:` key in the YAML input.
- Consider fixing the initial and final images `fix_ends: True`
- Always use `align: True` when optimizing a COS in cartesian coordinates to remove translation and rotation. `align: True` must not be used when running a COS with DLC.
- Don't over-converge a COS. It is usually a better idea to converge a COS loosely and use the highest energy image (HEI) as a guess for a subsequent transition state (TS) search.
- If possible use a climbing image (CI) `climb: True`
- When running a growing string calculation (`type: gs`) use `stop_in_when_full: [n]` in the `opt:` section with a small integer `[n]` to stop the COS relaxation after the string is fully grown

11.6 Chain Of States base class

Base class for chain of state methods

```

class pysisyphus.cos.ChainOfStates.ChainOfStates(images, fix_first=True, fix_last=True,
                                                align_fixed=True, climb=False, climb_rms=0.005,
                                                climb_lanczos=False, climb_lanczos_rms=0.005,
                                                climb_fixed=True, energy_min_mix=False,
                                                scheduler=None, progress=False)

    as_xyz(comments=None)

    property atoms

    calculate_forces()

    property calculator

    property cart_coords
        Return a flat 1d array containing the cartesian coordinates of all images.

    check_for_climbing_start(ref_rms)

    clear()

    concurrent_force_calcs(images_to_calculate, image_indices)

```

property coords

Return a flat 1d array containing the coordinates of all images.

property coords3d

describe()

property energy

property forces

get_climbing_forces(*ind*)

get_climbing_indices()

get_dask_client()

get_fixed_indices()

get_hei_index(*energies=None*)

Return index of highest energy image.

get_image_calc_counter_sum()

get_perpendicular_forces(*i*)

[1] Eq. 12

get_splined_hei()

get_tangent(*i, kind='upwinding', lanczos_guess=None, disable_lanczos=False*)

[1] Equations (8) - (11)

get_tangents()

property gradient

property image_coords

property image_inds

property last_index

log(*message*)

logger = <Logger cos (DEBUG)>

property masses_rep

property max_image_num

property moving_images

property moving_indices

Returns the indices of the images that aren't fixed and can be optimized.

par_image_calc(*image*)

property perpendicular_forces

prepare_opt_cycle(*last_coords*, *last_energies*, *last_forces*)

Implements additional logic in preparation of the next optimization cycle.

Should be called by the optimizer at the beginning of a new optimization cycle. Can be used to implement additional logic as needed for AdaptiveNEB etc.

property results

rms(*arr*)

Root mean square

Returns the root mean square of the given array.

Parameters

arr (*iterable of numbers*)

Returns

rms -- Root mean square of the given array.

Return type

float

set_climbing_forces(*forces*)

set_coords_at(*i*, *coords*)

Called from helpers.procrustes with cartesian coordinates. Then tries to set cartesian coordinate as self.images[i].coords which will raise an error when coord_type != "cart".

set_images(*indices*, *images*)

set_vector(*name*, *vector*, *clear=False*)

set_zero_forces_for_fixed_images()

This is always done in cartesian coordinates, independent of the actual coord_type of the images as setting forces only work with cartesian forces.

valid_coord_types = ('cart', 'cartesian', 'dlc')

zero_fixed_vector(*vector*)

11.7 Chain Of State Methods

11.7.1 Nudged Elastic Band (NEB)

class pysisyphus.cos.NEB.**NEB**(*images*, *variable_springs=False*, *k_max=0.3*, *k_min=0.1*, *perp_spring_forces=None*, *bandwidth=None*, ***kwargs*)

Bases: [ChainOfStates](#)

fmt_k()

property forces

get_parallel_forces(*i*)

get_quenched_dneb_forces(*i*)

See [3], Sec. VI and [4] Sec. D.

```

get_spring_forces(i)

property parallel_forces

set_variable_springs()

update_springs()

```

11.7.2 Adaptive NEB

```

class pysisyphus.cos.AdaptiveNEB.AdaptiveNEB(images, adapt=True, adapt_fact=0.25, adapt_between=1,
                                              scale_fact=False, keep_hei=True, free_ends=True,
                                              **kwargs)

```

Bases: [NEB](#)

```

__init__(images, adapt=True, adapt_fact=0.25, adapt_between=1, scale_fact=False, keep_hei=True,
         free_ends=True, **kwargs)

```

(Free-End) Adaptive Nudged Elastic Band.

Parameters

- **images** (*list of Geometry objects*) -- Images of the band.
- **adapt** (*bool, default True*) -- Whether to adapt the image number or not. This switch is included to support the FreeEndNEB class, that is just a thin wrapper around this class.
- **adapt_fact** (*positive float*) -- Factor that is used to decide whether to adapt. The initial threshold is calculated by multiplying the RMS force of the band with this factor. When the RMS of the force falls below this threshold adaption takes place.
- **adapt_between** (*positive integer*) -- Number of images to interpolate between the highest energy image and its neighbours. The number of starting images must be higher or equal then $2 \cdot \text{adapt_between} + 3$, as we reuse/transfer the calculators from the starting images onto the new ones.
- **scale_fact** (*bool, default False*) -- Whether to increase adapt_fact in deeper levels. This may lead to earlier adaption.
- **keep_hei** (*bool, optional*) -- Whether to keep the highest energy image (usually a very good idea) or to interpolate only between the neighbouring images.
- **free_ends** (*bool, default True*) -- Whether to use modified forces on the end images.

```

adapt_this_cycle(forces)

```

Decide whether to adapt.

Parameters

forces (*np.array*) -- Forces of the previous optimization cycle.

Returns

adapt -- Flag that indicates if adaption should take place in this cycle.

Return type

bool

```

property forces

```

See Eq. (7) in [2].

prepare_opt_cycle(*args, **kwargs)

Check for adaption and adapt if needed.

See ChainOfStates.prepare_opt_cycle for a complete docstring.

update_adapt_thresh(forces)

Update the adaption threshold.

Parameters

forces (*np.array*) -- Forces of the previous optimization cycle.

11.7.3 Free-End NEB

class pysisyphus.cos.FreeEndNEB.**FreeEndNEB**(*args, fix_first=False, fix_last=False, **kwargs)

Bases: *AdaptiveNEB*

__init__(*args, fix_first=False, fix_last=False, **kwargs)

Simple Free-End-NEB method.

Derived from AdaptiveNEB with disabled adaptation. Only implements Eq. (7) from [2]. For other implementations please see the commit 01bc8812ca6f1cd3645d43e0337d9e3c5fb0ba55. There the other variants are present but I think Eq. (7) in [2] is the simplest & best bet.

11.7.4 Simple Zero-Temperature String

class pysisyphus.cos.SimpleZTS.**SimpleZTS**(images, param='equal', **kwargs)

Bases: *ChainOfStates*

reparametrize()

11.8 Growing Chain Of States base class

Base class for growing chain of state methods

class pysisyphus.cos.GrowingChainOfStates.**GrowingChainOfStates**(images, calc_getter, max_nodes=10, **kwargs)

property arc_dims

get_new_image_from_coords(coords, index)

property max_image_num

new_node_coords(k)

prepare_opt_cycle(*args, **kwargs)

Implements additional logic in preparation of the next optimization cycle.

Should be called by the optimizer at the beginning of a new optimization cycle. Can be used to implement additional logic as needed for AdaptiveNEB etc.

set_new_node(k)

11.9 Growing Chain Of State Methods

11.9.1 Growing String Method

```
class pysisyphus.cos.GrowingString.GrowingString(images, calc_getter, perp_thresh=0.05,  
                                                  param='equi', reparam_every=2,  
                                                  reparam_every_full=3, reparam_tol=None,  
                                                  reparam_check='rms', max_micro_cycles=5,  
                                                  reset_dlc=True, climb=False, **kwargs)
```

Bases: *GrowingChainOfStates*

property forces

property full_string_image_inds

property fully_grown

Returns whether the string is fully grown. Don't count the first and last node.

get_additional_print()

get_cur_param_density(kind=None)

get_new_image(ref_index)

Get new image by taking a step from self.images[ref_index] towards the center of the string.

get_tangent(i)

[1] Equations (8) - (11)

property image_inds

property left_size

property lf_ind

Index of the left frontier node in self.images.

property nodes_missing

Returns the number of nodes to be grown.

reparam_cart(desired_param_density)

reparam_dlc(desired_param_density, thresh=0.001)

reparametrize()

reset_geometries(ref_geometry)

property rf_ind

Index of the right frontier node in self.images.

property right_size

set_coords(image, coords)

spline(tangents=False)

property string_size

TRANSITION STATE OPTIMIZATION

To cite Frank Jensen: "Locating transition states (TSs) is black magic, especially in internal coordinates" and I can say this is true. The most promising TS optimizers in *pysisyphus* employ second derivative information (hessian) but locating TS is also possible using only first derivatives by means of the *dimer method* (DM).

TS optimizations should preferably be carried out in internal coordinates (*type: redund|dlc*). Before starting the actual TS search, one should **always** check the internal coordinates that *pysisyphus* sets up, by executing *pysistrj [xyz] -internals*, with *[xyz]* corresponding to the geometry you plan to use for the optimization. **Check carefully** if all supposed reaction coordinates are present. **If they are missing** define them manually with the *add_prims* key in the YAML input.

Calculation of the exact hessian can be avoided by using the DM by using *rx_coords: [list of primitives]* in combination with *hessian_init: fischer|lindh|swart|simple*. By using both options, a diagonal model hessian is calculated and modified for use in a TS optimization.

The DM can be used with a powerful preconditioned LBFGS optimizer (*type: lbfgs*).

12.1 Hessian Based TS Optimization

12.1.1 YAML example

Below you can find an example YAML-input including the most important options that the user may want to modify for the Hessian-based optimizers (RS-I-RFO, RS-P-RFO, TRIM).

```
tsopt:
  type: rsirfo|rsprfo|trim          # Optimization algorithm

  do_hess: True                     # Calculate the hessian at the final geometry
                                   # after the optimization.

  hessian_recalc: 1                 # Recalculate the exact hessian every n-th cycle
                                   # Very costly but a small number like 5 may be a good
                                   # idea.

  prim_coord: [BOND, 3, 18]         # Select the mode to follow uphill by overlap with
                                   # this primitive internal coordinate. Expects exactly
                                   # one typed primitive.

  rx_modes: [[[[BOND, 3, 18], 1]]] # Select initial mode based on overlaps with a
                                   # constructed mode(s). Can be seen as generalization of
                                   # prim_coord. Expects a a list of (lists of pairs, ...)
```

(continues on next page)

(continued from previous page)

```

↪with each                                # pair comprising a typed primitive and a phase
↪factor).                                #
                                           # See examples/tsopt/{05..,06..} for examples.

#rx_coords: [[BOND, 3, 18],]             # Obtain initial mode by modifying a model Hessian.
↪Has                                     # to be used with 'hess_init: swart/fischer/lindh|xtb'
                                           # to avoid calculation of exact hessian.
                                           # Also requires 'redund' or 'dlc' to work.

#root: 0                                 # Follow the n-th imaginary mode uphill, minimize
                                           # along the others.

#hessian_init: calc                      # Initial hessian is calculated exactly.

#hessian_update: bofill                  # Bofill hessian-update. Should not be modified for
                                           # TS optimizations.

#hessian_ref: [path]                    # Expects a path to a precalculated hessian at any
↪selected                               # level of theory. The emode to maximize along is
                                           # by highest overlap with imaginary mode from the
↪reference                               # hessian.

#max_micro_cycles: 50                    # No. of micro cycles for the RS-variants. Does not
↪apply                                  # to TRIM.

#trust_radius: 0.3                       # Initial trust radius.
#trust_max: 1.0                           # Max. trust radius
#trust_min: 0.1                           # Min. trust radius
calc:
  type: xtb
  pal: 4
  charge: 0
  mult: 1
geom:
  type: redund
  fn: shaken.geom_000.xyz
  add_prims: [[24, 20], ]                # If using internal coordinates ALWAYS check the
↪coordinates                             # that pysisyphus generates (pysistrj [xyz] --
                                           # some important (reaction) coordinates appears to be
↪internals). If                          # define them manually. In this example
                                           # we add a bond. If additional internal coordinates can
↪missing                                 # derived from the added primitives pysisyphus will do

```

(continues on next page)

(continued from previous page)

it.

Second-derivative-free TS optimization can be done using the *Dimer* method. Attached you can find a sample input.

Further examples for TS optimizations from *.yaml* input can be found [here](#).

12.2 Dimer Method

pysisyphus implements the dimer method (DM) as calculator, wrapping another calculator for the actual energy and gradient calculations. All DM configurations have to be done in the `calc:` section. The best results are obtained by optimizing the DM with LBFGS in connection with a preconditioner (PLBFGS), estimated from the Lindh model Hessian. in contrast to the TS optimizers mentioned above, PBLFGS is configured in the `opt` section of the YAML input. DM related information is logged to *dimer.log*. The current DM orientation is saved in files with extension *.N*. Animated *.trj* files of the DM orientation are saved to *.N.trj*.

12.2.1 YAML example

Below you can find an example for the DM, applied to the isomerization of HCN. Default values are commented out. See `examples/tsopt/02_hcn_tsopt_dimer` for the full example.

```
opt:
  type: plbfgs      # Preconditioned LBFGS
  thresh: baker    # Baker threshold, comparable to the Gaussian defaults
  do_hess: True     # Calculate Hessian at the end. May not be applicable to systems
                   # where Hessian calculation is infeasible.

calc:
  type: dimer      # Dimer calculator wrapping PySCF at HF/3-21G level of theory
  calc:
    type: pyscf
    basis: 321g
    pal: 2
    charge: 0
    mult: 1
    #N_raw: [file]      # Path to a file containing an initial dimer orientation
    #length: 0.0189     # Separation of dimer images from common midpoint
    #rotation_max_cycles: 15 # Maximum number of rotations
    #rotation_method: fourier
    #rotation_thresh: 1e-4 # rms(rotationa force) threshold
    #rotation_tol: 1     # Rotation tolerance in degree. If a proposed trial
                        # rotation angle is below the tolerance the rotation
                        # will be skipped.
    #rotation_max_element: 0.001 # Max. step element for "rotation_method: direct"
    #rotation_interpolate: True  # Interpolate force on image after rotation
    #rotation_remove_trans: True # Remove overall translation from N-vector
    #seed: 20182503         # Seed for the RNG for reproducibility reasons.

geom:
  type: cart
  fn: 01_hcn.xyz
```

12.3 General advice for TS optimizations

- Use as many exact Hessians as your computational budget allows (*hessian_recalc*: [n])
- Use tight convergence settings for your SCF. In contrast to codes like ORCA *pysisyphus* does not enforce this automatically
- Grid-based methods (DFT, some RI-types) may need finer grids to reduce numerical noise
- Try to preoptimize the TS using a cheap method (xtb) that allows *hessian_recalc*: 1
- Decrease current & allowed trust radius (*trust_radius*: 0.1 and *trust_max*: 0.3|0.5)
- Check for correct coordinate setup

12.4 TSHessianOptimizer base class

Base class for TS optimizers that employ a hessian.

```
class pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer(geometry, roots=None,
                                                                    root=0, hessian_ref=None,
                                                                    rx_modes=None,
                                                                    prim_coord=None,
                                                                    rx_coords=None,
                                                                    hessian_init='calc',
                                                                    hessian_update='bofill',
                                                                    hessian_recalc_reset=True,
                                                                    max_micro_cycles=50,
                                                                    trust_radius=0.3,
                                                                    trust_max=0.5,
                                                                    augment_bonds=False,
                                                                    min_line_search=False,
                                                                    max_line_search=False,
                                                                    assert_neg_eigval=False,
                                                                    **kwargs)
```

Bases: [HessianOptimizer](#)

Optimizer to find first-order saddle points.

```
__init__(geometry, roots=None, root=0, hessian_ref=None, rx_modes=None, prim_coord=None,
         rx_coords=None, hessian_init='calc', hessian_update='bofill', hessian_recalc_reset=True,
         max_micro_cycles=50, trust_radius=0.3, trust_max=0.5, augment_bonds=False,
         min_line_search=False, max_line_search=False, assert_neg_eigval=False, **kwargs)
```

Baseclass for transition state optimizers utilizing Hessian information.

Several arguments expect a typed primitive or an iterable of typed primitives. A typed primitive is specified as (PrimType, int, ...), e.g., for a bond between atoms 0 and 1: (BOND, 0, 1) or for a bend between the atom triple 0, 1, 2 as (BEND, 0, 1, 2).

Parameters

- **geometry** ([Geometry](#)) -- Geometry to be optimized.
- **roots** (Optional[List[int]], default: None) -- Indices of modes to maximize along, e.g., to optimize saddle points of 2nd order. Overrides 'root'.
- **root** (int, default: 0) -- Index of imaginary mode to maximize along. Shortcut for 'roots' with only one root.

- **hessian_ref** (Optional[str], default: None) -- Filename pointing to a pysisyphus HDF5 Hessian.
- **rx_modes** (*iterable of (iterable of (typed_prim, phase_factor))*) -- Select initial root(s) by overlap with a modes constructed from the given typed primitives with respective phase factors.
- **prim_coord** (*typed_prim*) -- Select initial root/imaginary mode by overlap with this internal coordinate. Shortcut for 'rx_modes' with only one internal coordinate.
- **rx_coords** (*iterable of (typed_prim)*) -- Construct imaginary mode comprising the given typed prims by modifying a model Hessian.
- **hessian_init** (Literal['calc', 'unit', 'fischer', 'lindh', 'simple', 'swart', 'xtb', 'xtb1', 'xtbff'], default: 'calc') -- Type of initial model Hessian.
- **hessian_update** (Literal['none', None, False, 'bfgs', 'damped_bfgs', 'flowchart', 'bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev'], default: 'bofill') -- Type of Hessian update. Defaults to BFGS for minimizations and Bofill for saddle point searches.
- **hessian_recalc_reset** (bool, default: True) -- Whether to skip Hessian recalculation after reset. Undocumented.
- **max_micro_cycles** (int, default: 50) -- Maximum number of RS iterations.
- **trust_radius** (float, default: 0.3) -- Initial trust radius in whatever unit the optimization is carried out.
- **trust_max** (float, default: 0.5) -- Maximum trust radius.
- **augment_bonds** (bool, default: False) -- Try to derive additional stretching coordinates from the imaginary mode.
- **min_line_search** (bool, default: False) -- Carry out line search along the imaginary mode.
- **max_line_search** (bool, default: False) -- Carry out line search in the subspace that is minimized.
- **assert_neg_eigval** (bool, default: False) -- Check for the existences for at least one significant negative eigenvalue. If enabled and no negative eigenvalue is present the optimization will be aborted.
- ****kwargs** -- Keyword arguments passed to the HessianOptimizer/Optimizer baseclass.

```
static do_line_search(e0, e1, g0, g1, prev_step, maximize, logger=None)
```

```
prepare_opt(*args, **kwargs)
```

```
property root
```

```
property roots
```

```
step_and_grad_from_line_search(energy, gradient_trans, eigvecs, min_indices, max_indices)
```

```
update_ts_mode(eigvals, eigvecs)
```

```
valid_updates = ('bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev')
```

12.5 TS-Optimizer using hessian information

12.5.1 Restricted-Step Partitioned-RFO

```
class pysisyphus.tsoptimizers.RSPRFOptimizer.RSPRFOptimizer(geometry, roots=None, root=0,
                                                            hessian_ref=None, rx_modes=None,
                                                            prim_coord=None, rx_coords=None,
                                                            hessian_init='calc',
                                                            hessian_update='bofill',
                                                            hessian_recalc_reset=True,
                                                            max_micro_cycles=50,
                                                            trust_radius=0.3, trust_max=0.5,
                                                            augment_bonds=False,
                                                            min_line_search=False,
                                                            max_line_search=False,
                                                            assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

12.5.2 Restricted-Step Image-Function-RFO

```
class pysisyphus.tsoptimizers.RSIRFOptimizer.RSIRFOptimizer(geometry, roots=None, root=0,
                                                            hessian_ref=None, rx_modes=None,
                                                            prim_coord=None, rx_coords=None,
                                                            hessian_init='calc',
                                                            hessian_update='bofill',
                                                            hessian_recalc_reset=True,
                                                            max_micro_cycles=50,
                                                            trust_radius=0.3, trust_max=0.5,
                                                            augment_bonds=False,
                                                            min_line_search=False,
                                                            max_line_search=False,
                                                            assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

12.5.3 Trust-Region Image-Function Method

```
class pysisyphus.tsoptimizers.TRIM.TRIM(geometry, roots=None, root=0, hessian_ref=None,
                                          rx_modes=None, prim_coord=None, rx_coords=None,
                                          hessian_init='calc', hessian_update='bofill',
                                          hessian_recalc_reset=True, max_micro_cycles=50,
                                          trust_radius=0.3, trust_max=0.5, augment_bonds=False,
                                          min_line_search=False, max_line_search=False,
                                          assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

12.6 TS-Optimization without hessian information

12.6.1 Dimer method

```
class pysisyphus.calculators.Dimer.Dimer(calculator, *args, N_raw=None, length=0.0189,
                                         rotation_max_cycles=15, rotation_method='fourier',
                                         rotation_thresh=0.0001, rotation_tol=1,
                                         rotation_max_element=0.001, rotation_interpolate=True,
                                         rotation_disable=False, rotation_disable_pos_curv=True,
                                         rotation_remove_trans=True, trans_force_f_perp=True,
                                         bonds=None, N_hessian=None, bias_rotation=False,
                                         bias_translation=False, bias_gaussian_dot=0.1, seed=None,
                                         write_orientations=True, forward_hessian=True, **kwargs)

    property C
        Shortcut for the curvature.

    property N

    add_gaussian(atoms, center, N, height=0.1, std=0.0529, max_cycles=50, dot_ref=None)

    property can_bias_f0

    property can_bias_f1

    property coords0

    property coords1

    curvature(f1, f2, N)
        Curvature of the mode represented by the dimer.

    direct_rotation(optimizer, prev_step)

    do_dimer_rotations(rotation_thresh=None)

    property energy0

    property f0

    property f1

    property f1_bias

    property f2
        Never calculated explicitly, but estimated from f0 and f1.

    fourier_rotation(optimizer, prev_step)

    get_N_raw_from_hessian(h5_fn, root=0)

    get_forces(atoms, coords)
        Meant to be extended.

    get_gaussian_energies(coords, sum_=True)

    get_gaussian_forces(coords, sum_=True)
```

get_hessian(*atoms*, *coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

remove_translation(*displacement*)

property **rot_force**

rotate_coords1(*rad*, *theta*)

Rotate dimer and produce new coords1.

set_N_raw(*coords*)

property **should_bias_f0**

May lead to calculation of f0 and/or f1 if present!

property **should_bias_f1**

May lead to calculation of f0 and/or f1 if present!

update_orientation(*coords*)

class pysisyphus.calculators.Dimer.**Gaussian**(*height*, *center*, *std*, *N*)

energy(*R*, *height*=None)

forces(*R*, *height*=None)

exception pysisyphus.calculators.Dimer.**RotationConverged**

INTRINSIC REACTION COORDINATE (IRC)

By default two paths are integrated in plus- and minus-direction of the imaginary mode (transition vector, TV) at the transition state (TS). If the IRC is started from a non-stationary point with non-vanishing gradient the *downhill: True* argument can be set to integrate only one path and skip the initial hessian calculation.

IRCs are integrated in mass-weighted coordinates and the default integrator is *EulerPC*.

The initial displacement from the TS is done by requiring a certain energy lowering dE from moving along the TV and calculating the corresponding step length from a quadratic potential: $dE = \frac{1}{2}(k \cdot dq^2)$, with k being the eigenvalue of the TV (imaginary mode) and dq the required step length. The default required energy lowering is 0.0005 au. Alternatively the initial can be done by a prescribed length: *displ: length* and *displ_length: [desired step]* (default is $0.1\sqrt{\text{amu} \cdot \text{bohr}}$).

The resulting endpoints of the IRC integration can be further optimized to stationary points by adding the *endopt:* section (vide infra). By setting *fragments: True* in *endopt* separate fragments will be detected and optimized individually. This may be useful if the molecules are only weakly bound. By setting *fragments: total* the total system, as well as the separate fragments will be optimized.

By default IRC path data is dumped to *dump_fn: irc_data.h5* every *dump_every: 5* cycles. IRC paths can be plotted with *pysisplot --irc*.

13.1 YAML example(s)

Below you can find an example YAML-input including the most important options that the user may want to modify.

```
geom:
  fn: hfabstraction_ts_opt_xtb.xyz  # Input coordinates
calc:
  type: xtb                        # extended tight-binding calculator
  pal: 4
  charge: 0
  mult: 1
irc:
  type: eulerpc                   # Similar to EulerPC from Gaussian

  #rms_grad_thresh: 0.001         # Convergence threshold
  #displ: energy/length/energy_cubic # How to do the initial displacement
  #displ_energy: 0.001            # Energy lowering in au (Hartree)
  #displ_length: 0.1              # Step length along the TV
  #forward: True
  #backward: True
  #downhill: False                # Only integrate downhill, disables forward/backward
```

(continues on next page)

(continued from previous page)

```

#hessian_init: null          # Path to HDF5 Hessian file
#displ_third_h5: null        # Path to HDF5 file containing third derivative data
endopt:
#fragments: False/True/total # Detect & optimize fragments separately. Default is
                             # False. When set to 'total' the total system as well
                             # as the fragments are optimized.
do_hess: False               # Frequency calculation at the end

```

Further examples for IRC calculations from `.yaml` input can be found [here](#).

13.2 IRC base class

Base class for IRC integrators from which actual IRC integrators are derived.

```

class pysisyphus.irc.IRC.IRC(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True,
                              backward=True, root=0, hessian_init=None, displ='energy',
                              displ_energy=0.001, displ_length=0.1, displ_third_h5=None,
                              rms_grad_thresh=0.001, hard_rms_grad_thresh=None, energy_thresh=1e-06,
                              imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',
                              prefix="", dump_fn='irc_data.h5', dump_every=5)

__init__(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True, backward=True,
         root=0, hessian_init=None, displ='energy', displ_energy=0.001, displ_length=0.1,
         displ_third_h5=None, rms_grad_thresh=0.001, hard_rms_grad_thresh=None,
         energy_thresh=1e-06, imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',
         prefix="", dump_fn='irc_data.h5', dump_every=5)

```

Base class for IRC calculations.

Parameters

- **geometry** (`Geometry`) -- Transition state geometry, or initial geometry for downhill run.
- **step_length** (`float`, *optional*) -- Step length in unweighted coordinates.
- **max_cycles** (`int`, *optional*) -- Positive integer, controlling the maximum number of IRC steps taken in a direction (forward/backward/downhill).
- **downhill** (`bool`, *default=False*) -- Downhill run from a non-stationary point with non-vanishing gradient. Disables forward and backward runs.
- **forward** (`bool`, *default=True*) -- Integrate IRC in positive *s* direction.
- **backward** (`bool`, *default=True*) -- Integrate IRC in negative *s* direction.
- **root** (`int`, *default=0*) -- Use *n*-th root for initial displacement from TS.
- **hessian_init** (`str`, *default=None*) -- Path to Hessian HDF5 file, e.g., from a previous TS calculation.
- **displ** (`str`, one of ("energy", "length", "energy_cubic")) -- Controls initial displacement from the TS. 'energy' assumes a quadratic model, from which a step length for a given energy lowering (see 'displ_energy') is determined. 'length' corresponds to a displacement along the transition vector. 'energy_cubic' considers 3rd derivatives of the energy along the transition vector.
- **displ_energy** (`float`, *default=1e-3*) -- Required energy lowering from the TS in au (Hartree). Used with 'displ: energy|energy_cubic'.

- **displ_length** (*float*, *default=0.1*) -- Step length along the transition vector. Used only with 'displ: length'.
- **displ_third_h5** (*str*, *optional*) -- Path to HDF5 file containing 3rd derivative information. Used with 'displ: energy_cubic'.
- **rms_grad_thresh** (*float*, *default=1e-3*,) -- Convergence is signalled when to root mean square of the unweighted gradient is less than or equal to this value
- **energy_thresh** (*float*, *default=1e-6*,) -- Signal convergence when the energy difference between two points is equal to or less than 'energy_thresh'.
- **imag_below** (*float*, *default=0.0*) -- Require the wavenumber of the imaginary mode to be below the given threshold. If given, it should be a negative number.
- **force_inflection** (*bool*, *optional*) -- Don't indicate convergence before passing an inflection point.
- **check_bonds** (*bool*, *optional*, *default=True*) -- Report whether bonds are formed/broken along the IRC, w.r.t the TS.
- **out_dir** (*str*, *optional*) -- Dump everything into 'out_dir' directory instead of the CWD.
- **prefix** (*str*, *optional*) -- Short string that is prepended to all files that are created by this class, e.g., trajectories and HDF5 dumps.
- **dump_fn** (*str*, *optional*) -- Base name for the HDF5 files.
- **dump_every** (*int*, *optional*) -- Dump to HDF5 every n-th cycle.

property coords

dump_data(*dump_fn=None*, *full=False*)

dump_ends(*path*, *prefix*, *coords=None*, *trj=False*)

property energy

get_conv_fact(*mw_grad*, *min_fact=2.0*)

get_endpoint_and_ts_geoms()

get_full_irc_data()

get_irc_data()

get_path_for_fn(*fn*)

property gradient

initial_displacement()

Returns non-mass-weighted steps in +s and -s direction for initial displacement from the TS. Earlier version only returned one step, that was later multiplied by either 1 or -1, depending on the desired IRC direction (forward/backward). The current implementation directly returns two steps for forward and backward direction. Whereas for plus and minus steps for displ 'length' and displ 'energy'

$\text{step_plus} = -\text{step_minus}$

is valid, it is not valid for displ 'energy_cubic' anymore. The latter step is formed as

$x(ds) = ds * v0 + ds**2 * v1$

```
so
    x(ds) != -x(ds)

as
    ds * v0 + ds**2 * v1 != -ds * v0 - ds**2 * v1 .
```

So, all required step are formed directly and later used as appropriate.

See

<https://aip.scitation.org/doi/pdf/10.1063/1.454172> <https://pubs.acs.org/doi/10.1021/j100338a027>
<https://aip.scitation.org/doi/pdf/10.1063/1.459634>

```
irc(direction)

log(msg)

property m_sqrt

mass_weigh_hessian(hessian)

property mw_coords

property mw_gradient

postprocess()

prepare(direction)

report_bonds(prefix, bonds)

report_conv_thresholds()

run()

set_data(prefix)

unweight_vec(vec)

valid_displs = ('energy', 'length', 'energy_cubic')
```

13.3 IRC Integrators

13.3.1 Damped-Velocity-Verlet integrator

```
class pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet(geometry, v0=0.04, dt0=0.5,
                                                                error_tol=0.003,
                                                                max_cycles=150, **kwargs)
```

Bases: *IRC*

damp_velocity(velocity)

estimate_error(new_mw_coords)

mw_grad_to_acc(mw_grad)

Takes care of the units for the mass-weighted gradient. Converts units of a mass-weighted gradient [Hartree/(Bohr*amu)] to units of acceleration [sqrt(amu)*Bohr/fs²]. The 1e30 comes from converting second² to femto second².

prepare(*direction*)

step()

13.3.2 Euler integrator

Not recommended as it only produces reasonable results with very small step sizes.

class pysisyphus.irc.Euler.**Euler**(*geometry, step_length=0.01, **kwargs*)

Bases: [IRC](#)

step()

13.3.3 Euler-Predictor-Corrector integrator

Recommended IRC integrator and default choice.

class pysisyphus.irc.EulerPC.**EulerPC**(**args, hessian_recalc=None, hessian_update='bofill',
hessian_init='calc', max_pred_steps=500, loose_cycles=3,
dump_dwi=False, scipy_method=None, corr_func='mbs',
**kwargs*)

Bases: [IRC](#)

corrector_step(*init_mw_coords, step_length, dwi*)

get_integration_length_func(*init_mw_coords*)

prepare(**args, **kwargs*)

scipy_corrector_step(*init_mw_coords, step_length, dwi*)

Solve IRC equation $dx/ds = -g/|g|$ on DWI PES in mass-weighted coordinates. Integration done until self.step_length in unweighted coordinates is achieved.

step()

13.3.4 Gonzales-Schlegel-2 integrator

class pysisyphus.irc.GonzalezSchlegel.**GonzalezSchlegel**(*geometry, max_micro_cycles=20,
micro_step_thresh=0.001,
hessian_recalc=None, line_search=False,
**kwargs*)

Bases: [IRC](#)

micro_step(*counter*)

Constrained optimization on a hypersphere.

perp_component(*vec, perp_to*)

postprocess()

step()

13.3.5 Local-Quadratic-Approximation integrator

```
class pysisyphus.irc.LQA.LQA(geometry, N_euler=5000, **kwargs)
```

Bases: [IRC](#)

```
step()
```

13.3.6 Modified-Ishida-Morokuma-Komornicki integrator

Similar to the algorithm implemented by ORCA 4.

```
class pysisyphus.irc.IMKMod.IMKMod(geometry, corr_first=True, corr_first_size=0.5,
                                   corr_first_energy=True, corr_bisec_size=0.0025,
                                   corr_bisec_energy=True, **kwargs)
```

Bases: [IRC](#)

```
fit_grad_and_energies(energy0, grad0, energy1, direction)
```

```
fit_parabola(x, y)
```

```
step()
```

13.3.7 Runge-Kutta-4 integrator

Not recommended, as it is very slow.

```
class pysisyphus.irc.RK4.RK4(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True,
                               backward=True, root=0, hessian_init=None, displ='energy',
                               displ_energy=0.001, displ_length=0.1, displ_third_h5=None,
                               rms_grad_thresh=0.001, hard_rms_grad_thresh=None, energy_thresh=1e-06,
                               imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',
                               prefix='', dump_fn='irc_data.h5', dump_every=5)
```

Bases: [IRC](#)

```
get_k(mw_coords)
```

```
step()
```


DIABATIZATION

Adiabatic states can be diabaticized with a suitable rotation matrix, which in turn can be determined by maximizing a cost function that depends on the rotation matrix and some selected adiabatic properties (see below). Pysisyphus implements property-based direct diabaticization, as outlined by Subotnik¹ and Truhlar et al.²

Currently, dipole moments, the trace of the primitive quadrupole tensor and the electrostatic potential can be utilized for diabaticization.

14.1 General Algorithm

Diabatic states $\{\Xi_A\}$ can be obtained via unitary transformation of adiabatic states $\{\Psi_I\}$.

$$\Xi_A = \sum_{I=1} \Psi_I U_{IA}$$

which is equivalent to

$$\Xi = \mathbf{U}^\top \Psi$$

with rotation matrix \mathbf{U} , as well as adiabatic and diabatic state vectors Ψ and Ξ . Given two adiabatic states Ψ_1 and Ψ_2 \mathbf{U} is defined as

$$\mathbf{U} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}.$$

The related diabatic states Ξ_A and Ξ_B are

$$\Xi = \mathbf{U}^\top \Psi \tag{14.1}$$

$$\begin{bmatrix} \Xi_A \\ \Xi_B \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \Psi_1 \\ \Psi_2 \end{bmatrix} \tag{14.3}$$

$$\begin{aligned} \Xi_A &= \cos(\theta)\Psi_1 + \sin(\theta)\Psi_2 \\ \Xi_B &= -\sin(\theta)\Psi_1 + \cos(\theta)\Psi_2 \end{aligned} \tag{14.4}$$

Similarly, diabatic expectation values are calculated as linear combination of adiabatic expectation values:

$$\Xi_A | \hat{O} | \Xi_B = \sum_{IJ} U_{IA} \Psi_I | \hat{O} | \Psi_J U_{JB}$$

¹ <https://doi.org/10.1063/1.3042233>

² <https://doi.org/10.1063/1.4894472>, <https://doi.org/10.1063/1.4948728>

or in matrix notation

$$\mathbf{O}_{\text{dia}} = \mathbf{U}^T \mathbf{O}_{\text{adia}} \mathbf{U}.$$

By maximizing a cost function that depends on diabatic properties \mathbf{O}_{dia} , a rotation matrix \mathbf{U} suitable for diabaticization can be determined.

With a known rotation matrix \mathbf{U} , the diagonal adiabatic electronic energy matrix \mathbf{V} can be transformed to its diabatic equivalent \mathbf{D} , e.g., to determine diabatic couplings $|D_{12}|$.

$$\mathbf{D} = \mathbf{U}^T \mathbf{V} \mathbf{U} \quad (14.7)$$

$$\begin{bmatrix} D_{11} & D_{12} \\ D_{12} & D_{22} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} V_{11} & 0 \\ 0 & V_{22} \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (14.8)$$

Depending on the employed properties, different cost functions $f(\mathbf{U})$ are used. Subotnik^{Page 129, 1} proposed to maximize the magnitude of the diabatic dipole moments:

$$f_D(\mathbf{U}) = \sum_{A=1} |\Xi_A| \hat{\mu} |\Xi_A|^2.$$

This is the same cost function as in classical Boys localization of molecular orbitals. By also considering the trace of the primitive quadrupole tensor, $f_D(\mathbf{U})$ is extended to $f_{DQ}(\mathbf{U})$:

$$f_{DQ}(\mathbf{U}) = f_D(\mathbf{U}) + \sum_{A=1}^{N_Q} \sum_{j=1} \alpha_j |\Xi_A| \text{tr}(\hat{Q}) |\Xi_A|^2.$$

The contribution of the quadrupole moments to $f(\mathbf{U})$ is controlled by the scaling factor α_j . Here, subscript j refers to the j -th expansion center. Currently, only one expansion center ($N_Q = 1$) can be used in pysisyphus. By default α_j is set to $10 a_0^{-2}$.

In some cases, considering only the dipole moments is not enough to discriminate different adiabatic states and quadrupole moments have to be taken into account. Several examples are outlined by Truhlar et al.^{Page 129, 2}

Slight improvements may be possible by incorporating the electronic component of the electrostatic potential (ESP) Φ .

$$f_{DQ\Phi}(\mathbf{U}) = f_{DQ}(\mathbf{U}) + \sum_{A=1}^{N_\Phi} \sum_{k=1} \beta_k |\Xi_A| \hat{\Phi} |\Xi_A|^2$$

Here again, N_Φ denotes the number of expansion centers where the ESP is calculated. Similar to the quadrupole moments, only one expansion center is currently supported in pysisyphus. β_k is a scaling factor that controls the contribution of the electrostatic potential to the overall cost function $f_{DQ\Phi}(\mathbf{U})$.

Maximization of cost function f is achieved via repeated 2 x 2 rotations (Jacobi sweeps) of diabatic state pairs. The formulas to calculate the required rotation angle are given by Kleier³ and Pulay et al.⁴ The maximization is started with a unit-matrix guess for \mathbf{U} and usually converges quite rapidly.

A different approach based on unitary optimization is outlined by Lehtola,⁵ but is not available in pysisyphus.

Diabatization is carried out via the *pysisdia* entry hook. See below for an example.

³ <https://doi.org/10.1063/1.1681683>

⁴ <https://doi.org/10.1002/jcc.540140615>

⁵ <https://doi.org/10.1021/ct400793q>

14.2 Example

Diabatization of 3 adiabatic states using dipole moments requires the 3 adiabatic permanent dipole moments (0,0; 1,1; 2,2), as well as the respective transition dipole moments (0,1; 0,2; 1,2).

Listing 14.1: ../tests/test_diabatization/guanine_indole_dia.yaml

```
# Guanine-Indole cation, wB97X-D/6-31G*, TDA, 3 states.
# Values are taken from the Table S2 in the SI of
# https://doi.org/10.1063/5.0035593
adiabatic_energies: [0.0, 0.717, 0.905]
adiabatic_labels: [GS, LE, CT]
dipoles:
- [0, 0, -1.253, 0.270, -1.876]
- [1, 1, -1.691, -0.653, -2.083]
- [2, 2, 0.948, 0.733, -1.327]
- [0, 1, -1.224, -0.279, -0.325]
- [0, 2, 2.227, 0.866, 0.735]
- [1, 2, -1.574, -1.029, -0.517]
```

Executing `pysisdia guanine_indole_dia.yaml` produces the following output:

```
#####
# D-DIABATIZATION #
#####

Dipole moments
-----
[[[-1.253 -1.224  2.227]
  [-1.224 -1.691 -1.574]
  [ 2.227 -1.574  0.948]]

 [[ 0.27  -0.279  0.866]
  [-0.279 -0.653 -1.029]
  [ 0.866 -1.029  0.733]]

 [[-1.876 -0.325  0.735]
  [-0.325 -2.083 -0.517]
  [ 0.735 -0.517 -1.327]]]

Starting Jacobi sweeps.
000: P=169.63193103 dP=169.63193103
001: P=172.49529007 dP= 2.86335904
002: P=172.49529008 dP= 0.00000000
Jacobi sweeps converged in 3 cycles.

All energies are given in eV.

Adiabatic energy matrix V
-----
[[0.    0.    0.   ]
 [0.    0.717 0.   ]
 [0.    0.    0.905]]
```

(continues on next page)

(continued from previous page)

Rotation matrix U

```
-----
[[ 0.16289082  0.84920213  0.50231695]
 [-0.84058556  0.38601857 -0.38000734]
 [-0.51660672 -0.36034067  0.77670593]]
det(U)=1.00000
```

Diabatic energy matrix D = UVU

```
-----
[[ 0.74814945 -0.06418359 -0.13410224]
 [-0.06418359  0.2243505  -0.35846691]
 [-0.13410224 -0.35846691  0.64950005]]
```

Diabatic states sorted by energy

```
-----
0: 1, 0.2244 eV
1: 2, 0.6495 eV
2: 0, 0.7481 eV
```

Composition of diabatic states

```
-----
0 = + 0.1629·0(GS) - 0.8406·1(LE) - 0.5166·2(CT)
1 = + 0.8492·0(GS) + 0.3860·1(LE) - 0.3603·2(CT)
2 = + 0.5023·0(GS) - 0.3800·1(LE) + 0.7767·2(CT)
```

Weights U²

```
-----
[[0.02653342 0.72114427 0.25232232]
 [0.70658408 0.14901034 0.14440558]
 [0.2668825  0.1298454  0.6032721  ]]
```

Absolute diabatic couplings

```
-----
|D01| = 0.0642 eV
|D02| = 0.1341 eV
|D12| = 0.3585 eV
```

14.3 YAML input

All possible input options are outlined below. The numbers are just dummy values.

```
# Adiabatic energies, of the states to diabatize.
#
# List of floats.
# [V0, V1, ...]
# May be absolute or relative energies.
adiabatic_energies: [0.0, 0.6541, 0.7351]
# Dipole moments.
#
```

(continues on next page)

(continued from previous page)

```

# List of lists of length 5, each containing 2 integers followed by 3 floats.
# [state0, state1, dpm_x, dpm_y, dpm_z]
# The integers are adiabatic state indices, the 3 floats are the X, Y and Z
# components of a dipole moment vector.
# If both integers are the same, the 3 floats correspond to the permanent dipole moment
# of a state. Otherwise, they correspond to a transition dipole moment
# between two states.
dipoles: [
  [0, 0, -1.0, -2.0, -3.0],
  [1, 1, -2.0, -1.0, -3.0],
  [0, 1, -2.0, -1.0, -3.0],
]

#
# Optional input below
#

# Adiabatic state labels.
#
# List of strings.
# [label1, label2, ...]
# Must be of same length as adiabatic energies.
adiabatic_labels: [GS, LE, CT]
# Energy unit.
#
# Literal["eV", "Eh"]
unit: eV
# Trace of primitive quadrupole moment tensor.
#
# List of lists of length 3, each containing 2 integers followed by 1 float.
# [state0, state1, tr_qpm] = [state0, state1, (qpm_xx + qpm_yy + qpm_zz)]
# The same comments as for the dipole moments apply.
quadrupoles: [
  [0, 0, 1.0],
  [1, 1, -14.0],
  [0, 1, 33.0],
]
# Quadrupole moment scaling factor alpha in 1/a02.
#
# Float
alpha: 10.0
# Electronic component of electrostatic potential in au
epots: [
  [0, 0.48],
  [1, 0.12],
]
# Electrostatic potential scaling factor beta in a0.
#
# Float
beta: 1.0

```


PLOTTING

pysisyphus offers extensive visualization capabilities via the *pysisplot* command. All relevant information produced by *pysisyphus* and the interfaced quantum chemistry programs is dumped to HDF5 files, namely:

| Filename | Description |
|----------------------|--|
| optimization.h5 | Produced by all optimizers when <i>dump: True</i> (default). |
| afir.h5 | Contains true and modified energies & forces in AFIR calculations. |
| forward_irc_data.h5 | IRC data from forward run. |
| backward_irc_data.h5 | IRC data from backward run. |
| finished_irc_data.h5 | IRC data from forward & backward run. |
| overlap_data.h5 | From excited state calculations by <i>OverlapCalculator</i> classes. |

A help message that shows possible usages of *pysisplot* is displayed by

```
pysisplot --help
```

If needed, results from multiple optimizations are written to different HD5-groups of the same HDF5-file. Available groups can be listed either by the HDF5-tool *h5ls* or *pysisplot --h5_list [hd55 file]*. The desired group for visualization is then selected by the *--h5_group [group name]* argument.

15.1 Example - Diels-Alder reaction

The plotting capabilities of *pysisplot* are demonstrated for the example of a Diels-Alder reaction between ethene and 1-4-butadiene (see examples/06_diels_alder_xtb_preopt_neb_tsopt_irc) at xtb-GFN2 level of theory. Educts and product of the reaction are preoptimized for 5 cycles, then interpolated in internal coordinates. A chain of states (COS) is optimized loosely via a LBFGS optimizer. The highest energy image (HEI) is employed for a subsequent transition state (TS) optimization. Finally the intrinsic reaction coordinate (IRC) is obtained by means of an Euler-Predictor-Corrector integrator.

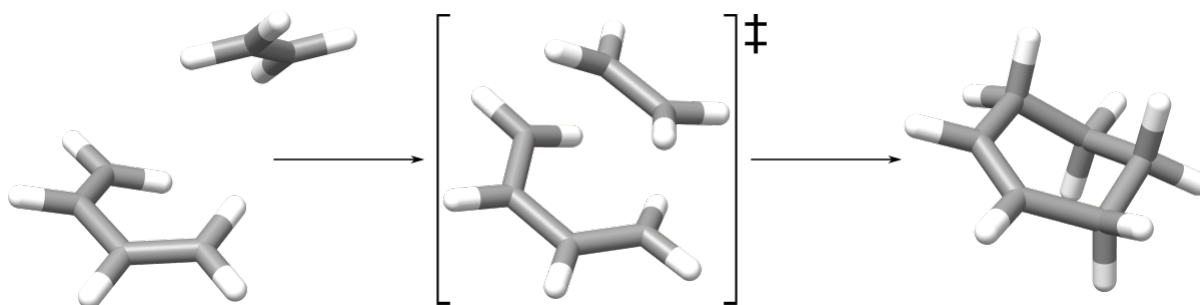


Fig. 15.1: Educts, TS and products of the Diels-Alder reaction between ethene and 1-4-butadiene.

15.1.1 Plotting optimization progress

The progress of an optimization can be plotted via

```
pysisplot --opt [--h5_fn optimization.h5] [--h5_group opt]
```

Arguments given in parantheses are the assumed defaults. The result is shown below for the preoptimizations of educts and product.

For COS optimizations all image energies of one optimization cycle are summed into a total energy, which is then plotted in the first panel. *pysisplot --opt* may not be the best choice in these cases. Use *pysisplot --cosens* and *pysisplot --cosforces* instead (see below).

15.1.2 Plotting COS optimization progress

Compared to simple surface-walking optimizations of single molecules, COSs consist of multiple images, each with its own energy and force vector. In this case a simple plot as shown above is not suitable. Instead of *pysisplot --opt* a better visualization is offered by *pysisplot --cosens* and *pysisplot --cosforces*. The latter two commands are compatible with all COS methods available in pysisyphus.

```
pysisplot --cosens [--h5_fn optimization.h5] [--h5_group opt]
```

This produces three plots.

1. Animated. COS image energies along the optimization.
2. Static. Energies of last optimization cycle.
3. Static. Energies of all cycles with earlier cycles given in a lighter shade and more recent cycles in a darker shade. The last cycle is splined and the position of the splined HEI is indicated.

Please note that equidistant image spacing is assumed for the latter two plots. Here only the two latter plots are shown.

The forces acting on the respective COS images can also be plotted.

```
pysisplot --coforces [--h5_fn optimization.h5] [--h5_group opt]
```

Please note that nothing is plotted for images 0 and 11, as they remained fixed in the optimization.

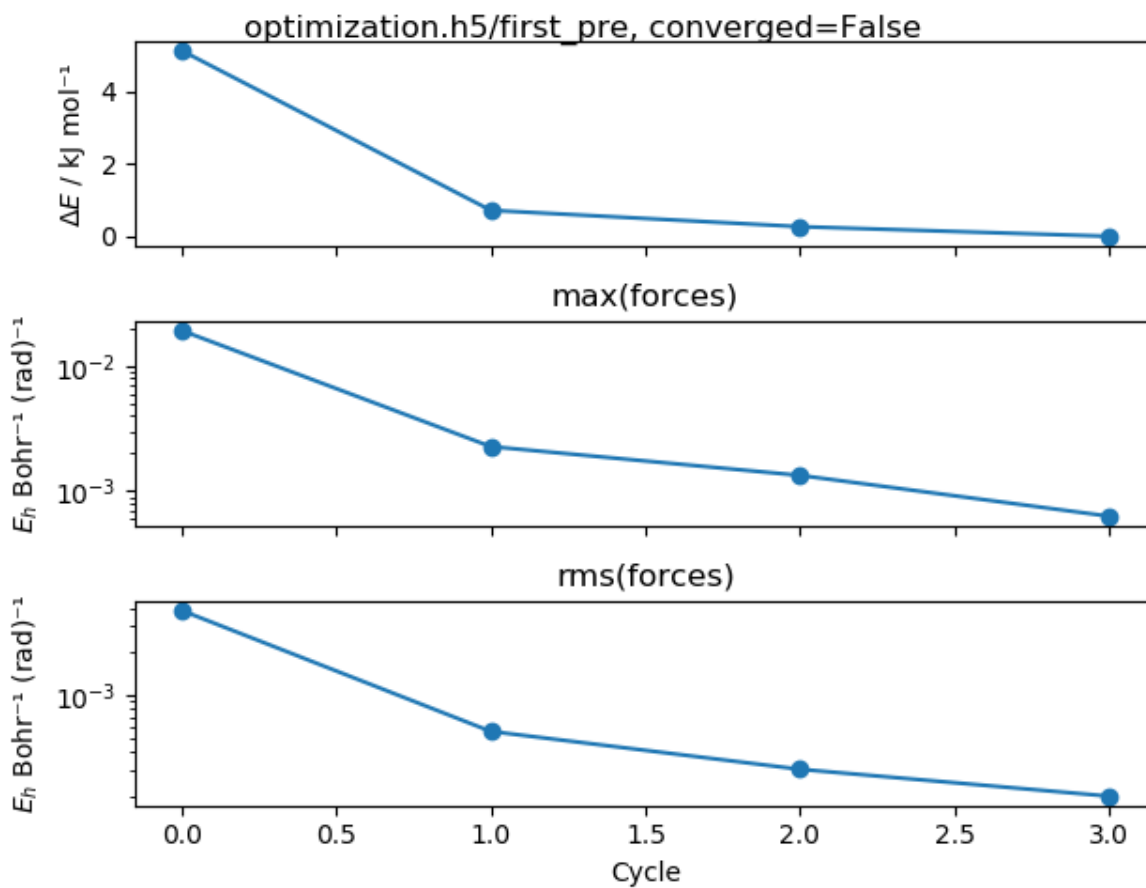


Fig. 15.2: Preoptimization of the Diels-Alder reaction educts ethene + 1,4-butadiene. The upper panel shows the energy change along the optimization. Middle and lower panel show the max and rms of the force. The HDF5 filename and group are noted in the image title.

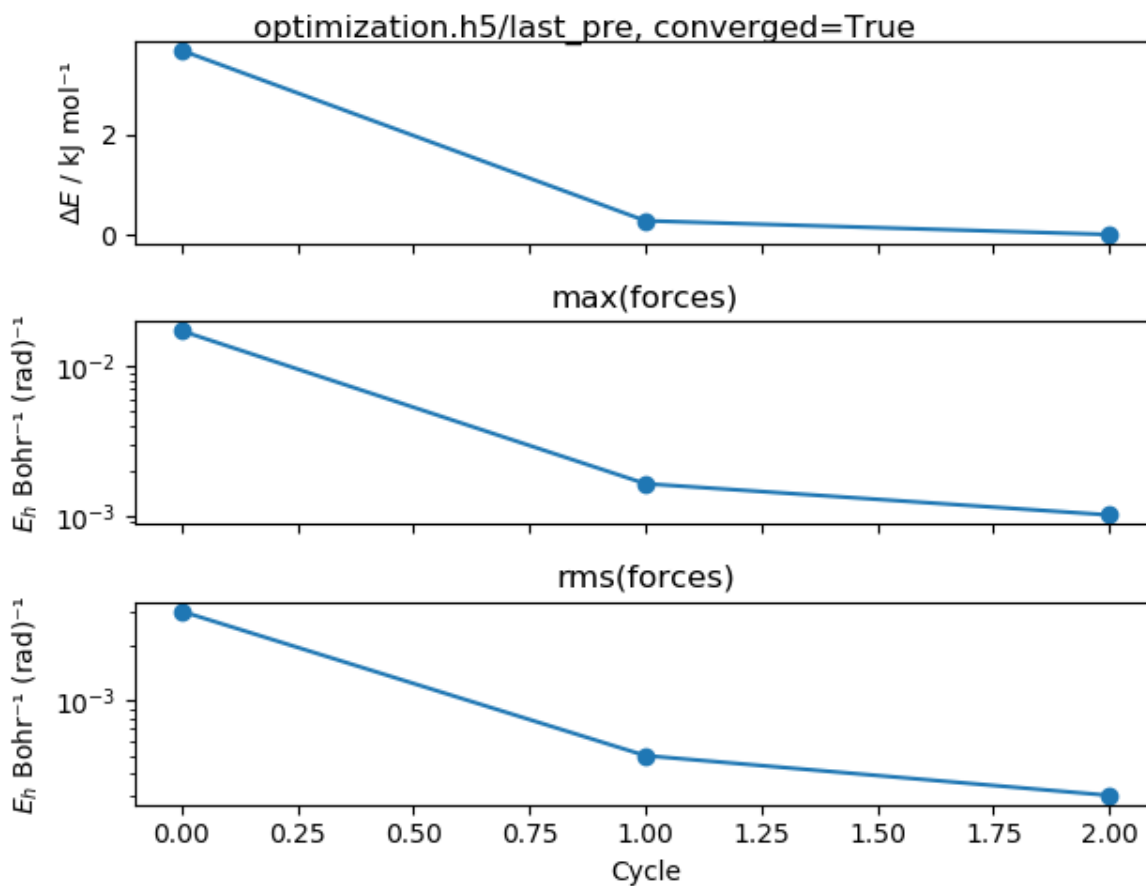


Fig. 15.3: Preoptimization of the Diels-Alder reaction product. Compared to the educts the optimization already converged after 4 cycles.

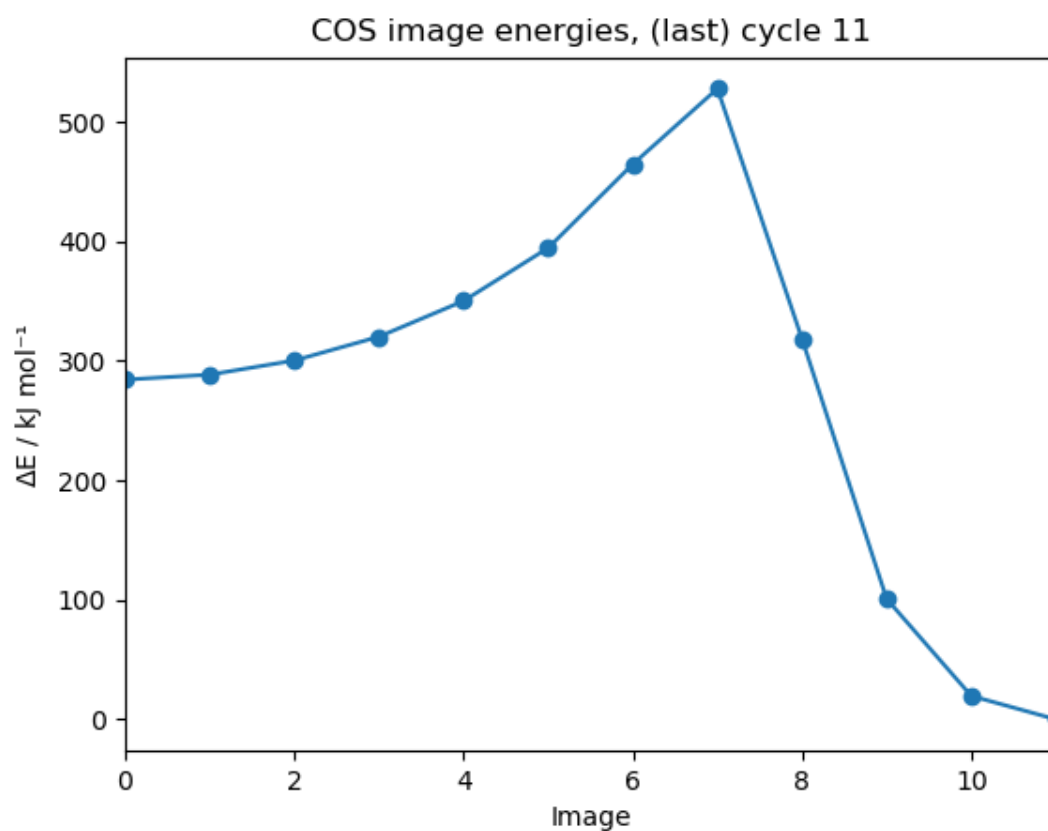


Fig. 15.4: COS image energies of the last (most recent) optimization cycle.

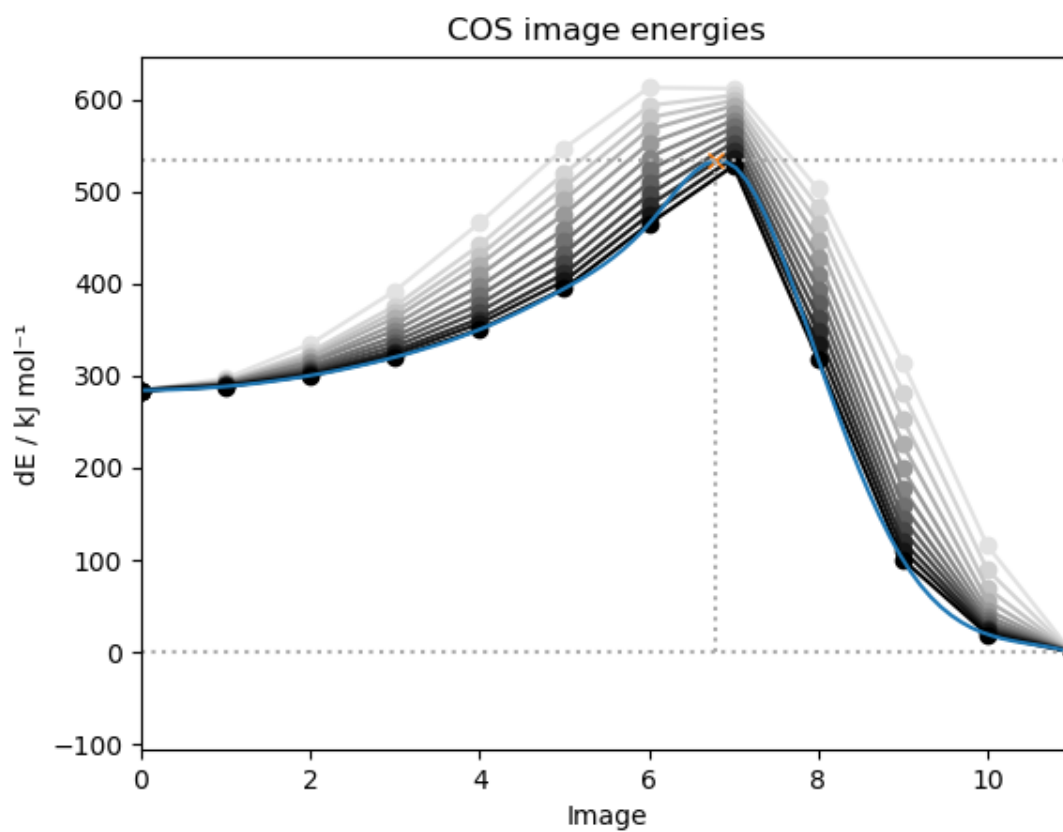


Fig. 15.5: COS image energies of all optimization cycles. Note that the actual difference between image geometries are not taken into account. Equidistance is assumed. Later (more recent) cycles are given in a darker shade.

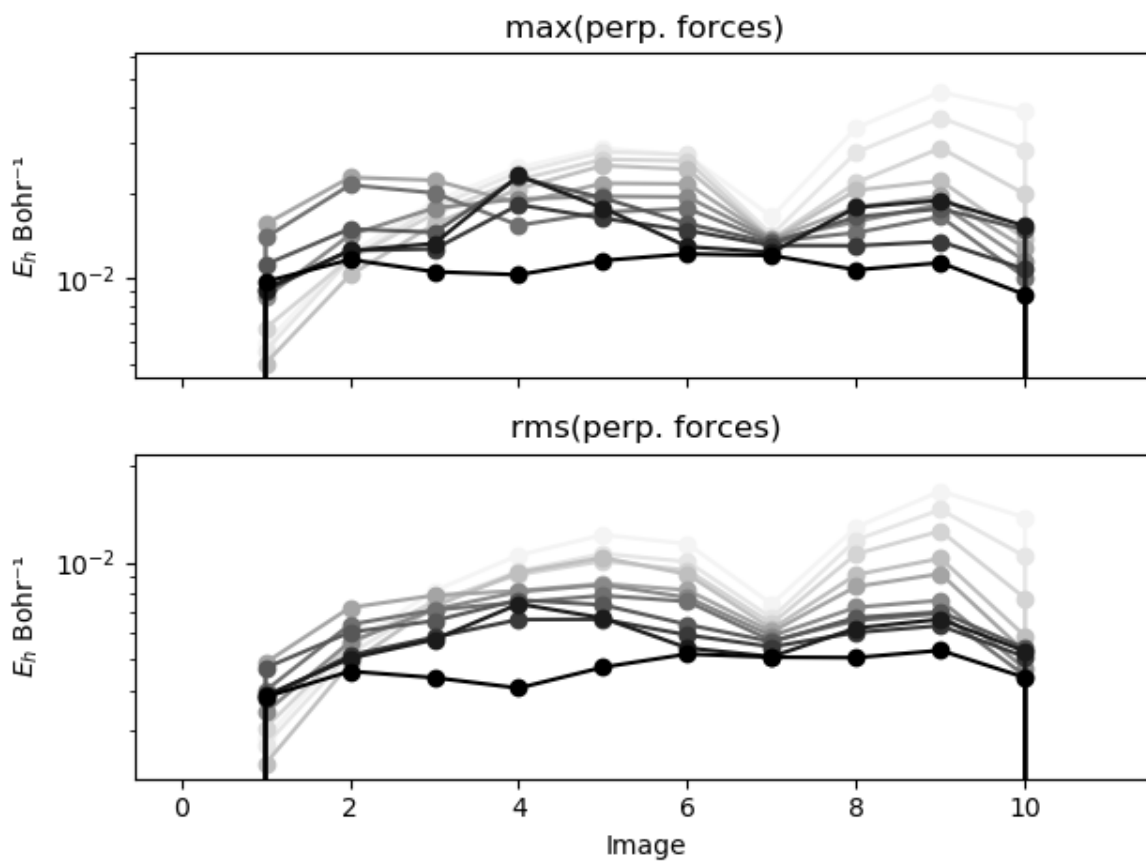


Fig. 15.6: Maximum component and root-mean-square (rms) of the perpendicular component of the force, acting on the COS images.

15.1.3 Plotting TS-optimization progress

The TS-optimization progress is plotted with `pysisplot --opt --h5_group tsopt`. Here we explicitly selected a different HDF5 group by `--h5_group`.

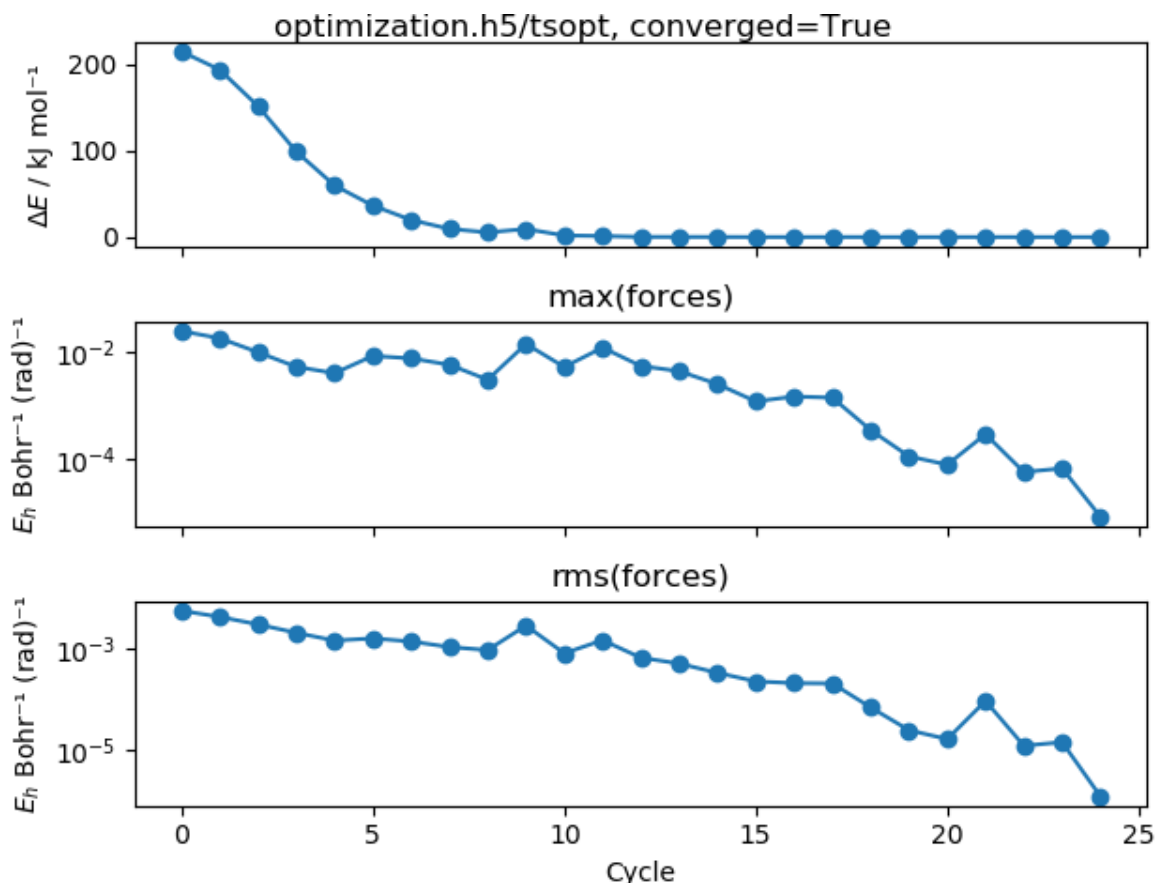


Fig. 15.7: Progress of the TS optimization, started from the HEI.

15.1.4 Plotting the Intrinsic Reaction Coordinate

IRC profiles are easily plotted by

```
pysisplot --irc
```

Multiple plots may appear, depending on the progress of the IRC. The IRC coordinate is given in mass-weighted cartesian coordinates, whereas gradients are given in non-mass-weighted units.

Evidently the IRC integration failed at the end, as can be seen from the the bunched up points, but unless you want to do some kind of transition-state-theory (TST; not supported by pysisyphus) calculations this should not be a problem.

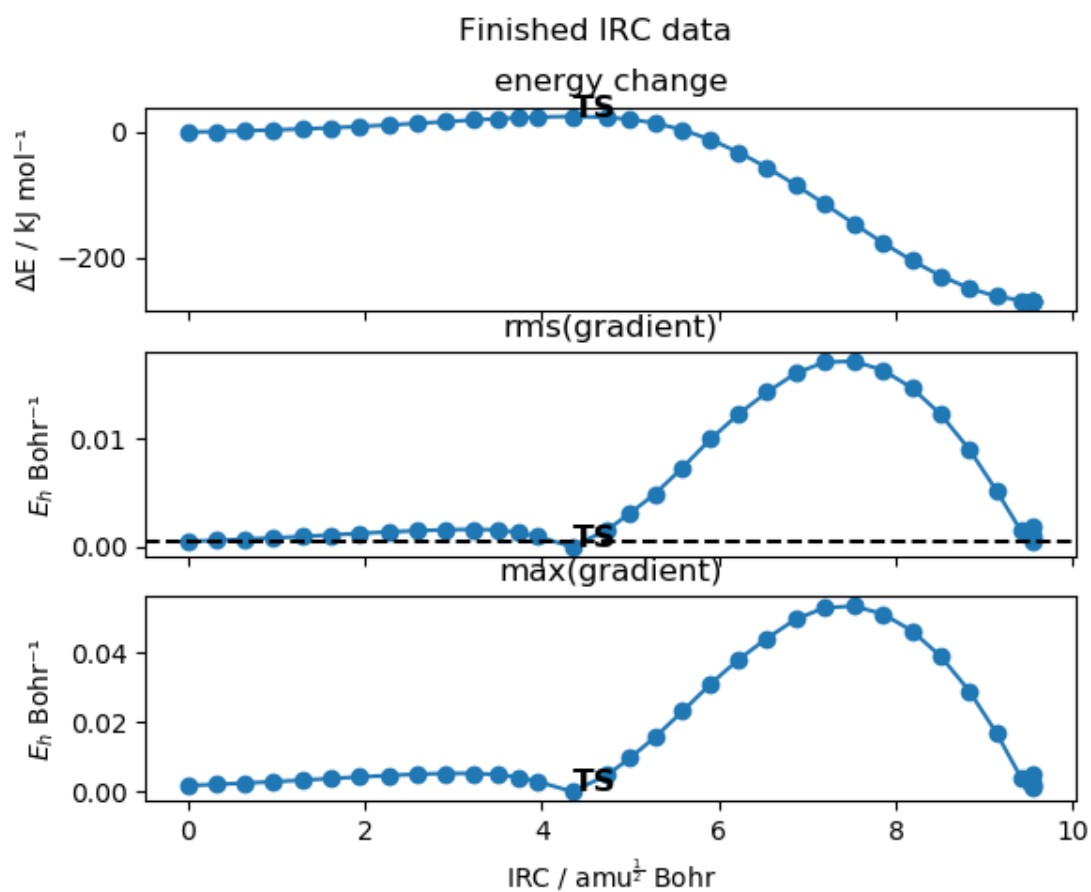


Fig. 15.8: IRC for the Diels-Alder reaction between ethene and 1,4-butadiene.

15.2 Example - AFIR

pysisplot is able to visualize AFIR calculations and to highlight interesting geometries along the optimization. Shown below is an example taken from the [AFIR-Paper](#). By using AFIR the S_N2 between OH^- and fluoromethylene can be forced, yielding methanol and the fluorine anion. The corresponding unit test can be found in the `tests/test_afir` directory of the repository.

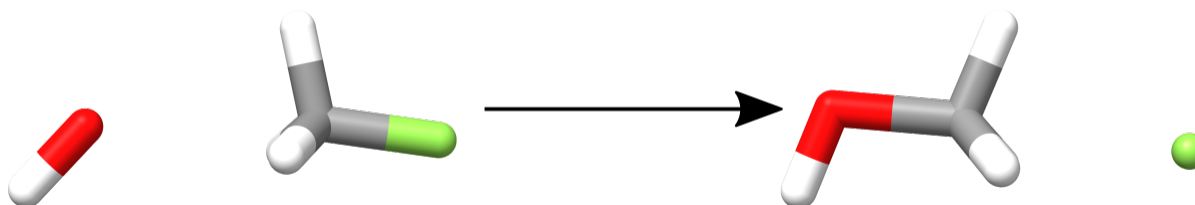


Fig. 15.9: Formation of methanol by means of a S_N2 reaction.

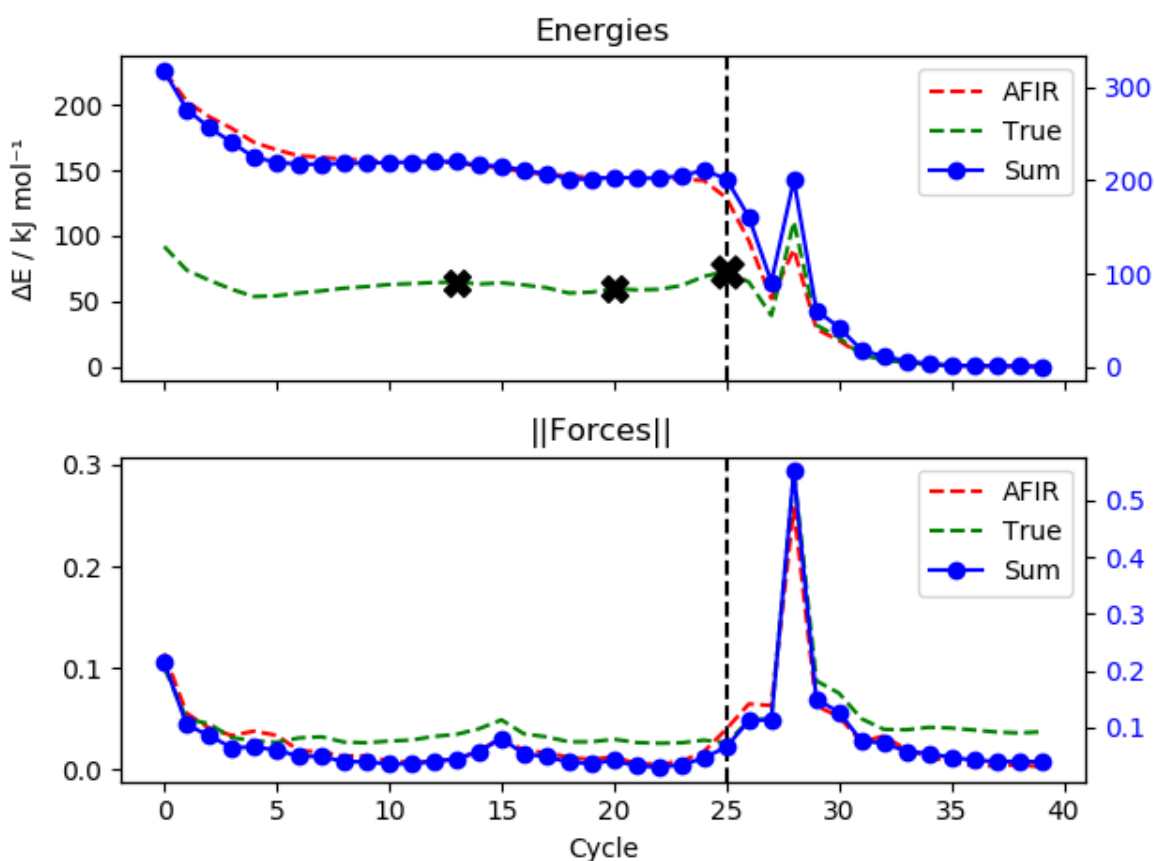


Fig. 15.10: Energy profile and force norms along the S_N2 reaction.

15.3 Example - Excited State Tracking

pysisyphus is aware of excited states (ES) and can track them using various approaches over the course of an optimization or an IRC. By calculating the overlap matrices between ESs at given geometry and a reference geometry pysisyphus can track the desired ES. All relevant data is stored in *overlap_data.h5*.

Optimizing an ES is demonstrated for the S_1 of the 1H-amino-keto tautomer of Cytosin at the PBE0/def-SVP level of theory. A corresponding test can be found under (*tests/test_cytosin_opt*). Right after the first optimization cycle a root flip occurs and the S_1 and S_2 switch. The potential energy curves along the optimization are plotted by:

```
pysisplot --all_energies
pysisplot -a
```

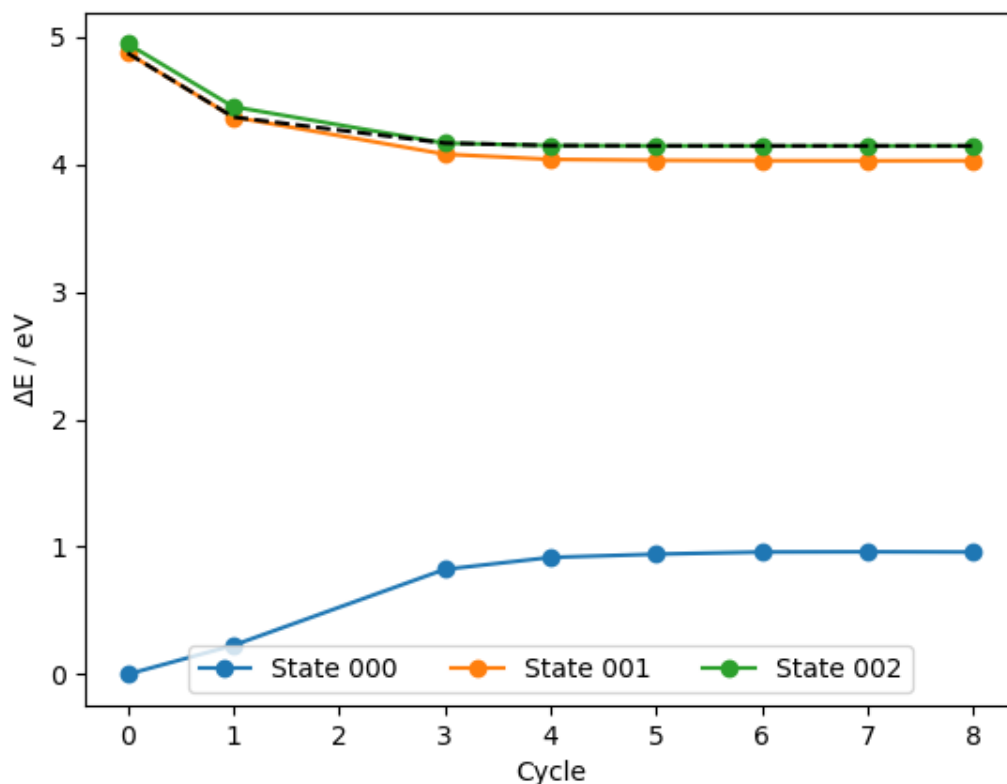


Fig. 15.11: Potential energy curves along the S_1 optimization of the 1H-amino-keto tautomer of Cytosin at the PBE0/def2-SVP level of theory. The root actually followed is indicated by a dashed line.

The calculated overlap matrices can be plotted by:

```
pysisplot --overlaps
pysisplot -o
```

If the calculation was set up to calculate charge-density-differences (CDDs) via MultiWFN and to render them via Jmol then the CDD images displayed beside the overlap matrices.

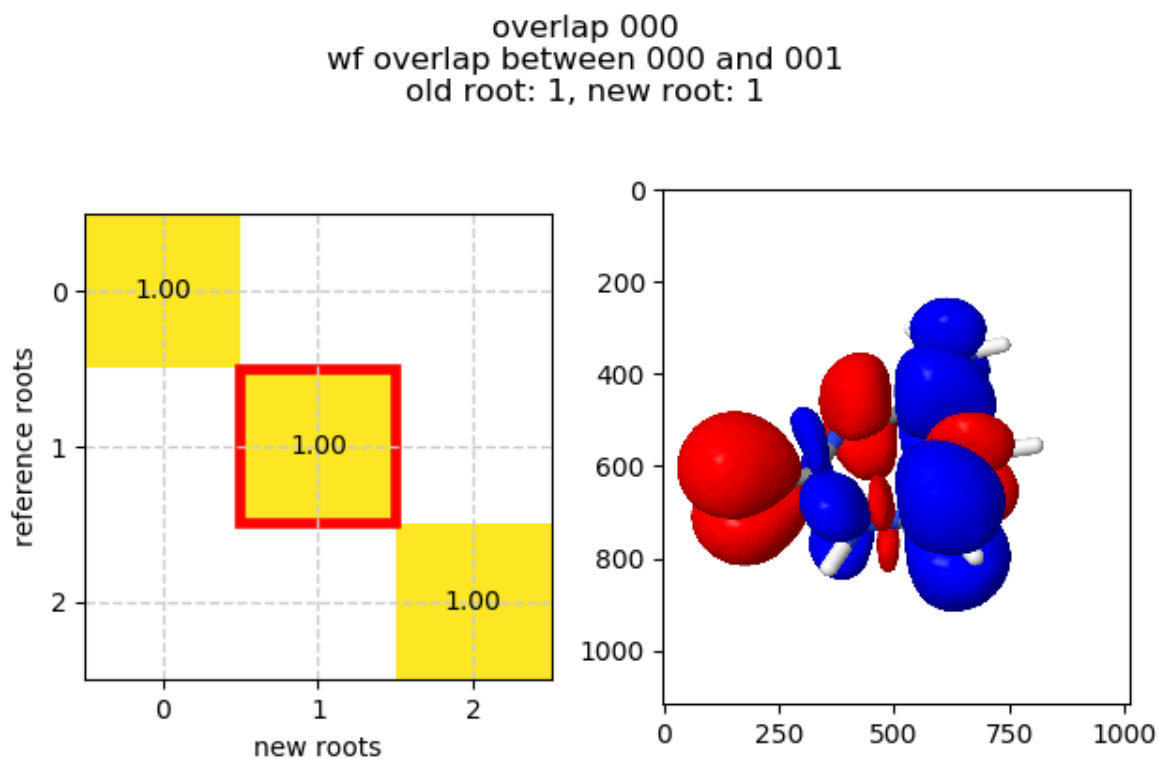


Fig. 15.12: Cytosin S_1 optimization. Overlaps between first and second cycle. No root flips occurred. All ES are in the same order as in the reference geometry at cycle 0.

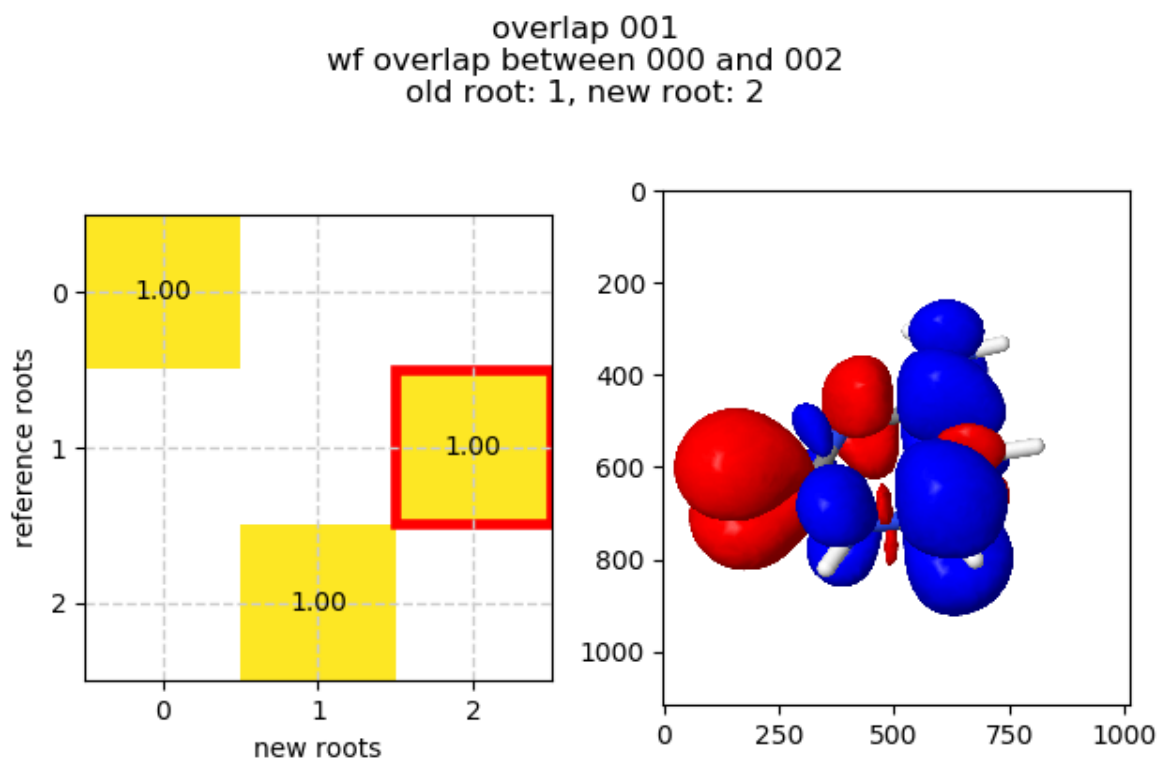


Fig. 15.13: Cytosin S_1 optimization. Overlaps between second and third cycle with root flip. The S_1 and S_2 switch their order.

WORKING WITH GEOMETRIES

TBD

Please see *pysistrj --help* for a list of possible invocations.

```
usage: Utility to transform .xyz and .trj files. [-h]
        (--between BETWEEN | --align | --split
→ | --reverse | --cleantrj | --spline | --first FIRST | --every EVERY | --center | --
→ centerm | --translate TRANSLATE TRANSLATE TRANSLATE | --append | --join | --match | --
→ std | --shake | --internals | --get GET | --origin | --fragsort)
        [--scale SCALE] [--seed SEED]
        [--idpp | --lst | --redund]
        [--noipalign] [--bohr]
        [--noxyz]
        [--atoms ATOMS [ATOMS ...]]
        [--add_prims ADD_PRIMS]
        fns [fns ...]
```

positional arguments:

| | |
|-----|--|
| fns | Filename of .xyz and/or .trj files (xyz and trj can be mixed). |
|-----|--|

optional arguments:

| | |
|---|---|
| -h, --help | show this help message and exit |
| --between BETWEEN | Interpolate additional images. |
| --align | Align geometries onto the first geometry. |
| --split | Split a supplied geometries in multiple .xyz files. |
| --reverse | Reverse a .trj file. |
| --cleantrj | Keep only the first four columns of xyz/trj files. |
| --spline | Evenly redistribute geometries along a splined path. |
| --first FIRST | Copy the first N geometries to a new .trj file. |
| --every EVERY | Create new .trj with every N-th geometry. Always includes the first and last point. |
| --center | Move the molecules centroid into the origin. |
| --centerm | Move the molecules center of mass into the origin. |
| --translate TRANSLATE TRANSLATE TRANSLATE | Translate the molecule by the given vector given in Ångström. |
| --append | Combine the given .xyz files into one .xyz file. |
| --join | Combine the given .xyz/.trj files into one .trj file. |
| --match | Resort the second .xyz file so the atom order matches the first .xyz file. Uses the hungarian method. |

(continues on next page)

(continued from previous page)

```

--std                Move supplied geometry to its standard orientation.
--shake             Shake (randomly displace) coordinates.
--internals         Print automatically generated internals.
--get GET           Get n-th geometry. Expects 0-based index input.
--origin            Translate geometry, so that min(X/Y/Z) == 0.
--fragsort          Resort atoms by fragments.
--idpp              Interpolate using Image Dependent Pair Potential.
--lst               Interpolate by linear synchronous transit.
--redund            Interpolate in internal coordinates.
--noipalign         Don't align geometries when interpolating.
--bohr              Input geometries are in Bohr instead of Angstrom.
--noxyz             Disable dumping of single .xyz files.
--atoms ATOMS [ATOMS ...]
                    Used with --internals. Only print primitives including
                    the given atoms.

--add_prims ADD_PRIMS
                    Used with --internals. Define additional primitives.
                    Expects a string representation of a nested list that
                    can be parsed as YAML e.g.
                    [[10,30],[1,2,3],[4,5,6,7]].

--scale SCALE       Scales the displacement in --shake.
--seed SEED         Initialize the RNG for reproducible results.

```

PYSISYPHUS

17.1 pysisyphus package

17.1.1 Subpackages

17.1.1.1 pysisyphus.benchmarks package

Submodules

pysisyphus.benchmarks.Benchmark module

```
class pysisyphus.benchmarks.Benchmark.Benchmark(name, exclude=None, inv_exclude=False,  
                                                only=None, coord_type='cart', calc_getter=None)
```

Bases: object

```
data_funcs = {'baker': <function get_baker_data>, 'baker_ts': <function  
get_baker_ts_data>, 'birkholz_rx': <function get_birkholz_rx_data>,  
'precon_pos_rot': <function get_precon_pos_rot_data>, 's22': <function  
get_s22_data>, 'xtb_rx': <function get_xtb_rx_data>, 'zimmerman': <function  
get_zimmerman_data>, 'zimmerman_xtb': <function get_zimmerman_xtb_data>}
```

property geom_iter

property geoms

get_geoms(*id_, set_calculator=True*)

pysisyphus.benchmarks.data module

pysisyphus.benchmarks.data.get_baker_data()

10.1002/jcc.540140910

HF/STO-3G

pysisyphus.benchmarks.data.get_baker_ts_data()

10.1002/(SICI)1096-987X(199605)17:7<888::AID-JCC12>3.0.CO;2-7

HF/3-21G

`pysisyphus.benchmarks.data.get_birkholz_rx_data()`

<https://doi.org/10.1002/jcc.23910>

`pysisyphus.benchmarks.data.get_precon_pos_rot_data()`

<https://doi.org/10.1002/jcc.23910>

`pysisyphus.benchmarks.data.get_s22_data()`

<https://doi.org/10.1039/B600027D>

`pysisyphus.benchmarks.data.get_xtb_rx_data()`

`pysisyphus.benchmarks.data.get_zimmerman_data()`

<https://dx.doi.org/10.1021/ct400319w>

`pysisyphus.benchmarks.data.get_zimmerman_xtb_data()`

Reoptimization of

<https://dx.doi.org/10.1021/ct400319w>

at the gfn2-xtb level of theory.

Includes set 1 (first 72 entries), minus (0-based) ids

7, 8, 13, 14, 15, 34, 39

Module contents

17.1.1.2 pysisyphus.calculators package

Submodules

pysisyphus.calculators.AFIR module

```
class pysisyphus.calculators.AFIR.AFIR(calculator, fragment_indices, gamma, rho=1, p=6,
                                         ignore_hydrogen=False, zero_hydrogen=True,
                                         complete_fragments=True, dump=True, h5_fn='afir.h5',
                                         h5_group_name='afir', **kwargs)
```

Bases: *Calculator*

```
__init__(calculator, fragment_indices, gamma, rho=1, p=6, ignore_hydrogen=False, zero_hydrogen=True,
          complete_fragments=True, dump=True, h5_fn='afir.h5', h5_group_name='afir', **kwargs)
```

Artificial Force Induced Reaction calculator.

Currently, there are no automated drivers to run large-scale AFIR calculations with many different initial orientations and/or increasing collision energy parameter. Nonetheless, selected AFIR calculations can be carried out by hand. After convergence, artificial potential & forces, as well as real energies and forces can be plotted with 'pysisplot --afir'. The highest energy point along the AFIR path can then be selected for a subsequent TS-optimization, e.g. via 'pysistrj --get [index] optimization.trj'.

Future versions of pysisyphus may provide drivers for more automated AFIR calculations.

Parameters

- **calculator** (*Calculator*) -- Actual QC calculator that provides energies and its derivatives, that are modified by the AFIR calculator, e.g., ORCA or Psi4.

- **fragment_indices** (List[List[int]]) -- List of lists of integers, specifying the separate fragments. If the indices in these lists don't comprise all atoms in the molecule, the remaining indices will be added as a separate fragment. If a AFIR calculation is carried out with 2 fragments and 'complete_fragments' is True (see below) it is enough to specify only the indices of one fragment, e.g., for a system of 10 atoms 'fragment_indices=[[0,1,2,3]]' is enough. The second system will be set up automatically with indices [4,5,6,7,8,9].
- **gamma** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) -- Collision energy parameter in au. For 2 fragments it can be a single integer, while for > 2 fragments a list of gammas must be given, specifying the pair-wise collision energy parameters. For 3 fragments 3 gammas must be given [_01, _02, _12], for 4 fragments 6 gammas would be required [_01, _02, _03, _12, _13, _23] and so on.
- **rho** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]], default: 1) -- Direction of the artificial force, either 1 or -1. The same comments as for gamma apply. For 2 fragments a single integer is enough, for > 2 fragments a list of rhos must be given (see above). For rho=1 fragments are pushed together, for rho=-1 fragments are pulled apart.
- **p** (int, default: 6) -- Exponent p used in the calculation of the weight function. Defaults to 6 and probably does not have to be changed.
- **ignore_hydrogen** (bool, default: False) -- Whether hydrogens are ignored in the calculation of the artificial force. All weights between atom pairs containing hydrogen will be set to 0.
- **zero_hydrogen** (bool, default: True) -- Whether to use 0.0 as covalent radius for hydrogen in the weight function. Compared to 'ignore_hydrogen', which results in zero weights for all atom pairs involving hydrogen, 'zero_hydrogen' may be non-zero, depending on the covalent radius of the second atom in the pair.
- **complete_fragments** (bool, default: True) -- Whether an incomplete specification in 'fragment_indices' is automatically completed.
- **dump** (bool, default: True) -- Whether an HDF5 file is created.
- **h5_fn** (str, default: 'afir.h5') -- Filename of the HDF5 file used for dumping.
- **h5_group_name** (str, default: 'afir') -- HDF5 group name used for dumping.
- ****kwargs** -- Keyword arguments passed to the Calculator baseclass.

afir_fd_hessian_wrapper(*coords3d*, *afir_grad_func*)

property charge

dump_h5(*atoms*, *coords*, *results*)

get_energy(*atoms*, *coords*, ****prepare_kwargs**)

Meant to be extended.

get_forces(*atoms*, *coords*, ****prepare_kwargs**)

Meant to be extended.

get_hessian(*atoms*, *coords*, ****prepare_kwargs**)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

init_h5_group(*atoms*, *max_cycles=None*)

log_fragments()

property *mult*

set_atoms_and_funcs(*atoms*, *coords*)

Initially *atoms* was also an argument to the constructor of AFIR. I removed it so creation becomes easier. The first time a calculation is requested with a proper atom set everything is set up (cov. radii, afir function and corresponding gradient). Afterwards there is only a check if *atoms* != None and it is expected that all functions are properly set.

fragment_indices can also be incomplete w.r.t. to the number of atoms. If the sum of the specified fragment atoms is less than the number of atoms present then all remaining unspecified atoms will be gathered in one fragment.

write_fragment_geoms(*atoms*, *coords*)

exception `pysisyphus.calculators.AFIR.CovRadiiSumZero`

Bases: `Exception`

`pysisyphus.calculators.AFIR.afir_closure`(*fragment_indices*, *cov_radii*, *gamma*, *rho=1*, *p=6*,
prefactor=1.0, *logger=None*)

rho=1 pushes fragments together, *rho=-1* pulls fragments apart.

`pysisyphus.calculators.AFIR.get_data_model`(*atoms*, *max_cycles*)

pysisyphus.calculators.AnaPot module

class `pysisyphus.calculators.AnaPot.AnaPot`(***kwargs*)

Bases: `AnaPotBase`

pysisyphus.calculators.AnaPot2 module

class `pysisyphus.calculators.AnaPot2.AnaPot2`

Bases: `AnaPotBase`

We can't use sympify as it replaces $1/\tan$ by \cot and this isn't supported by numpy when we call `lambdify`.

class `pysisyphus.calculators.AnaPot2.AnaPot2_`

Bases: `Calculator`

get_energy(*atoms*, *coords*)

Meant to be extended.

pysisyphus.calculators.AnaPot3 module

class pysisyphus.calculators.AnaPot3.**AnaPot3**

Bases: [AnaPotBase](#)

pysisyphus.calculators.AnaPot4 module

class pysisyphus.calculators.AnaPot4.**AnaPot4**

Bases: [AnaPotBase](#)

pysisyphus.calculators.AnaPotBase module

class pysisyphus.calculators.AnaPotBase.**AnaPotBase**(*V_str*, *scale*=1.0, *xlim*=(-1, 1), *ylim*=(-1, 1),
levels=None, *use_sympify*=True, *minima*=None,
saddles=None)

Bases: [Calculator](#)

anim_coords(*coords*, *interval*=50, *show*=False, *title_func*=None)

anim_opt(*opt*, *energy_profile*=False, *colorbar*=False, *figsize*=(8, 6), *show*=False)

get_energy(*atoms*, *coords*)

Meant to be extended.

get_forces(*atoms*, *coords*)

Meant to be extended.

classmethod **get_geom**(*coords*, *atoms*=('X',), *V_str*=None, *calc_kwargs*=None, *geom_kwargs*=None)

get_geoms_from_stored_coords(*coords_list*, *i*=None, *calc_kwargs*=None, *geom_kwargs*=None)

get_hessian(*atoms*, *coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

get_minima(*i*=None, *calc_kwargs*=None, *geom_kwargs*=None)

get_path(*num*, *minima_inds*=None)

get_saddles(*i*=None, *calc_kwargs*=None, *geom_kwargs*=None)

plot(*levels*=None, *show*=False, ***figkwargs*)

plot3d(*levels*=None, *show*=False, *zlim*=None, *vmin*=None, *vmax*=None, *resolution*=100, *rcount*=50,
ccount=50, *nan_above*=None, *init_view*=None, *colorbar*=False, ***figkwargs*)

plot_coords(*xs*, *ys*, *enum*=True, *show*=False, *title*=None)

plot_eigenvalue_structure(*grid*=50, *levels*=None, *show*=False)

plot_geoms(*geoms*, ***kwargs*)

plot_irc(*irc*, **args*, ***kwargs*)

```
plot_opt(opt, *args, **kwargs)
```

```
statistics()
```

pysisyphus.calculators.AnaPotCBM module

```
class pysisyphus.calculators.AnaPotCBM.AnaPotCBM
```

Bases: [AnaPotBase](#)

pysisyphus.calculators.AtomAtomTransTorque module

```
class pysisyphus.calculators.AtomAtomTransTorque.AtomAtomTransTorque(geom, frags, A_mats,  
                                                                    kappa=2.0)
```

Bases: object

```
__init__(geom, frags, A_mats, kappa=2.0)
```

Atom-atom translational and torque forces.

See A.5. [1], Eq. (A6).

```
get_forces(atoms, coords)
```

pysisyphus.calculators.Calculator module

```
class pysisyphus.calculators.Calculator.Calculator(calc_number=0, charge=0, mult=1,  
                                                    base_name='calculator', pal=1, mem=1000,  
                                                    keep_kind='all', check_mem=True, retry_calc=0,  
                                                    last_calc_cycle=None, clean_after=True,  
                                                    out_dir='qm_calcs', force_num_hess=False,  
                                                    num_hess_kwargs=None)
```

Bases: object

```
__init__(calc_number=0, charge=0, mult=1, base_name='calculator', pal=1, mem=1000, keep_kind='all',  
        check_mem=True, retry_calc=0, last_calc_cycle=None, clean_after=True, out_dir='qm_calcs',  
        force_num_hess=False, num_hess_kwargs=None)
```

Base-class of all calculators.

Meant to be extended.

Parameters

- **calc_number** (*int, default=0*) -- Identifier of the Calculator. Used in distinguishing it from other Calculators, e.g. in ChainOfStates calculations. Also used in the creation of filenames.
- **charge** (*int, default=0*) -- Molecular charge.
- **mult** (*int, default=1*) -- Molecular multiplicity (1 = singlet, 2 = doublet, ...)
- **base_name** (*str, default=calculator*) -- Generated filenames will start with this string.
- **pal** (*int, default=1*) -- Positive integer that gives the number of physical cores to use on 1 node.

- **mem** (*int*, *default=1000*) -- Memory per core in MB. The total amount of memory is given as `mem*pal`.
- **check_mem** (*bool*, *default=True*) -- Whether to adjust the requested memory if too much is requested.
- **retry_calc** (*int*, *default=0*) -- Number of additional retries when calculation failed.
- **last_calc_cycle** (*int*) -- Internal variable used in restarts.
- **clean_after** (*bool*) -- Delete temporary directory were calculations were executed after a calculation.
- **out_dir** (*str*) -- Path that is prepended to generated filenames.
- **force_hess_kwargs** (*bool*, *default False*) -- Force numerical Hessians.
- **num_hess_kwargs** (*dict*) -- Keyword arguments for finite difference Hessian calculation.

apply_keep_kind()

apply_set_plans(*kept_fns*, *set_plans=None*)

build_set_plans(*_set_plans=None*)

clean(*path*)

Delete the temporary directory.

Parameters

path (*Path*) -- Directory to delete.

conf_key = `None`

force_num_hessian()

Always calculate numerical Hessians.

classmethod geom_from_fn(*fn*, ***kwargs*)

get_cmd(*key='cmd'*)

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

get_num_hessian(*atoms*, *coords*, ***prepare_kwargs*)

get_restart_info()

Return a dict containing chkfiles.

Returns

restart_info -- Dictionary holding the calculator state. Used for restoring calculators in restarted calculations.

Return type

dict

keep(*path*)

Backup calculation results.

Parameters

path (*Path*) -- Temporary directory of the calculation.

Returns

kept_fns -- Dictionary holding the filenames that were backed up. The keys correspond to the type of file.

Return type

dict

log(*message=""*)

Write a log message.

Wraps the logger variable.

Parameters

message (*str*) -- Message to be logged.

make_fn(*name, counter=None, return_str=False*)

Make a full filename.

Return a full filename including the calculator name and the current counter given a suffix.

Parameters

- **name** (*str*) -- Suffix of the filename.
- **counter** (*int, optional*) -- If not given use the current calc_counter.
- **return_str** (*int, optional*) -- Return a string instead of a Path when True.

Returns

fn -- Filename.

Return type

str

property name

popen(*cmd, cwd=None*)

prepare(*inp*)

Prepare a temporary directory and write input.

Similar to prepare_path, but the input is also written into the prepared directory.

Parameters

inp

[str] Input to be written into the file `self.inp_fn` in the prepared directory.

returns

path: Path

Prepared directory.

prepare_coords(*atoms*, *coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*)

Meant to be extended.

prepare_path(*use_in_run=False*)

Get a temporary directory handle.

Create a temporary directory that can later be used in a calculation.

Parameters

use_in_run (*bool*, *option*) -- Sets the internal variable *self.path_already_prepared* that is later read by *self.run()*. No new temporary directory will be created in *self.run()*.

Returns

path: Path

Prepared directory.

prepare_pattern(*raw_pat*)

Prepare globs.

Transforms an entry of *self.to_keep* into a glob and a key suitable for the use in *self.keep()*.

Parameters

raw_pat (*str*) -- Entry of *self.to_keep*

Returns

- **pattern** (*str*) -- Glob that can be used in *Path.glob()*
- **multi** (*bool*) -- Flag if glob may match multiple files.
- **key** (*str*) -- A key to be used in the *kept_fns* dict.

prepare_turbo_coords(*atoms*, *coords*)

Get a Turbomole coords string.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- String holding coordinates in Turbomole coords format.

Return type

str

prepare_xyz_string(*atoms*, *coords*)

Returns a xyz string in Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

xyz_str -- Coordinates in .xyz format.

Return type

string

print_capabilities()

print_out_fn(*path*)

Print calculation output.

Prints the output of a calculator after a calculation.

Parameters

path (*Path*) -- Temporary directory of the calculation.

restore_org_hessian()

Restore original 'get_hessian' method, which may also fallback to numerical Hessians, if not implemented.

run(*inp*, *calc*, *add_args=None*, *env=None*, *shell=False*, *hold=False*, *keep=True*, *cmd=None*,
inc_counter=True, *run_after=True*, *parser_kwargs=None*, *symlink=True*)

Run a calculation.

The bread-and-butter method to actually run an external quantum chemistry code.

Parameters

- **inp** (*str*) -- Input for the external program that is written to the temp-dir.
- **calc** (*str*, *hashable*) -- Key (and more or less type of calculation) to select the right parsing function from `self.parser_funcs`.
- **add_args** (*iterable*, *optional*) -- Additional arguments that will be appended to the program call.
- **env** (*Environment*, *optional*) -- A potentially modified environment for the subprocess call.
- **shell** (*bool*, *optional*) -- Use a shell to execute the program call. Need for Turbomole were we chain program calls like `dscf; escf`.
- **hold** (*bool*, *optional*) -- Wether to remove the temporary directory after the calculation.
- **keep** (*bool*, *optional*) -- Wether to backup files as specified in `self.to_keep()`. Usually you want this.
- **cmd** (*str or iterable*, *optional*) -- Overwrites `self.base_cmd`.
- **inc_counter** (*bool*, *optional*) -- Wether to increment the counter after a calculation.

Returns

results -- Dictionary holding all applicable results of the calculations like the energy, a forces vector and/or excited state energies from TDDFT.

Return type

dict

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

set_restart_info(*restart_info*)

Sets restart information (chkfiles etc.) on the calculator.

Parameters

restart_info (*dict*) -- Dictionary holding the calculator state. Used for restoring calculators in restarted calculations.

verify_chkfiles(*chkfiles*)

Checks if given chkfiles exist and return them as Paths

Parameters

chkfiles (*dict*) -- Dictionary holding the chkfiles. The keys correspond to the attribute names, the values are strs holding the (potentially full) filename (path).

Returns

paths -- Dictionary of Paths.

Return type

dict

```
class pysisyphus.calculators.Calculator.HessKind(value)
```

Bases: Enum

An enumeration.

NUMERICAL = 2

ORG = 1

```
class pysisyphus.calculators.Calculator.KeepKind(value)
```

Bases: Enum

An enumeration.

ALL = 1

LATEST = 2

NONE = 3

```
class pysisyphus.calculators.Calculator.SetPlan(key, name=None, condition=<function
SetPlan.<lambda>>, fail=None)
```

Bases: object

condition()

fail: Optional[Callable] = None

key: str

name: Optional[str] = None

pysisyphus.calculators.CerjanMiller module

```
class pysisyphus.calculators.CerjanMiller.CerjanMiller(a=1, b=1, c=1)
```

Bases: *AnaPotBase*

pysisyphus.calculators.Composite module

```
class pysisyphus.calculators.Composite.Composite(final, keys_calcs=None, calcs=None,
                                                  remove_translation=False, **kwargs)
```

Bases: *Calculator*

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_final_energy(energies)
```

```
get_forces(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
run_calculation(atoms, coords, **prepare_kwargs)
```

pysisyphus.calculators.ConicalIntersection module

```
class pysisyphus.calculators.ConicalIntersection.CIQuantities(energy1, gradient1, energy2,
                                                                gradient2, energy_diff,
                                                                gradient_diff, gradient_mean, P, x,
                                                                y, energy, forces)
```

Bases: object

P: ndarray

energy: float

energy1: float

energy2: float

energy_diff: float

forces: ndarray

gradient1: ndarray

gradient2: ndarray

gradient_diff: ndarray

gradient_mean: ndarray

x: ndarray

y: ndarray

class pysisyphus.calculators.ConicalIntersection.**ConicalIntersection**(*calculator1*, *calculator2*,
***kwargs*)

Bases: [Calculator](#)

Calculator for conical intersection optimization.

Based on [1].

get_ci_quantities(*atoms*, *coords*, ***prepare_kwargs*)

Relevant quantities including branching plane and projector P.

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Energy of calculator 1.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Projected gradient for CI optimization.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Projected Hessian.

pysisyphus.calculators.ConicalIntersection.**get_P**(*x*, *y*)

Projector to project out components in branching plane.

pysisyphus.calculators.ConicalIntersection.**update_y**(*x*, *x_prev*, *y_prev*)

Update approximate coupling derivative vector y.

pysisyphus.calculators.DFTBp module

class pysisyphus.calculators.DFTBp.**DFTBp**(*parameter*, **args*, *slakos*=None, *root*=None, ***kwargs*)

Bases: [OverlapCalculator](#)

conf_key = 'dftbp'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

static get_excited_state_str(*root*, *forces*=False)

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

static get_gen_str(*atoms*, *coords*)

hubbard_derivs = {'3ob': {'Br': -0.0573, 'C': -0.1492, 'Ca': -0.034, 'Cl':
-0.0697, 'F': -0.1623, 'H': -0.1857, 'I': -0.0433, 'K': -0.0339, 'Mg': -0.02, 'N':
-0.1535, 'Na': -0.0454, 'O': -0.1575, 'P': -0.14, 'S': -0.11, 'Zn': -0.03}}

max_ang_moms = {'3ob': {'Br': 'd', 'C': 'p', 'Ca': 'p', 'Cl': 'd', 'F': 'p',
'H': 's', 'I': 'd', 'K': 'p', 'Mg': 'p', 'N': 'p', 'Na': 'p', 'O': 'p', 'P': 'd',
'S': 'd', 'Zn': 'd'}, 'mio-ext': {'C': 'p', 'H': 's', 'N': 'p', 'O': 'p'}}

parse_all_energies(*out_fn*=None, *exc_dat*=None)

parse_energy(*path*)

static `parse_exc_dat(text)`

parse_forces(*path*)

parse_total_energy(*text*)

prepare_input(*atoms, coords, calc_type*)

Meant to be extended.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

run_calculation(*atoms, coords, **prepare_kwargs*)

store_and_track(*results, func, atoms, coords, **prepare_kwargs*)

`pysisyphus.calculators.DFTBp.parse_mo(eigvec)`

`pysisyphus.calculators.DFTBp.parse_xplusy(text)`

pysisyphus.calculators.Dalton module

class `pysisyphus.calculators.Dalton.Dalton(basis, method='hf', **kwargs)`

Bases: `Calculator`

compute(*mol, prop*)

conf_key = 'dalton'

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

prepare_input(*atoms, coords*)

Meant to be extended.

pysisyphus.calculators.Dimer module

class `pysisyphus.calculators.Dimer.Dimer(calculator, *args, N_raw=None, length=0.0189, rotation_max_cycles=15, rotation_method='fourier', rotation_thresh=0.0001, rotation_tol=1, rotation_max_element=0.001, rotation_interpolate=True, rotation_disable=False, rotation_disable_pos_curv=True, rotation_remove_trans=True, trans_force_f_perp=True, bonds=None, N_hessian=None, bias_rotation=False, bias_translation=False, bias_gaussian_dot=0.1, seed=None, write_orientations=True, forward_hessian=True, **kwargs)`

Bases: `Calculator`

property `C`

Shortcut for the curvature.

property *N*

add_gaussian(*atoms*, *center*, *N*, *height*=0.1, *std*=0.0529, *max_cycles*=50, *dot_ref*=None)

property *can_bias_f0*

property *can_bias_f1*

property *coords0*

property *coords1*

curvature(*f1*, *f2*, *N*)

Curvature of the mode represented by the dimer.

direct_rotation(*optimizer*, *prev_step*)

do_dimer_rotations(*rotation_thresh*=None)

property *energy0*

property *f0*

property *f1*

property *f1_bias*

property *f2*

Never calculated explicitly, but estimated from *f0* and *f1*.

fourier_rotation(*optimizer*, *prev_step*)

get_N_raw_from_hessian(*h5_fn*, *root*=0)

get_forces(*atoms*, *coords*)

Meant to be extended.

get_gaussian_energies(*coords*, *sum_*=True)

get_gaussian_forces(*coords*, *sum_*=True)

get_hessian(*atoms*, *coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

remove_translation(*displacement*)

property *rot_force*

rotate_coords1(*rad*, *theta*)

Rotate dimer and produce new *coords1*.

set_N_raw(*coords*)

property *should_bias_f0*

May lead to calculation of *f0* and/or *f1* if present!

property *should_bias_f1*

May lead to calculation of *f0* and/or *f1* if present!

update_orientation(*coords*)

class pysisyphus.calculators.Dimer.**Gaussian**(*height, center, std, N*)

Bases: object

energy(*R, height=None*)

forces(*R, height=None*)

exception pysisyphus.calculators.Dimer.**RotationConverged**

Bases: Exception

pysisyphus.calculators.Dummy module

class pysisyphus.calculators.Dummy.**Dummy**(*calc_number=0, charge=0, mult=1, base_name='calculator',
pal=1, mem=1000, keep_kind='all', check_mem=True,
retry_calc=0, last_calc_cycle=None, clean_after=True,
out_dir='qm_calcs', force_num_hess=False,
num_hess_kwargs=None*)

Bases: [Calculator](#)

get_energy(*args, **kwargs)

Meant to be extended.

get_forces(*args, **kwargs)

Meant to be extended.

get_hessian(*args, **kwargs)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

raise_exception()

run_calculation(*args, **kwargs)

pysisyphus.calculators.EGO module

class pysisyphus.calculators.EGO.**EGO**(*calculator, ref_geom, max_force=0.175, **kwargs*)

Bases: [Calculator](#)

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_mods(*atoms, coords*)

property **ref_hessian**

property **s**

pysisyphus.calculators.EnergyMin module

```
class pysisyphus.calculators.EnergyMin.EnergyMin(calculator1, calculator2, mix=False, alpha=0.02,
                                                sigma=3.5, min_energy_diff=0.0, check_after=0,
                                                **kwargs)
```

Bases: [Calculator](#)

```
__init__(calculator1, calculator2, mix=False, alpha=0.02, sigma=3.5, min_energy_diff=0.0,
        check_after=0, **kwargs)
```

Use energy and derivatives of the calculator with lower energy.

This calculators carries out two calculations with different settings and returns the results of the lower energy one. This can be used to consider flips between a singlet and a triplet PES etc.

Parameters

- **calculator1** ([Calculator](#)) -- Wrapped QC calculator that provides energies and its derivatives.
- **calculator2** ([Calculator](#)) -- Wrapped QC calculator that provides energies and its derivatives.
- **mix** (bool, default: False) -- Enable mixing of both forces, according to the approach outlined in [2]. Can be used to optimize guesses for MECPs. Pass
- **alpha** (float, default: 0.02) -- Smoothing parameter in Hartree. See [2] for a discussion.
- **sigma** (float, default: 3.5) -- Unitless gap size parameter. The final gap becomes smaller for bigger sigmas. Has to be adapted for each case. See [2] for a discussion (p. 407 right column and p. 408 left column.)
- **min_energy_diff** (float, default: 0.0) -- Energy difference in Hartree. When set to a value != 0 and the energy difference between both calculators drops below this value, execution of both calculations is disabled for 'check_after' cycles. In these cycles the calculator choice remains fixed. After 'check_after' cycles, both energies will be calculated and it is checked, if the previous calculator choice remains valid. In conjunction with 'check_after' both arguments can be used to save computational resources.
- **check_after** (int, default: 0) -- Amount of cycles in which the calculator choice remains fixed.
- ****kwargs** -- Keyword arguments passed to the Calculator baseclass.

```
do_calculations(name, atoms, coords, **prepare_kwargs)
```

```
get_chkfiles()
```

Return type
dict

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
set_chkfiles(chkfiles)
```

pysisyphus.calculators.ExternalPotential module

```
class pysisyphus.calculators.ExternalPotential.ExternalPotential(calculator=None,
                                                                  potentials=None, geom=None,
                                                                  **kwargs)
```

Bases: *Calculator*

```
available_potentials = {'d3': <class 'pysisyphus.calculators.DFTD3.DFTD3'>,
                        'harmonic_sphere': <class
'pysisyphus.calculators.ExternalPotential.HarmonicSphere'>, 'logfermi': <class
'pysisyphus.calculators.ExternalPotential.LogFermi'>, 'restraint': <class
'pysisyphus.calculators.ExternalPotential.Restraint'>, 'rmsd': <class
'pysisyphus.calculators.ExternalPotential.RMSD'>}
```

```
get_energy(atoms, coords)
```

Meant to be extended.

```
get_forces(atoms, coords)
```

Meant to be extended.

```
get_hessian(atoms, coords)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

```
get_potential_energy(coords)
```

```
get_potential_forces(coords)
```

```
class pysisyphus.calculators.ExternalPotential.HarmonicSphere(k, radius, origin=(0.0, 0.0, 0.0),
                                                             geom=None)
```

Bases: object

```
calc(coords3d, gradient=False)
```

```
instant_pressure(coords3d)
```

```
property surface_area
```

In Bohr**2

```
class pysisyphus.calculators.ExternalPotential.LogFermi(beta, radius, T=300, origin=(0.0, 0.0, 0.0),
                                                         geom=None)
```

Bases: object

```
__init__(beta, radius, T=300, origin=(0.0, 0.0, 0.0), geom=None)
```

As described in the XTB docs.

<https://xtb-docs.readthedocs.io/en/latest/xcontrol.html#confining-in-a-cavity>

```
calc(coords3d, gradient=False)
```

```
class pysisyphus.calculators.ExternalPotential.RMSD(geom, k, beta=0.5, atom_indices=None)
```

Bases: object

__init__(*geom*, *k*, *beta*=0.5, *atom_indices*=None)

Restrain based on RMSD with a reference geometry.

As described in <https://doi.org/10.1021/acs.jctc.0c01306>, Eq. (5).

Parameters

- **geom** (*Geometry*) -- Reference geometry for RMSD calculation.
- **k** (float) -- Gaussian height in units of energy. Should be a negative number if the system under study should stay close to the reference geometry (pulling k). A positive Gaussian height k results in forces that push the system under study away from the reference geometry (pushing k).
- **b** -- Gaussian width in inverse units of lengths.
- **atom_indices** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]], None], default: None) -- Optional, numpy array or iterable of integer atom indices. Restricts the RMSD calculation to these atoms. If omitted, all atoms are used.

calc(*coords3d*, *gradient*=False)

class pysisyphus.calculators.ExternalPotential.**Restraint**(*restraints*, *geom*=None)

Bases: object

calc(*coords3d*, *gradient*=False)

static calc_prim_restraint(*prim*, *coords3d*, *force_const*, *ref_val*)

pysisyphus.calculators.FakeASE module

class pysisyphus.calculators.FakeASE.**FakeASE**(*calc*)

Bases: object

get_atoms_coords(*atoms*)

get_forces(*atoms*=None)

get_potential_energy(*atoms*=None)

pysisyphus.calculators.FourWellAnaPot module

class pysisyphus.calculators.FourWellAnaPot.**FourWellAnaPot**

Bases: *AnaPotBase*

pysisyphus.calculators.FreeEndNEBPot module

```
class pysisyphus.calculators.FreeEndNEBPot.FreeEndNEBPot
```

Bases: *AnaPotBase*

```
__init__()
```

Analytical potential as described in [1] Appendix A

pysisyphus.calculators.Gaussian09 module

```
class pysisyphus.calculators.Gaussian09.Gaussian09(*args, **kwargs)
```

Bases: *Gaussian16*

```
conf_key = 'gaussian09'
```

pysisyphus.calculators.Gaussian16 module

```
class pysisyphus.calculators.Gaussian16.Gaussian16(route, gbs="", gen="", keep_chk=False, stable="",
                                                    fchk=None, **kwargs)
```

Bases: *OverlapCalculator*

```
conf_key = 'gaussian16'
```

```
get_chkfiles()
```

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

```
make_exc_str()
```

```
make_fchk(path)
```

```
make_gbs_str()
```

```
parse_635r_dump(dump_path, roots, nmos)
```

```
parse_all_energies(fchk=None)
```

```
parse_charges(path=None)
```

```
parse_double_mol(path, out_fn=None)
```

```
static parse_fchk(fchk_path, keys)
```

```
parse_force(path)
```

```
parse_hessian(path)
```

parse_keyword(*text*)

parse_log(**args*, ***kwargs*)

parse_stable(*path*)

parse_tddft(*path*)

prepare_input(*atoms*, *coords*, *calc_type*, *did_stable=False*, *point_charges=None*)

Meant to be extended.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reuse_data(*path*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like formchk or ricctools.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_double_mol_calculation(*atoms*, *coords1*, *coords2*)

run_rwfdump(*path*, *rwf_index*, *chk_path=None*)

run_stable(*atoms*, *coords*, ***prepare_kwargs*)

set_chkfiles(*chkfiles*)

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

pysisyphus.calculators.HardSphere module

class pysisyphus.calculators.HardSphere.**HardSphere**(*geom*, *frags*, *kappa=1.0*, *permutations=False*, *frag_radii=None*, *radii_offset=0.9452*)

Bases: object

__init__(*geom*, *frags*, *kappa=1.0*, *permutations=False*, *frag_radii=None*, *radii_offset=0.9452*)

Intra-Image Inter-Molecular Hard-Sphere force.

See A.2. in [1], Eq. (A1).

get_forces(*atoms*, *coords*, *kappa=None*)

class pysisyphus.calculators.HardSphere.**PWHardSphere**(*geom*, *frags*, *sub_frgs*, *kappa=1.0*)

Bases: object

__init__(*geom*, *frags*, *sub_frgs*, *kappa=1.0*)

Inter-Molecular pairwise Hard-Sphere forces between atoms.

Hardsphere forces are only applied between certain atoms of given fragments, but the whole fragment is moved. Can be used to remove atom inter-molecular atom clashes.

get_forces(*atoms*, *coords*, *kappa=None*)

pysisyphus.calculators.IDPPCalculator module

class pysisyphus.calculators.IDPPCalculator.**IDPPCalculator**(*target*)

Bases: [Calculator](#)

get_forces(*atoms*, *coords*)

Meant to be extended.

pysisyphus.calculators.IPIClient module

pysisyphus.calculators.IPIClient.**calc_ipi_client**(*addr*, *atoms*, *calc*, *queue=None*, ***kwargs*)

pysisyphus.calculators.IPIClient.**ipi_client**(*addr*, *atoms*, *energy_getter*, *forces_getter*,
hessian_getter=None, *hdrlen=12*)

pysisyphus.calculators.IPIServer module

class pysisyphus.calculators.IPIServer.**IPIServer**(**args*, *address=None*, *host=None*, *port=None*,
unlink=True, *hdrlen=12*, *max_retries=0*,
verbose=False, ***kwargs*)

Bases: [Calculator](#)

get_energy(*atoms*, *coords*)

Meant to be extended.

get_forces(*atoms*, *coords*)

Meant to be extended.

get_hessian(*atoms*, *coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

listen_for(*atoms*, *coords*, *kind='forces'*)

listen_for_client_atom_num(*atom_num*)

listen_for_energy()

listen_for_forces(*atom_num*)

listen_for_hessian(*atom_num*)

listen_kinds = ('coords', 'energy', 'forces', 'hessian')

reset_client_connection()

retried_listen_for(*atoms*, *coords*)

unlink(*address*)

pysisyphus.calculators.LEPSBase module

class pysisyphus.calculators.LEPSBase.LEPSBase(*pot_type='leps'*)

Bases: *AnaPotBase*

pysisyphus.calculators.LEPSExpr module

class pysisyphus.calculators.LEPSExpr.LEPSExpr

Bases: object

G(*a, b*)

Gaussian function.

Gdimer(*x, y, A, x0, y0, sx, sy*)

J(*d, alpha, r0, r*)

Quantum mechanical exchange interaction.

Q(*d, alpha, r0, r*)

Coulomb interactions.

V_LEPS(*x=None, y=None, abc=None*)

Equation (A.1) in [1]. Mimics reaction involving three atoms confined to motion along a line.

V_dimer()

III. Results Section A in [3]. Two additional saddle points from two added gaussians.

V_harmonic()

Equation (A.2) in [1]. A and C are fixed, only B can move. A condensed phase environment is represented by adding a harmonic oscillator degree of freedom.

V_tot()

Equation (A.3) in [1]. Additional saddle point.

__init__()

Generates sympy expression for various LEPS potentials.

get_dimer()

get_expr(*pot_type='leps'*)

get_harmonic()

get_leps()

get_tot()

pysisyphus.calculators.LennardJones module

```
class pysisyphus.calculators.LennardJones.LennardJones(sigma=1.8897261251, epsilon=1, rc=None)
    Bases: Calculator
    calculate(coords3d)
    get_energy(atoms, coords)
        Meant to be extended.
    get_forces(atoms, coords)
        Meant to be extended.
```

pysisyphus.calculators.MOPAC module

```
class pysisyphus.calculators.MOPAC.MOPAC(method='PM7', **kwargs)
    Bases: Calculator
    http://openmopac.net/manual/
    CALC_TYPES = {'energy': '1SCF', 'gradient': '1SCF GRADIENTS', 'hessian': 'DFORCE
    FORCE LET'}
    METHODS = ['am1', 'pm3', 'pm6', 'pm6-dh2', 'pm6-d3', 'pm6-dh+', 'pm6-dh2',
    'pm6-dh2x', 'pm6-d3h4', 'pm6-d3h4x', 'pm7', 'pm7-ts']
    MULT_STRS = {1: 'SINGLET', 2: 'DOUBLET', 3: 'TRIPLET', 4: 'QUARTET', 5:
    'QUINTET', 6: 'SEXTET', 7: 'SEPTET', 8: 'OCTET'}
    base_cmd
        Do only SCF AUX: Creates a checkpoint file NOREO: Dont reorient geometry
        Type
        1SCF
    conf_key = 'mopac'
    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.
    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.
    get_hessian(atoms, coords, **prepare_kwargs)
        Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
        Preferrably, this method should provide an analytical Hessian.
    parse_energy(path)
    static parse_energy_from_aux(inp, *args, **kwargs)
    parse_grad(path)
    parse_hessian(path)
    static parse_hessian_from_aux(inp, *args, **kwargs)
```

prepare_coords(*atoms*, *coords*, *opt=False*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*, *opt=False*)

Meant to be extended.

read_aux(*path*)

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

pysisyphus.calculators.MullerBrownSympyPot module

class pysisyphus.calculators.MullerBrownSympyPot.**MullerBrownPot**

Bases: *AnaPotBase*

pysisyphus.calculators.MultiCalc module

class pysisyphus.calculators.MultiCalc.**MultiCalc**(*calcs*, ***kwargs*)

Bases: *Calculator*

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

pysisyphus.calculators.MultiCalc.**calcs_from_dict**(*calc_dict*, *base_name*, *calc_number*, *charge*, *mult*, *pal*, *mem*)

pysisyphus.calculators.OBabel module

class pysisyphus.calculators.OBabel.**OBabel**(*ff='gaff'*, *mol=None*, ***kwargs*)

Bases: *Calculator*

conv_dict = {'kcal/mol': 627.5094740630558, 'kJ/mol': 2625.4996394798254}

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

setup(*atoms*, *coords*)

pysisyphus.calculators.ONIOMv2 module

```
class pysisyphus.calculators.ONIOMv2.LayerCalc(models, total_size, parent_layer_calc=None)
```

Bases: object

property charge

do_parent(with_parent)

static energy_from_results(model_energies, parent_energy=None)

get_energy(atoms, coords, with_parent=True)

get_forces(atoms, coords, with_parent=True)

get_hessian(atoms, coords, with_parent=True)

property mult

run_calculations(atoms, coords, method)

```
class pysisyphus.calculators.ONIOMv2.Link(ind, parent_ind, atom, g)
```

Bases: tuple

atom

Alias for field number 2

g

Alias for field number 3

ind

Alias for field number 0

parent_ind

Alias for field number 1

```
class pysisyphus.calculators.ONIOMv2.Model(name, calc_level, calc, parent_name, parent_calc_level,  
                                           parent_calc, atom_inds, parent_atom_inds,  
                                           use_link_atoms=True)
```

Bases: object

as_calculator(cap=False)

as_geom(all_atoms, all_coords)

capped_atoms_coords(all_atoms, all_coords)

create_bond_vec_getters(atoms)

create_links(atoms, coords, debug=False)

get_energy(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_forces(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_hessian(atoms, coords, point_charges=None, parent_correction=True, cap=True)

get_jacobian()

`get_sparse_jacobian()`

`log(message="")`

`parse_charges()`

`class pysisyphus.calculators.ONIOMv2.ModelDummyCalc(model, cap=False)`

Bases: `object`

`get_energy(atoms, coords)`

`get_forces(atoms, coords)`

`class pysisyphus.calculators.ONIOMv2.ONIOM(calcs, models, geom, layers=None, embedding="",
real_key='real', use_link_atoms=True, *args, **kwargs)`

Bases: `Calculator`

`__init__(calcs, models, geom, layers=None, embedding="", real_key='real', use_link_atoms=True, *args,
**kwargs)`

layer: list of models

`len(layer) == 1`: normal ONIOM, `len(layer) >= 1`: multicenter ONIOM.

model:

(sub)set of all atoms that resides in a certain layer and has a certain calculator.

`atom_inds_in_layer(index, exclude_inner=False)`

Returns list of atom indices in layer at index.

Atoms that also appear in inner layer can be excluded on request.

Parameters

- **index** (`int`) -- pasd
- **exclude_inner** (`bool`, `default=False`, `optional`) -- Whether to exclude atom indices that also appear in inner layers.

Returns

atom_indices -- List containing the atom indices in the selected layer.

Return type

list

`calc_layer(atoms, coords, index, parent_correction=True)`

property charge

```
embeddings = {'': '', 'electronic': 'Electronic embedding', 'electronic_rc':
'Electronic embedding with redistributed charges', 'electronic_rcd': 'Electronic
embedding with redistributed charges and dipoles'}
```

`get_energy(atoms, coords)`

Meant to be extended.

`get_forces(atoms, coords)`

Meant to be extended.

`get_hessian(atoms, coords)`

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
get_layer_calc(layer_ind)
```

```
property model_iter
```

```
property mult
```

```
run_calculation(atoms, coords)
```

```
run_calculations(atoms, coords, method)
```

```
pysisyphus.calculators.ONIOMv2.atom_inds_to_cart_inds(atom_inds)
```

```
pysisyphus.calculators.ONIOMv2.cap_fragment(atoms, coords, fragment, link_atom='H', g=None)
```

```
pysisyphus.calculators.ONIOMv2.get_embedding_charges(embedding, layer, parent_layer, coords3d)
```

```
pysisyphus.calculators.ONIOMv2.get_g_value(atom, parent_atom, link_atom)
```

pysisyphus.calculators.ORCA module

```
class pysisyphus.calculators.ORCA.ORCA(keywords, blocks="", gbw=None, do_stable=False,
                                         numfreq=False, json_dump=True, **kwargs)
```

Bases: [OverlapCalculator](#)

```
__init__(keywords, blocks="", gbw=None, do_stable=False, numfreq=False, json_dump=True, **kwargs)
```

ORCA calculator.

Wrapper for creating ORCA input files for energy, gradient and Hessian calculations. The PAL and memory inputs must not be given in the keywords and/or blocks, as they are handled by the 'pal' and 'memory' arguments.

Parameters

- **keywords** (*str*) -- Keyword line, as normally given in ORCA, excluding the leading "!".
- **blocks** (*str*, *optional*) -- ORCA block input(s), e.g. for TD-DFT calculations (%tddft ... end). As the blocks start with a leading "%", wrapping the input in quotes (") is required, otherwise the parsing will fail.
- **gbw** (*str*, *optional*) -- Path to an input gbw file, which will be used as initial guess for the first calculation. Will be overridden later, with the path to the gbw file of a previous calculation.
- **do_stable** (*bool*, *optional*) -- Run stability analysis until a stable wavefunction is obtained, before every calculation.
- **numfreq** (*bool*, *optional*) -- Use numerical frequencies instead of analytical ones.
- **json_dump** (*bool*, *optional*) -- Whether to dump the wavefunction to JSON via orca_2json. The JSON can become very large in calculations comprising many basis functions.

```
check_termination(*args, **kwargs)
```

```
clean_tmp(path)
```

```
conf_key = 'orca'
```

```
get_block_str()
```

get_chkfiles()

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

get_moinp_str(*gbw*)

get_stable_wavefunction(*atoms*, *coords*)

parse_all_energies(*text=None*, *triplets=None*)

static parse_atoms_coords(*inp*, **args*, ***kwargs*)

static parse_cis(*cis*)

Simple wrapper of external function.

Currently, only returns X and Y.

parse_energy(*path*)

parse_engrad(*path*)

static parse_engrad_info(*inp*, **args*, ***kwargs*)

static parse_gbw(*gbw_fn*)

static parse_hess_file(*inp*, **args*, ***kwargs*)

parse_hessian(*path*)

parse_mo_numbers(*out_fn*)

parse_stable(*path*)

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges=None*, *do_stable=False*)

Meant to be extended.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reattach(*last_calc_cycle*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

Basically some kind of dummy method that can be called to execute ORCA with the stored cmd of this calculator.

set_chkfiles(*chkfiles*)

set_mo_coeffs(*mo_coeffs=None*, *gbw=None*)

static set_mo_coeffs_in_gbw(*in_gbw_fn*, *out_gbw_fn*, *mo_coeffs*)

See self.parse_gbw.

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

pysisyphus.calculators.ORCA.**get_exc_ens_fosc**(*wf_fn*, *cis_fn*, *log_fn*)

pysisyphus.calculators.ORCA.**get_name**(*text*)

Return string that comes before first character & offset.

pysisyphus.calculators.ORCA.**make_sym_mat**(*table_block*)

pysisyphus.calculators.ORCA.**parse_orca_cis**(*cis_fn*)

Read binary CI vector file from ORCA.

Loosly based on TheoDORE 1.7.1, Authors: S. Mai, F. Plasser <https://sourceforge.net/p/theodore-qc>

pysisyphus.calculators.ORCA.**parse_orca_gbw**(*gbw_fn*)

Adapted from <https://orcaforum.kofo.mpg.de/viewtopic.php?f=8&t=3299>

The first 5 long int values represent pointers into the file:

Pointer @+0: Internal ORCA data structures Pointer @+8: Geometry Pointer @+16: BasisSet Pointer @+24: Orbitals Pointer @+32: ECP data

pysisyphus.calculators.ORCA.**save_orca_pc_file**(*point_charges*, *pc_fn*, *hardness=None*)

pysisyphus.calculators.ORCA5 module

class pysisyphus.calculators.ORCA5.**ORCA5**(*keywords*, *blocks=""*, *gbw=None*, *do_stable=False*, *numfreq=False*, *json_dump=True*, ***kwargs*)

Bases: *ORCA*

conf_key = 'orca5'

pysisyphus.calculators.OpenMM module

class pysisyphus.calculators.OpenMM.**OpenMM**(*topology*, *params*, ***kwargs*)

Bases: *Calculator*

get_charges()

get_dipole_moment(*coords3d*, *reference=None*, *masses=None*)

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

report_charges()

Total charge of the system.

Adapted from modeller.py _addIons() in OpenMM.

pysisyphus.calculators.OpenMolcas module

```
class pysisyphus.calculators.OpenMolcas.OpenMolcas(basis, inorb, rasscf=None, gateway=None,
                                                mcpdft=None, track=True, **kwargs)
```

Bases: *Calculator*

build_gateway_str()

build_mcpdft_str()

build_rasscf_str()

build_rassi_str()

build_str_from_dict(*dct*)

conf_key = 'openmolcas'

get_energy(*atoms, coords*)

Meant to be extended.

get_forces(*atoms, coords*)

Meant to be extended.

get_pal_env()

get_root()

parse_energies(*text*)

parse_energy(*path*)

parse_gradient(*path*)

parse_rassi_track(*path*)

prepare_coords(*atoms, coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms, coords, calc_type*)

Meant to be extended.

reattach(*last_calc_cycle*)

run_calculation(*atoms, coords, calc_type='energy'*)

pysisyphus.calculators.OverlapCalculator module

```
class pysisyphus.calculators.OverlapCalculator.NTOs(ntos, lambdas)
    Bases: tuple

    lambdas
        Alias for field number 1

    ntos
        Alias for field number 0

class pysisyphus.calculators.OverlapCalculator.OverlapCalculator(*args, track=False,
                                                                    ovlp_type='tden',
                                                                    double_mol=False,
                                                                    ovlp_with='previous',
                                                                    adapt_args=(0.5, 0.3, 0.6),
                                                                    cdds=None, orient="",
                                                                    dump_fn='overlap_data.h5',
                                                                    h5_dump=False,
                                                                    conf_thresh=0.001,
                                                                    mos_ref='cur',
                                                                    mos_renorm=True, **kwargs)

    Bases: Calculator

    H5_MAP = {'Ca': 'Ca_list', 'Cb': 'Cb_list', 'Xa': 'Xa_list', 'Xb': 'Xb_list',
              'Ya': 'Ya_list', 'Yb': 'Yb_list', 'all_energies': 'all_energies_list', 'coords':
              'coords_list', 'ref_roots': 'reference_roots', 'roots': 'roots_list'}

    OVLP_TYPE_VERBOSE = {'nto': 'natural transition orbital overlap', 'nto_org':
                          'original natural transition orbital overlap', 'tden': 'transition density matrix
                          overlap', 'top': 'transition orbital pair overlap', 'wf': 'wavefunction overlap'}

    VALID_CDDS = (None, 'calc', 'render')

    VALID_KEYS = ['wf', 'tden', 'nto', 'nto_org', 'top']

    VALID_XY = ('X', 'X+Y', 'X-Y')

    calc_cdd_cube(root, cycle=-1)

    conf_thresh
        self.dyn_roots = int(dyn_roots) if self.dyn_roots != 0:
            self.dyn_roots = 0 self.log("dyn_roots = 0 is hardcoded right now")

    property data_model

    dump_overlap_data()

    static from_overlap_data(h5_fn, set_wfow=False)

    get_ci_coeffs_for(ind)

    get_h5_group()
```

get_indices(*indices=None*)

A new root is determined by selecting the overlap matrix row corresponding to the reference root and checking for the root with the highest overlap (at the current geometry).

The overlap matrix is usually formed by a double loop like:

```
overlap_matrix = np.empty((ref_states, cur_states))
for i, ref_state in enumerate(ref_states):
```

```
    for j, cur_state in enumerate(cur_states):
```

```
        overlap_matrix[i, j] = make_overlap(ref_state, cur_state)
```

So the reference states run along the rows. That's why the `ref_state` index comes first in the 'indices' tuple.

static get_mo_norms(*C, S_AO*)

get_orbital_matrices(*indices=None, S_AO=None*)

Return MO coefficients and AO overlaps for the given indices.

If not provided, a AO overlap matrix is constructed from one of the MO coefficient matrices (controlled by `self.mos_ref`). Also, if requested one of the two MO coefficient matrices is re-normalized.

get_ref_mos(*C_ref, C_cur*)

static get_sao_from_mo_coeffs(*C*)

Recover AO overlaps from given MO coefficients.

For MOs in the columns of `mo_coeffs`:

$$S_{AO} = C^{\dagger} T C^{\dagger} S_{AO} C = C^{\dagger} T (S_{AO} C)^{\dagger} T = C^{\dagger} C^{\dagger} T S_{AO}^{\dagger} T = C^{\dagger} C^{\dagger} T S_{AO} C = I$$

get_tden_overlaps(*indices=None, S_AO=None*)

get_top_differences(*indices=None, S_AO=None*)

Transition orbital projection.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like `mos`, a MO coefficient array and a CI coefficient array.

render_cdd_cube()

static renorm_mos(*C, S_AO*)

store_overlap_data(*atoms, coords, path=None, overlap_data=None*)

property stored_calculations

track_root(*ovlp_type=None*)

Check if a root flip occurred compared to the previous cycle by calculating the overlap matrix wrt. a reference cycle.

```
pysisyphus.calculators.OverlapCalculator.get_data_model(exc_state_num, occ_a, virt_a, occ_b,
                                                         virt_b, ovlp_type, atoms, max_cycles)
```

pysisyphus.calculators.Psi4 module

```
class pysisyphus.calculators.Psi4.Psi4(method, basis, to_set=None, to_import=None, pcm='iefpcm',
                                       solvent=None, write_fchk=False, **kwargs)
```

Bases: [Calculator](#)

conf_key = 'psi4'

get_energy(atoms, coords, **prepare_kwargs)

Meant to be extended.

get_fchk_str()

get_forces(atoms, coords, **prepare_kwargs)

Meant to be extended.

get_hessian(atoms, coords, **prepare_kwargs)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

parse_energy(path)

parse_grad(path)

parse_hessian(path)

prepare_input(atoms, coords, calc_type)

Meant to be extended.

run_calculation(atoms, coords, **prepare_kwargs)

pysisyphus.calculators.PyPsi4 module

```
class pysisyphus.calculators.PyPsi4.PyPsi4(method, basis, **kwargs)
```

Bases: [Calculator](#)

get_forces(atoms, coords)

Meant to be extended.

pysisyphus.calculators.PySCF module

pysisyphus.calculators.PyXTB module

```
class pysisyphus.calculators.PyXTB.PyXTB(*args, gfn=2, acc=None, verbosity=0, keep_calculator=False,
                                           **kwargs)
```

Bases: [Calculator](#)

get_calculator(atoms, coords)

get_energy(atoms, coords, **prepare_kwargs)

Meant to be extended.

get_forces(atoms, coords, **prepare_kwargs)

Meant to be extended.

pysisyphus.calculators.QCEngine module

pysisyphus.calculators.Rastrigin module

class pysisyphus.calculators.Rastrigin.**Rastrigin**

Bases: [AnaPotBase](#)

<http://www.sfu.ca/~ssurjano/rastr.html>

pysisyphus.calculators.Remote module

class pysisyphus.calculators.Remote.**Remote**(*remote_calc, host, prefix="", **kwargs*)

Bases: [Calculator](#)

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms, coords, **prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

parse_results()

run_calculation(*atoms, coords, run_func='get_energy'*)

pysisyphus.calculators.Rosenbrock module

class pysisyphus.calculators.Rosenbrock.**Rosenbrock**

Bases: [AnaPotBase](#)

pysisyphus.calculators.SocketCalc module

class pysisyphus.calculators.SocketCalc.**SocketCalc**(**args, host='localhost', port=8080, **kwargs*)

Bases: [Calculator](#)

get_energy(*atoms, coords*)

Meant to be extended.

get_forces(*atoms, coords*)

Meant to be extended.

get_hessian(*atoms, coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

listen_for(*atoms, coords, request*)

valid_requests = ('energy', 'forces', 'hessian')

pysisyphus.calculators.TIP3P module

```
class pysisyphus.calculators.TIP3P.TIP3P(rc=9.44863062728914)
    Bases: Calculator
    Transferable Intermolecular Potential 3 Point
    aHOH = 104.52
    calculate(coords3d)
    charges
        coulomb_energy = (multiple of elem. charge * multiple of elem. charge)
            / (distance in bohr) * 1 / (4 * pi * vacuum permittivity)
        coulomb_prefactor converts everything to atomic units and it is ... drum roll ... 1. from scipy.constants
        import value as pcval self.coulomb_prefactor = (1 / (4 * np.pi) * pcval("elementary charge"))**2
            / pcval("Hartree energy") / pcval("Bohr radius") / pcval("vacuum electric permittivity")
    )
    coulomb(coords3d)
    epsilon = 0.0002423919586315716
    get_energy(atoms, coords)
        Meant to be extended.
    get_forces(atoms, coords)
        Meant to be extended.
    qH = 0.417
    qO = -0.834
    rOH = 1.8088458464917874
    sigma = 5.953790025507198
```

pysisyphus.calculators.TransTorque module

```
class pysisyphus.calculators.TransTorque.TransTorque(frag, iter_frag, a_mats, b_mats,
    weight_func=None, skip=True, kappa=1.0, b_coords3d=None, do_trans=True)
    Bases: object
    __init__(frag, iter_frag, a_mats, b_mats, weight_func=None, skip=True, kappa=1.0, b_coords3d=None,
        do_trans=True)
        Translational and torque forces. See A.4. [1], Eqs. (A3) - (A5).
    get_forces(atoms, coords, kappa=None)
    get_forces_naive(atoms, coords, kappa=None)
    set_N_invs()
```

```
pysisyphus.calculators.TransTorque.get_trans_torque_forces(mfrag, a_coords3d, b_coords3d,
                                                           a_mats, b_mats, m, frags, N_inv,
                                                           weight_func=None, skip=True,
                                                           kappa=1, do_trans=True)
```

pysisyphus.calculators.Turbomole module

```
class pysisyphus.calculators.Turbomole.Turbomole(control_path=None, simple_input=None,
                                                  root=None, double_mol_path=None,
                                                  cosmo_kwargs=None, **kwargs)
```

Bases: [OverlapCalculator](#)

```
append_control(to_append, log_msg="", **kwargs)
```

```
conf_key = 'turbomole'
```

```
get_chkfiles()
```

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, cmd=None, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

```
get_pal_env()
```

```
get_ricc2_root(text)
```

```
static make_molden(path)
```

```
parse_all_energies()
```

```
parse_cc2_vectors(ccre)
```

```
parse_ci_coeffs()
```

```
parse_double_mol(path)
```

Parse a double molecule overlap matrix from Turbomole output to be used with WFOWrapper.

```
parse_energy(path)
```

```
parse_force(path)
```

```
parse_gs_energy()
```

Several places are possible: \$subenergy from control file total energy from turbomole.out Final MP2 energy from turbomole.out with ADC(2) Final CC2 energy from turbomole.out with CC(2)

```
parse_hessian(path, fn=None)
```

```
parse_mos()
```

parse_td_vectors(*text*)

For TDA calculations only the X vector is present in the `ciss_a/etc.` file. In TDDFT calculations there are twice as much items compared with TDA. The first half corresponds to (X+Y) and the second half to (X-Y). X can be calculated as $X = ((X+Y)+(X-Y))/2$. Y is then given as $Y = (X+Y)-X$. The normalization can then be checked as

```
np.concatenate((X, Y)).dot(np.concatenate((X, -Y)))
```

and should be 1.

static parse_tddft_tden(*inp*, **args*, ***kwargs*)**prepare_input**(*atoms*, *coords*, *calc_type*, *point_charges=None*)

To rectify this we have to construct the `basecmd` dynamically and construct it ad hoc. We could set a RI flag in the beginning and select the correct scf binary here from it. Then we select the following binary on demand, e.g. `aoforce` or `rdgrad` or `egrad` etc.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like `mos`, a MO coefficient array and a CI coefficient array.

prepare_point_charges(*point_charges*)

```
$point_charges <x> <y> <z> <q>
```

prepare_td(*text*)**run_after**(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)**run_double_mol_calculation**(*atoms*, *coords1*, *coords2*)**set_chkfiles**(*chkfiles*)**set_occ_and_mo_nums**(*text*)**store_and_track**(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)**sub_control**(*pattern*, *repl*, *log_msg=""*, ***kwargs*)

```
pysisyphus.calculators.Turbomole.control_from_simple_input(simple_inp, charge, mult,  
                                                            cosmo_kwargs=None)
```

Create control file from 'simple input'.

See examples/opt/26_turbomole_simple_input/ for an example.

```
pysisyphus.calculators.Turbomole.get_cosmo_data_groups(atoms, epsilon, rsolv=None,  
                                                         dcosmo_rs=None)
```

```
pysisyphus.calculators.Turbomole.index_strs_for_atoms(atoms)
```

```
pysisyphus.calculators.Turbomole.render_data_groups(raw_data_groups)
```

pysisyphus.calculators.WFOWrapper module

```
class pysisyphus.calculators.WFOWrapper.WFOWrapper(occ_mo_num, virt_mo_num, conf_thresh=0.001,
                                                    calc_number=0, out_dir='./', wfow_mem=8000,
                                                    ncore=0, debug=False)
```

Bases: object

ci_coeffs_above_thresh(ci_coeffs, thresh=None)

property conf_thresh

fake_turbo_mos(mo_coeffs)

Create a mos file suitable for TURBOMOLE input. All MO eigenvalues are set to 0.0. There is also a little deviation in the formatting (see turbo_fmt()) but it works ...

generate_all_dets(occ_set1, virt_set1, occ_set2, virt_set2)

Generate all possible single excitation determinant strings from union(occ_mos) to union(virt_mos).

get_from_to_sets(ci_coeffs)

get_gs_line(ci_coeffs_with_gs)

log(message)

logger = <Logger wfoverlap (DEBUG)>

make_det_string(inds)

Return spin adapted strings.

make_dets_header(cic, dets_list)

make_full_dets_list(all_inds, det_strings, ci_coeffs)

matrix_types = {'ortho': 'Orthonormalized overlap matrix', 'ovlp': 'Overlap matrix', 'renorm': 'Renormalized overlap matrix'}

parse_wfoverlap_out(text, type_='ortho')

Returns overlap matrix.

set_from_nested_list(nested)

wf_overlap(cycle1, cycle2, ao_ovlp=None)

pysisyphus.calculators.WFOWrapper2 module

```
class pysisyphus.calculators.WFOWrapper2.WFOWrapper2(overlap_data, calc_number=0,
                                                       conf_thresh=0.0001, out_dir='./')
```

Bases: object

all_overlaps()

ci_coeffs_above_thresh(ci_coeffs, thresh=1e-05)

static fake_turbo_mos(mo_coeffs)

Create a mos file suitable for TURBOMOLE input. All MO eigenvalues are set to 0.0. There is also a little deviation in the formatting (see turbo_fmt()) but it works ...

generate_all_dets(*occ_set1*, *virt_set1*, *occ_set2*, *virt_set2*)
Generate all possible single excitation determinant strings from union(occ_mos) to union(virt_mos).

get_iteration(*ind*)

property last_two_coords

log(*message*)

logger = <Logger wfoverlap (DEBUG)>

make_det_string(*inds*)
Return spin adapted strings.

make_dets_header(*cic*, *dets_list*)

make_full_dets_list(*all_inds*, *det_strings*, *ci_coeffs*)

matrix_types = {'ortho': 'Orthonormalized overlap matrix', 'ovlp': 'Overlap matrix', 'renorm': 'Renormalized overlap matrix'}

parse_wfoverlap_out(*text*, *type_*='ortho')
Returns overlap matrix.

set_data()

set_from_nested_list(*nested*)

wf_overlap(*ind1*=-2, *ind2*=-1, *ao_ovlp*=None)

pysisyphus.calculators.XTB module

class pysisyphus.calculators.XTB.**OptResult**(*opt_geom*, *opt_log*)
Bases: tuple

opt_geom
Alias for field number 0

opt_log
Alias for field number 1

class pysisyphus.calculators.XTB.**XTB**(*gbsa*="", *alpb*="", *gfn*=2, *acc*=1.0, *iterations*=250, *etemp*=None, *retry_etemp*=None, *restart*=False, *topo*=None, *topo_update*=None, *quiet*=False, ***kwargs*)
Bases: [Calculator](#)

__init__(*gbsa*="", *alpb*="", *gfn*=2, *acc*=1.0, *iterations*=250, *etemp*=None, *retry_etemp*=None, *restart*=False, *topo*=None, *topo_update*=None, *quiet*=False, ***kwargs*)
XTB calculator.
Wrapper for running energy, gradient and Hessian calculations by XTB.

Parameters

- **gbsa** (*str*, *optional*) -- Solvent for GBSA calculation, by default no solvent model is used.
- **alpb** (*str*, *optional*) -- Solvent for ALPB calculation, by default no solvent model is used.

- **gfn**(*int or str, must be (0, 1, 2, or "ff")*) -- Hamiltonian for the XTB calculation (GFN0, GFN1, GFN2, or GFNFF).
- **acc**(*float, optional*) -- Accuracy control of the calculation, the lower the tighter several numerical thresholds are chosen.
- **iterations**(*int, optional*) -- The number of iterations in SCC calculation.
- **topo**(*str, optional*) -- Path the a GFNFF-topolgy file. As setting up the topology may take some time for sizable systems, it may be desired to reuse the file.
- **topo_update**(*int*) -- Integer controlling the update interval of the GFNFF topology update. If supplied, the topology will be recreated every N-th calculation.
- **mem**(*int*) -- Mememory per core in MB.
- **quiet**(*bool, optional*) -- Suppress creation of log files.

static check_termination(*inp, *args, **kwargs*)

conf_key = 'xtb'

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms, coords, **prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

get_mdrestart_str(*coords, velocities*)

coords and velocities have to given in au!

get_pal_env()

get_retry_args()

parse_charges(*fn=None*)

parse_charges_from_json(*fn=None*)

parse_energy(*path*)

parse_gradient(*path*)

parse_hessian(*path*)

parse_md(*path*)

parse_opt(*path, keep_log=False*)

parse_topo(*path*)

prepare_add_args(*xcontrol=None*)

prepare_coords(*atoms*, *coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges=None*)

Meant to be extended.

reattach(*last_calc_cycle*)

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_md(*atoms*, *coords*, *t*, *dt*, *velocities=None*, *dump=1*)

Expecting t and dt in fs, even though xtb wants t in ps!

run_opt(*atoms*, *coords*, *keep=True*, *keep_log=False*)

run_topo(*atoms*, *coords*)

write_mdrestart(*path*, *mdrestart_str*)

pysisyphus.calculators.parser module

pysisyphus.calculators.parser.**make_float_class**(***kwargs*)

pysisyphus.calculators.parser.**parse_turbo_ccre0_ascii**(*text*)

pysisyphus.calculators.parser.**parse_turbo_exstates**(*text*)

Parse excitation energies (first blocks) from an exstates file.

pysisyphus.calculators.parser.**parse_turbo_exstates_re**(*text*)

pysisyphus.calculators.parser.**parse_turbo_gradient**(*path*)

pysisyphus.calculators.parser.**parse_turbo_mos**(*text*)

pysisyphus.calculators.parser.**to_float**(*s*, *loc*, *toks*)

Module contents

```
class pysisyphus.calculators.AFIR(calculator, fragment_indices, gamma, rho=1, p=6,
                                  ignore_hydrogen=False, zero_hydrogen=True,
                                  complete_fragments=True, dump=True, h5_fn='afir.h5',
                                  h5_group_name='afir', **kwargs)
```

Bases: [Calculator](#)

```
__init__(calculator, fragment_indices, gamma, rho=1, p=6, ignore_hydrogen=False, zero_hydrogen=True,
          complete_fragments=True, dump=True, h5_fn='afir.h5', h5_group_name='afir', **kwargs)
```

Artificial Force Induced Reaction calculator.

Currently, there are no automated drivers to run large-scale AFIR calculations with many different initial orientations and/or increasing collision energy parameter. Nonetheless, selected AFIR calculations can be carried out by hand. After convergence, artificial potential & forces, as well as real energies and forces can be plotted with 'pysisplot --afir'. The highest energy point along the AFIR path can then be selected for a subsequent TS-optimization, e.g. via 'pysistrj --get [index] optimization.trj'.

Future versions of pysisyphus may provide drivers for more automated AFIR calculations.

Parameters

- **calculator** ([Calculator](#)) -- Actual QC calculator that provides energies and its derivatives, that are modified by the AFIR calculator, e.g., ORCA or Psi4.
- **fragment_indices** (List[List[int]]) -- List of lists of integers, specifying the separate fragments. If the indices in these lists don't comprise all atoms in the molecule, the remaining indices will be added as a separate fragment. If a AFIR calculation is carried out with 2 fragments and 'complete_fragments' is True (see below) it is enough to specify only the indices of one fragment, e.g., for a system of 10 atoms 'fragment_indices=[[0,1,2,3]]' is enough. The second system will be set up automatically with indices [4,5,6,7,8,9].
- **gamma** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) -- Collision energy parameter in au. For 2 fragments it can be a single integer, while for > 2 fragments a list of gammas must be given, specifying the pair-wise collision energy parameters. For 3 fragments 3 gammas must be given [_01, _02, _12], for 4 fragments 6 gammas would be required [_01, _02, _03, _12, _13, _23] and so on.
- **rho** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]], default: 1) -- Direction of the artificial force, either 1 or -1. The same comments as for gamma apply. For 2 fragments a single integer is enough, for > 2 fragments a list of rhos must be given (see above). For rho=1 fragments are pushed together, for rho=-1 fragments are pulled apart.
- **p** (int, default: 6) -- Exponent p used in the calculation of the weight function. Defaults to 6 and probably does not have to be changed.
- **ignore_hydrogen** (bool, default: False) -- Whether hydrogens are ignored in the calculation of the artificial force. All weights between atom pairs containing hydrogen will be set to 0.
- **zero_hydrogen** (bool, default: True) -- Whether to use 0.0 as covalent radius for hydrogen in the weight function. Compared to 'ignore_hydrogen', which results in zero weights for all atom pairs involving hydrogen, 'zero_hydrogen' may be non-zero, depending on the covalent radius of the second atom in the pair.

- **complete_fragments** (bool, default: True) -- Whether an incomplete specification in 'fragment_indices' is automatically completed.
- **dump** (bool, default: True) -- Whether an HDF5 file is created.
- **h5_fn** (str, default: 'afir.h5') -- Filename of the HDF5 file used for dumping.
- **h5_group_name** (str, default: 'afir') -- HDF5 group name used for dumping.
- ****kwargs** -- Keyword arguments passed to the Calculator baseclass.

afir_fd_hessian_wrapper(*coords3d, afir_grad_func*)

property charge

dump_h5(*atoms, coords, results*)

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms, coords, **prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

init_h5_group(*atoms, max_cycles=None*)

log_fragments()

property mult

set_atoms_and_funcs(*atoms, coords*)

Initially atoms was also an argument to the constructor of AFIR. I removed it so creation becomes easier. The first time a calculation is requested with a proper atom set everything is set up (cov. radii, afir function and corresponding gradient). Afterwards there is only a check if atoms != None and it is expected that all functions are properly set.

fragment_indices can also be incomplete w.r.t. to the number of atoms. If the sum of the specified fragment atoms is less than the number of atoms present then all remaining unspecified atoms will be gathered in one fragment.

write_fragment_geoms(*atoms, coords*)

class pysisyphus.calculators.**AtomAtomTransTorque**(*geom, frags, A_mats, kappa=2.0*)

Bases: object

__init__(*geom, frags, A_mats, kappa=2.0*)

Atom-atom translational and torque forces.

See A.5. [1], Eq. (A6).

get_forces(*atoms, coords*)

class pysisyphus.calculators.**CFour**(*cfour_input, wavefunction_dump=True, initden_file=None, **kwargs*)

Bases: [Calculator](#)

__init__(*cfour_input*, *wavefunction_dump*=True, *initden_file*=None, ***kwargs*)

CFOUR calculator.

Wrapper handling CFOUR ground state energy and gradient calculations.

Parameters

- **cfour_input** (*dict*) -- CFOUR keywords and values. Note: "on" must be encapsulated in quotes to avoid being translated to True by YAML.
- **wavefunction_dump** (*bool*, *optional*) -- Whether or not to keep ground state SCF orbitals for each geometry step.
- **initden_file** (*str*, *optional*) -- Path to an input initden file for use as a guess SCF density.

conf_key = 'cfour'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

keep(*path*)

Backup calculation results.

Parameters

path (*Path*) -- Temporary directory of the calculation.

Returns

kept_fns -- Dictionary holding the filenames that were backed up. The keys correspond to the type of file.

Return type

dict

parse_energy(*path*)

parse_gradient(*path*)

prepare(*inp*)

Prepare a temporary directory and write input.

Similar to `prepare_path`, but the input is also written into the prepared directory.

Parameters

inp

[str] Input to be written into the file `self.inp_fn` in the prepared directory.

returns

path: Path

Prepared directory.

prepare_coords(*atoms*, *coords*)

Get 3d coords in Bohr

Reshape internal 1d coords to 3d.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Bohr.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*)

Meant to be extended.

run_calculation(*atoms*, *coords*, *calc_type*, ***prepare_kwargs*)

class pysisyphus.calculators.**Composite**(*final*, *keys_calcs=None*, *calcs=None*, *remove_translation=False*, ***kwargs*)

Bases: [*Calculator*](#)

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_final_energy(*energies*)

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

class pysisyphus.calculators.**ConicalIntersection**(*calculator1*, *calculator2*, ***kwargs*)

Bases: [*Calculator*](#)

Calculator for conical intersection optimization.

Based on [1].

get_ci_quantities(*atoms*, *coords*, ***prepare_kwargs*)

Relevant quantities including branching plane and projector P.

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Energy of calculator 1.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Projected gradient for CI optimization.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Projected Hessian.

```

class pysisyphus.calculators.DFTBp(parameter, *args, slakos=None, root=None, **kwargs)
    Bases: OverlapCalculator
    conf_key = 'dftbp'

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    static get_excited_state_str(root, forces=False)

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    static get_gen_str(atoms, coords)

    hubbard_derivs = {'3ob': {'Br': -0.0573, 'C': -0.1492, 'Ca': -0.034, 'Cl':
-0.0697, 'F': -0.1623, 'H': -0.1857, 'I': -0.0433, 'K': -0.0339, 'Mg': -0.02, 'N':
-0.1535, 'Na': -0.0454, 'O': -0.1575, 'P': -0.14, 'S': -0.11, 'Zn': -0.03}}

    max_ang_moms = {'3ob': {'Br': 'd', 'C': 'p', 'Ca': 'p', 'Cl': 'd', 'F': 'p',
'H': 's', 'I': 'd', 'K': 'p', 'Mg': 'p', 'N': 'p', 'Na': 'p', 'O': 'p', 'P': 'd',
'S': 'd', 'Zn': 'd'}}, 'mio-ext': {'C': 'p', 'H': 's', 'N': 'p', 'O': 'p'}}

    parse_all_energies(out_fn=None, exc_dat=None)

    parse_energy(path)

    static parse_exc_dat(text)

    parse_forces(path)

    parse_total_energy(text)

    prepare_input(atoms, coords, calc_type)
        Meant to be extended.

    prepare_overlap_data(path)
        This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and en-
        ergies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI
        coefficient array.

    run_calculation(atoms, coords, **prepare_kwargs)

    store_and_track(results, func, atoms, coords, **prepare_kwargs)

class pysisyphus.calculators.DFTD4(method=None, damp_params=None, model_params=None,
                                   **kwargs)
    Bases: Calculator

    get_dispersion(atoms, coords, grad)

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_model(atoms, coords)

```

```
class pysisyphus.calculators.Dimer(calculator, *args, N_raw=None, length=0.0189,
                                   rotation_max_cycles=15, rotation_method='fourier',
                                   rotation_thresh=0.0001, rotation_tol=1, rotation_max_element=0.001,
                                   rotation_interpolate=True, rotation_disable=False,
                                   rotation_disable_pos_curv=True, rotation_remove_trans=True,
                                   trans_force_f_perp=True, bonds=None, N_hessian=None,
                                   bias_rotation=False, bias_translation=False, bias_gaussian_dot=0.1,
                                   seed=None, write_orientations=True, forward_hessian=True,
                                   **kwargs)
```

Bases: [Calculator](#)

property C

Shortcut for the curvature.

property N

add_gaussian(atoms, center, N, height=0.1, std=0.0529, max_cycles=50, dot_ref=None)

property can_bias_f0

property can_bias_f1

property coords0

property coords1

curvature(f1, f2, N)

Curvature of the mode represented by the dimer.

direct_rotation(optimizer, prev_step)

do_dimer_rotations(rotation_thresh=None)

property energy0

property f0

property f1

property f1_bias

property f2

Never calculated explicitly, but estimated from f0 and f1.

fourier_rotation(optimizer, prev_step)

get_N_raw_from_hessian(h5_fn, root=0)

get_forces(atoms, coords)

Meant to be extended.

get_gaussian_energies(coords, sum_=True)

get_gaussian_forces(coords, sum_=True)

get_hessian(atoms, coords)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

remove_translation(*displacement*)

property *rot_force*

rotate_coords1(*rad, theta*)

Rotate dimer and produce new coords1.

set_N_raw(*coords*)

property *should_bias_f0*

May lead to calculation of f0 and/or f1 if present!

property *should_bias_f1*

May lead to calculation of f0 and/or f1 if present!

update_orientation(*coords*)

```
class pysisyphus.calculators.Dummy(calc_number=0, charge=0, mult=1, base_name='calculator', pal=1,
                                   mem=1000, keep_kind='all', check_mem=True, retry_calc=0,
                                   last_calc_cycle=None, clean_after=True, out_dir='qm_calcs',
                                   force_num_hess=False, num_hess_kwargs=None)
```

Bases: [Calculator](#)

get_energy(**args, **kwargs*)

Meant to be extended.

get_forces(**args, **kwargs*)

Meant to be extended.

get_hessian(**args, **kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

raise_exception()

run_calculation(**args, **kwargs*)

```
class pysisyphus.calculators.EGO(calculator, ref_geom, max_force=0.175, **kwargs)
```

Bases: [Calculator](#)

get_energy(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_forces(*atoms, coords, **prepare_kwargs*)

Meant to be extended.

get_mods(*atoms, coords*)

property *ref_hessian*

property *s*

```
class pysisyphus.calculators.EnergyMin(calculator1, calculator2, mix=False, alpha=0.02, sigma=3.5,
                                       min_energy_diff=0.0, check_after=0, **kwargs)
```

Bases: [Calculator](#)

```
__init__(calculator1, calculator2, mix=False, alpha=0.02, sigma=3.5, min_energy_diff=0.0,
         check_after=0, **kwargs)
```

Use energy and derivatives of the calculator with lower energy.

This calculators carries out two calculations with different settings and returns the results of the lower energy one. This can be used to consider flips between a singlet and a triplet PES etc.

Parameters

- **calculator1** (*Calculator*) -- Wrapped QC calculator that provides energies and its derivatives.
- **calculator2** (*Calculator*) -- Wrapped QC calculator that provides energies and its derivatives.
- **mix** (bool, default: False) -- Enable mixing of both forces, according to the approach outlined in [2]. Can be used to optimize guesses for MECPs. Pass
- **alpha** (float, default: 0.02) -- Smoothing parameter in Hartree. See [2] for a discussion.
- **sigma** (float, default: 3.5) -- Unitless gap size parameter. The final gap becomes smaller for bigger sigmas. Has to be adapted for each case. See [2] for a discussion (p. 407 right column and p. 408 left column.)
- **min_energy_diff** (float, default: 0.0) -- Energy difference in Hartree. When set to a value != 0 and the energy difference between both calculators drops below this value, execution of both calculations is disabled for 'check_after' cycles. In these cycles the calculator choice remains fixed. After 'check_after' cycles, both energies will be calculated and it is checked, if the previous calculator choice remains valid. In conjunction with 'check_after' both arguments can be used to save computational resources.
- **check_after** (int, default: 0) -- Amount of cycles in which the calculator choice remains fixed.
- ****kwargs** -- Keyword arguments passed to the Calculator baseclass.

```
do_calculations(name, atoms, coords, **prepare_kwargs)
```

```
get_chkfiles()
```

Return type
dict

```
get_energy(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_forces(atoms, coords, **prepare_kwargs)
```

Meant to be extended.

```
get_hessian(atoms, coords, **prepare_kwargs)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

```
set_chkfiles(chkfiles)
```

```
class pysisyphus.calculators.ExternalPotential(calculator=None, potentials=None, geom=None,
                                              **kwargs)
```

Bases: *Calculator*


```
available_potentials = {'d3': <class 'pysisyphus.calculators.DFTD3.DFTD3'>,
'harmonic_sphere': <class
'pysisyphus.calculators.ExternalPotential.HarmonicSphere'>, 'logfermi': <class
'pysisyphus.calculators.ExternalPotential.LogFermi'>, 'restraint': <class
'pysisyphus.calculators.ExternalPotential.Restraint'>, 'rmsd': <class
'pysisyphus.calculators.ExternalPotential.RMSD'>}
```

get_energy(*atoms*, *coords*)
Meant to be extended.

get_forces(*atoms*, *coords*)
Meant to be extended.

get_hessian(*atoms*, *coords*)
Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
Preferably, this method should provide an analytical Hessian.

get_potential_energy(*coords*)

get_potential_forces(*coords*)

class pysisyphus.calculators.**FakeASE**(*calc*)
Bases: object

get_atoms_coords(*atoms*)

get_forces(*atoms=None*)

get_potential_energy(*atoms=None*)

class pysisyphus.calculators.**Gaussian09**(**args*, ***kwargs*)
Bases: [Gaussian16](#)

conf_key = 'gaussian09'

class pysisyphus.calculators.**Gaussian16**(*route*, *gbs*="", *gen*="", *keep_chk*=False, *stable*="", *fchk*=None, ***kwargs*)
Bases: [OverlapCalculator](#)

conf_key = 'gaussian16'

get_chkfiles()

get_energy(*atoms*, *coords*, ***prepare_kwargs*)
Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)
Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)
Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
Preferably, this method should provide an analytical Hessian.

make_exc_str()

make_fchk(*path*)

make_gbs_str()

parse_635r_dump(*dump_path*, *roots*, *nmos*)

parse_all_energies(*fchk=None*)

parse_charges(*path=None*)

parse_double_mol(*path*, *out_fn=None*)

static parse_fchk(*fchk_path*, *keys*)

parse_force(*path*)

parse_hessian(*path*)

parse_keyword(*text*)

parse_log(**args*, ***kwargs*)

parse_stable(*path*)

parse_tddft(*path*)

prepare_input(*atoms*, *coords*, *calc_type*, *did_stable=False*, *point_charges=None*)

Meant to be extended.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reuse_data(*path*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like `formchk` or `ricctools`.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_double_mol_calculation(*atoms*, *coords1*, *coords2*)

run_rwf_dump(*path*, *rwf_index*, *chk_path=None*)

run_stable(*atoms*, *coords*, ***prepare_kwargs*)

set_chkfiles(*chkfiles*)

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

class pysisyphus.calculators.**HardSphere**(*geom*, *frags*, *kappa=1.0*, *permutations=False*, *frag_radii=None*, *radii_offset=0.9452*)

Bases: object

__init__(*geom*, *frags*, *kappa=1.0*, *permutations=False*, *frag_radii=None*, *radii_offset=0.9452*)

Intra-Image Inter-Molecular Hard-Sphere force.

See A.2. in [1], Eq. (A1).

```
get_forces(atoms, coords, kappa=None)
```

```
class pysisyphus.calculators.IPIServer(*args, address=None, host=None, port=None, unlink=True,  
                                         hdrlen=12, max_retries=0, verbose=False, **kwargs)
```

Bases: *Calculator*

```
get_energy(atoms, coords)
```

Meant to be extended.

```
get_forces(atoms, coords)
```

Meant to be extended.

```
get_hessian(atoms, coords)
```

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

```
listen_for(atoms, coords, kind='forces')
```

```
listen_for_client_atom_num(atom_num)
```

```
listen_for_energy()
```

```
listen_for_forces(atom_num)
```

```
listen_for_hessian(atom_num)
```

```
listen_kinds = ('coords', 'energy', 'forces', 'hessian')
```

```
reset_client_connection()
```

```
retried_listen_for(atoms, coords)
```

```
unlink(address)
```

```
class pysisyphus.calculators.LennardJones(sigma=1.8897261251, epsilon=1, rc=None)
```

Bases: *Calculator*

```
calculate(coords3d)
```

```
get_energy(atoms, coords)
```

Meant to be extended.

```
get_forces(atoms, coords)
```

Meant to be extended.

```
class pysisyphus.calculators.MOPAC(method='PM7', **kwargs)
```

Bases: *Calculator*

<http://openmopac.net/manual/>

```
CALC_TYPES = {'energy': '1SCF', 'gradient': '1SCF GRADIENTS', 'hessian': 'DFORCE  
FORCE LET'}
```

```
METHODS = ['am1', 'pm3', 'pm6', 'pm6-dh2', 'pm6-d3', 'pm6-dh+', 'pm6-dh2',  
'pm6-dh2x', 'pm6-d3h4', 'pm6-d3h4x', 'pm7', 'pm7-ts']
```

```
MULT_STRS = {1: 'SINGLET', 2: 'DOUBLET', 3: 'TRIPLET', 4: 'QUARTET', 5:  
'QUINTET', 6: 'SEXTET', 7: 'SEPTET', 8: 'OCTET'}
```

base_cmd

Do only SCF AUX: Creates a checkpoint file NOREO: Dont reorient geometry

Type

1SCF

conf_key = 'mopac'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

parse_energy(*path*)

static parse_energy_from_aux(*inp*, **args*, ***kwargs*)

parse_grad(*path*)

parse_hessian(*path*)

static parse_hessian_from_aux(*inp*, **args*, ***kwargs*)

prepare_coords(*atoms*, *coords*, *opt=False*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*, *opt=False*)

Meant to be extended.

read_aux(*path*)

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

class pysisyphus.calculators.**MultiCalc**(*calcs*, ***kwargs*)

Bases: [Calculator](#)

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

class pysisyphus.calculators.**OBabel**(*ff='gaff'*, *mol=None*, ***kwargs*)

Bases: [Calculator](#)

conv_dict = {'kcal/mol': 627.5094740630558, 'kJ/mol': 2625.4996394798254}

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

setup(*atoms*, *coords*)

class pysisyphus.calculators.**ONIOM**(*calcs*, *models*, *geom*, *layers=None*, *embedding=""*, *real_key='real'*, *use_link_atoms=True*, **args*, ***kwargs*)

Bases: [Calculator](#)

__init__(*calcs*, *models*, *geom*, *layers=None*, *embedding=""*, *real_key='real'*, *use_link_atoms=True*, **args*, ***kwargs*)

layer: list of models

len(layer) == 1: normal ONIOM, len(layer) >= 1: multicenter ONIOM.

model:

(sub)set of all atoms that resides in a certain layer and has a certain calculator.

atom_inds_in_layer(*index*, *exclude_inner=False*)

Returns list of atom indices in layer at index.

Atoms that also appear in inner layer can be excluded on request.

Parameters

- **index** (*int*) -- pasd
- **exclude_inner** (*bool*, *default=False*, *optional*) -- Whether to exclude atom indices that also appear in inner layers.

Returns

atom_indices -- List containing the atom indices in the selected layer.

Return type

list

calc_layer(*atoms*, *coords*, *index*, *parent_correction=True*)

property charge

```
embeddings = {'': '', 'electronic': 'Electronic embedding', 'electronic_rc':
'Electronic embedding with redistributed charges', 'electronic_rcd': 'Electronic
embedding with redistributed charges and dipoles'}
```

get_energy(*atoms*, *coords*)

Meant to be extended.

get_forces(*atoms*, *coords*)

Meant to be extended.

get_hessian(*atoms*, *coords*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferably, this method should provide an analytical Hessian.

get_layer_calc(*layer_ind*)

property model_iter

property `mult`

run_calculation(*atoms*, *coords*)

run_calculations(*atoms*, *coords*, *method*)

class `pysisyphus.calculators.ORCA`(*keywords*, *blocks*="", *gbw*=None, *do_stable*=False, *numfreq*=False, *json_dump*=True, ***kwargs*)

Bases: `OverlapCalculator`

__init__(*keywords*, *blocks*="", *gbw*=None, *do_stable*=False, *numfreq*=False, *json_dump*=True, ***kwargs*)

ORCA calculator.

Wrapper for creating ORCA input files for energy, gradient and Hessian calculations. The PAL and memory inputs must not be given in the keywords and/or blocks, as they are handled by the 'pal' and 'memory' arguments.

Parameters

- **keywords** (*str*) -- Keyword line, as normally given in ORCA, excluding the leading "!".
- **blocks** (*str*, *optional*) -- ORCA block input(s), e.g. for TD-DFT calculations (%tddft ... end). As the blocks start with a leading "%", wrapping the input in quotes (") is required, otherwise the parsing will fail.
- **gbw** (*str*, *optional*) -- Path to an input gbw file, which will be used as initial guess for the first calculation. Will be overridden later, with the path to the gbw file of a previous calculation.
- **do_stable** (*bool*, *optional*) -- Run stability analysis until a stable wavefunction is obtained, before every calculation.
- **numfreq** (*bool*, *optional*) -- Use numerical frequencies instead of analytical ones.
- **json_dump** (*bool*, *optional*) -- Whether to dump the wavefunction to JSON via `orca_2json`. The JSON can become very large in calculations comprising many basis functions.

check_termination(**args*, ***kwargs*)

clean_tmp(*path*)

conf_key = 'orca'

get_block_str()

get_chkfiles()

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

get_moinp_str(*gbw*)

```

get_stable_wavefunction(atoms, coords)

parse_all_energies(text=None, triplets=None)

static parse_atoms_coords(inp, *args, **kwargs)

static parse_cis(cis)
    Simple wrapper of external function.
    Currently, only returns X and Y.

parse_energy(path)

parse_engrad(path)

static parse_engrad_info(inp, *args, **kwargs)

static parse_gbw(gbw_fn)

static parse_hess_file(inp, *args, **kwargs)

parse_hessian(path)

parse_mo_numbers(out_fn)

parse_stable(path)

prepare_input(atoms, coords, calc_type, point_charges=None, do_stable=False)
    Meant to be extended.

prepare_overlap_data(path)
    This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

reattach(last_calc_cycle)

run_after(path)
    Meant to be extended.

    This method is called after a calculation was done, but before entering self.keep() and self.clean(). Can be used to call tools like formchk or ricctools.

run_calculation(atoms, coords, **prepare_kwargs)
    Basically some kind of dummy method that can be called to execute ORCA with the stored cmd of this calculator.

set_chkfiles(chkfiles)

set_mo_coeffs(mo_coeffs=None, gbw=None)

static set_mo_coeffs_in_gbw(in_gbw_fn, out_gbw_fn, mo_coeffs)
    See self.parse_gbw.

store_and_track(results, func, atoms, coords, **prepare_kwargs)

class pysisyphus.calculators.ORCA5(keywords, blocks="", gbw=None, do_stable=False, numfreq=False, json_dump=True, **kwargs)

    Bases: ORCA

```

```
conf_key = 'orca5'
```

```
class pysisyphus.calculators.OpenMolcas(basis, inorb, rasscf=None, gateway=None, mcpdft=None,
                                         track=True, **kwargs)
```

Bases: [Calculator](#)

```
build_gateway_str()
```

```
build_mcpdft_str()
```

```
build_rasscf_str()
```

```
build_rassi_str()
```

```
build_str_from_dict(dct)
```

```
conf_key = 'openmolcas'
```

```
get_energy(atoms, coords)
```

Meant to be extended.

```
get_forces(atoms, coords)
```

Meant to be extended.

```
get_pal_env()
```

```
get_root()
```

```
parse_energies(text)
```

```
parse_energy(path)
```

```
parse_gradient(path)
```

```
parse_rassi_track(path)
```

```
prepare_coords(atoms, coords)
```

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms** (*iterable*) -- Atom descriptors (element symbols).
- **coords** (*np.array, 1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

```
prepare_input(atoms, coords, calc_type)
```

Meant to be extended.

```
reattach(last_calc_cycle)
```

```
run_calculation(atoms, coords, calc_type='energy')
```



```
class pysisyphus.calculators.Psi4(method, basis, to_set=None, to_import=None, pcm='iefpcm',
                                   solvent=None, write_fchk=False, **kwargs)

    Bases: Calculator
    conf_key = 'psi4'

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_fchk_str()

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_hessian(atoms, coords, **prepare_kwargs)
        Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
        Preferably, this method should provide an analytical Hessian.

    parse_energy(path)

    parse_grad(path)

    parse_hessian(path)

    prepare_input(atoms, coords, calc_type)
        Meant to be extended.

    run_calculation(atoms, coords, **prepare_kwargs)

class pysisyphus.calculators.PyPsi4(method, basis, **kwargs)
    Bases: Calculator

    get_forces(atoms, coords)
        Meant to be extended.

class pysisyphus.calculators.PyXTB(*args, gfn=2, acc=None, verbosity=0, keep_calculator=False,
                                   **kwargs)
    Bases: Calculator

    get_calculator(atoms, coords)

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.

class pysisyphus.calculators.Remote(remote_calc, host, prefix="", **kwargs)
    Bases: Calculator

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, **prepare_kwargs)
        Meant to be extended.
```

get_hessian(atoms, coords, **prepare_kwargs)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

parse_results()

run_calculation(atoms, coords, run_func='get_energy')

class pysisyphus.calculators.**TIP3P**(rc=9.44863062728914)

Bases: [Calculator](#)

Transferable Intermolecular Potential 3 Point

aHOH = 104.52

calculate(coords3d)

charges

coulomb_energy = (multiple of elem. charge * multiple of elem. charge)
/ (distance in bohr) * 1 / (4 * pi * vacuum permittivity)

coulomb_prefactor converts everything to atomic units and it is ... drum roll ... 1. from scipy.constants
import value as pcval self.coulomb_prefactor = (1 / (4 * np.pi) * pcval("elementary charge"))**2

/ pcval("Hartree energy") / pcval("Bohr radius") / pcval("vacuum electric permittivity")

)

coulomb(coords3d)

epsilon = 0.0002423919586315716

get_energy(atoms, coords)

Meant to be extended.

get_forces(atoms, coords)

Meant to be extended.

qH = 0.417

qO = -0.834

rOH = 1.8088458464917874

sigma = 5.953790025507198

class pysisyphus.calculators.**TransTorque**(frags, iter_fragments, a_mats, b_mats, weight_func=None,
skip=True, kappa=1.0, b_coords3d=None, do_trans=True)

Bases: object

__init__(frags, iter_fragments, a_mats, b_mats, weight_func=None, skip=True, kappa=1.0, b_coords3d=None,
do_trans=True)

Translational and torque forces. See A.4. [1], Eqs. (A3) - (A5).

get_forces(atoms, coords, kappa=None)

get_forces_naive(atoms, coords, kappa=None)

set_N_invs()

```

class pysisyphus.calculators.Turbomole(control_path=None, simple_input=None, root=None,
                                       double_mol_path=None, cosmo_kwargs=None, **kwargs)

    Bases: OverlapCalculator

    append_control(to_append, log_msg="", **kwargs)

    conf_key = 'turbomole'

    get_chkfiles()

    get_energy(atoms, coords, **prepare_kwargs)
        Meant to be extended.

    get_forces(atoms, coords, cmd=None, **prepare_kwargs)
        Meant to be extended.

    get_hessian(atoms, coords, **prepare_kwargs)
        Get Hessian matrix. Fall back to numerical Hessian, if not overridden.
        Preferably, this method should provide an analytical Hessian.

    get_pal_env()

    get_ricc2_root(text)

    static make_molden(path)

    parse_all_energies()

    parse_cc2_vectors(ccre)

    parse_ci_coeffs()

    parse_double_mol(path)
        Parse a double molecule overlap matrix from Turbomole output to be used with WFOWrapper.

    parse_energy(path)

    parse_force(path)

    parse_gs_energy()
        Several places are possible: $subenergy from control file total energy from turbomole.out Final MP2 energy
        from turbomole.out with ADC(2) Final CC2 energy from turbomole.out with CC(2)

    parse_hessian(path, fn=None)

    parse_mos()

    parse_td_vectors(text)
        For TDA calculations only the X vector is present in the ciss_a/etc. file. In TDDFT calculations there are
        twice as much items compared with TDA. The first half corresponds to (X+Y) and the second half to (X-Y).
        X can be calculated as  $X = ((X+Y)+(X-Y))/2$ . Y is then given as  $Y = (X+Y)-X$ . The normalization can then
        be checked as
        
$$\text{np.concatenate}((X, Y)).\text{dot}(\text{np.concatenate}((X, -Y)))$$

        and should be 1.

    static parse_tddft_tden(inp, *args, **kwargs)

```

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges*=None)

To rectify this we have to construct the bascmd dynamically and construct it ad hoc. We could set a RI flag in the beginning and select the correct scf binary here from it. Then we select the following binary on demand, e.g. aoforce or rdgrad or egrad etc.

prepare_overlap_data(*path*)

This method has to implement the calculator specific parsing of MO-coefficients, CI-coefficients and energies. Should return a filename pointing to TURBOMOLE like mos, a MO coefficient array and a CI coefficient array.

prepare_point_charges(*point_charges*)

\$point_charges <x> <y> <z> <q>

prepare_td(*text*)

run_after(*path*)

Meant to be extended.

This method is called after a calculation was done, but before entering `self.keep()` and `self.clean()`. Can be used to call tools like formchk or ricctools.

run_calculation(*atoms*, *coords*, ***prepare_kwargs*)

run_double_mol_calculation(*atoms*, *coords1*, *coords2*)

set_chkfiles(*chkfiles*)

set_occ_and_mo_nums(*text*)

store_and_track(*results*, *func*, *atoms*, *coords*, ***prepare_kwargs*)

sub_control(*pattern*, *repl*, *log_msg*="", ***kwargs*)

```
class pysisyphus.calculators.XTB(gbsa="", alpb="", gfn=2, acc=1.0, iterations=250, etemp=None,
                                retry_etemp=None, restart=False, topo=None, topo_update=None,
                                quiet=False, **kwargs)
```

Bases: [Calculator](#)

```
__init__(gbsa="", alpb="", gfn=2, acc=1.0, iterations=250, etemp=None, retry_etemp=None, restart=False,
         topo=None, topo_update=None, quiet=False, **kwargs)
```

XTB calculator.

Wrapper for running energy, gradient and Hessian calculations by XTB.

Parameters

- **gbsa** (*str*, *optional*) -- Solvent for GBSA calculation, by default no solvent model is used.
- **alpb** (*str*, *optional*) -- Solvent for ALPB calculation, by default no solvent model is used.
- **gfn** (*int or str*, *must be (0, 1, 2, or "ff")*) -- Hamiltonian for the XTB calculation (GFN0, GFN1, GFN2, or GFNFF).
- **acc** (*float*, *optional*) -- Accuracy control of the calculation, the lower the tighter several numerical thresholds are chosen.
- **iterations** (*int*, *optional*) -- The number of iterations in SCC calculation.

- **topo**(*str*, *optional*) -- Path the a GFNFF-topolgy file. As setting up the topology may take some time for sizable systems, it may be desired to reuse the file.
- **topo_update**(*int*) -- Integer controlling the update interval of the GFNFF topology update. If supplied, the topolgy will be recreated every N-th calculation.
- **mem**(*int*) -- Mememory per core in MB.
- **quiet**(*bool*, *optional*) -- Suppress creation of log files.

static check_termination(*inp*, **args*, ***kwargs*)

conf_key = 'xtb'

get_energy(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_forces(*atoms*, *coords*, ***prepare_kwargs*)

Meant to be extended.

get_hessian(*atoms*, *coords*, ***prepare_kwargs*)

Get Hessian matrix. Fall back to numerical Hessian, if not overridden.

Preferrably, this method should provide an analytical Hessian.

get_mdrestart_str(*coords*, *velocities*)

coords and velocities have to given in au!

get_pal_env()

get_retry_args()

parse_charges(*fn=None*)

parse_charges_from_json(*fn=None*)

parse_energy(*path*)

parse_gradient(*path*)

parse_hessian(*path*)

parse_md(*path*)

parse_opt(*path*, *keep_log=False*)

parse_topo(*path*)

prepare_add_args(*xcontrol=None*)

prepare_coords(*atoms*, *coords*)

Get 3d coords in Angstrom.

Reshape internal 1d coords to 3d and convert to Angstrom.

Parameters

- **atoms**(*iterable*) -- Atom descriptors (element symbols).
- **coords**(*np.array*, *1d*) -- 1D-array holding coordinates in Bohr.

Returns

coords -- 3D-array holding coordinates in Angstrom.

Return type

np.array, 3d

prepare_input(*atoms*, *coords*, *calc_type*, *point_charges=None*)

Meant to be extended.

reattach(*last_calc_cycle*)**run_calculation**(*atoms*, *coords*, ***prepare_kwargs*)**run_md**(*atoms*, *coords*, *t*, *dt*, *velocities=None*, *dump=1*)

Expecting t and dt in fs, even though xtb wants t in ps!

run_opt(*atoms*, *coords*, *keep=True*, *keep_log=False*)**run_topo**(*atoms*, *coords*)**write_mdrestart**(*path*, *mdrestart_str*)

17.1.1.3 pysisyphus.cos package

Submodules

pysisyphus.cos.AdaptiveNEB module

```
class pysisyphus.cos.AdaptiveNEB.AdaptiveNEB(images, adapt=True, adapt_fact=0.25, adapt_between=1,  
                                              scale_fact=False, keep_hei=True, free_ends=True,  
                                              **kwargs)
```

Bases: [NEB](#)

```
__init__(images, adapt=True, adapt_fact=0.25, adapt_between=1, scale_fact=False, keep_hei=True,  
         free_ends=True, **kwargs)
```

(Free-End) Adaptive Nudged Elastic Band.

Parameters

- **images** (*list of Geometry objects*) -- Images of the band.
- **adapt** (*bool, default True*) -- Whether to adapt the image number or not. This switch is included to support the FreeEndNEB class, that is just a thin wrapper around this class.
- **adapt_fact** (*positive float*) -- Factor that is used to decide whether to adapt. The initial threshold is calculated by multiplying the RMS force of the band with this factor. When the RMS of the force falls below this threshold adaption takes place.
- **adapt_between** (*positive integer*) -- Number of images to interpolate between the highest energy image and its neighbours. The number of starting images must be higher or equal than 2*adapt_between+3, as we reuse/transfer the calculators from the starting images onto the new ones.
- **scale_fact** (*bool, default False*) -- Whether to increase adapt_fact in deeper levels. This may lead to earlier adaption.
- **keep_hei** (*bool, optional*) -- Whether to keep the highest energy image (usually a very good idea) or to interpolate only between the neighbouring images.
- **free_ends** (*bool, default True*) -- Whether to use modified forces on the end images.

adapt_this_cycle(*forces*)

Decide whether to adapt.

Parameters

forces (*np.array*) -- Forces of the previous optimization cycle.

Returns

adapt -- Flag that indicates if adaption should take place in this cycle.

Return type

bool

property forces

See Eq. (7) in [2].

prepare_opt_cycle(*args, **kwargs)

Check for adaption and adapt if needed.

See ChainOfStates.prepare_opt_cycle for a complete docstring.

update_adapt_thresh(*forces*)

Update the adaption threshold.

Parameters

forces (*np.array*) -- Forces of the previous optimization cycle.

pysisyphus.cos.ChainOfStates module

```
class pysisyphus.cos.ChainOfStates.ChainOfStates(images, fix_first=True, fix_last=True,
                                                align_fixed=True, climb=False, climb_rms=0.005,
                                                climb_lanczos=False, climb_lanczos_rms=0.005,
                                                climb_fixed=True, energy_min_mix=False,
                                                scheduler=None, progress=False)
```

Bases: object

as_xyz(*comments=None*)

property atoms

calculate_forces()

property calculator

property cart_coords

Return a flat 1d array containing the cartesian coordinates of all images.

check_for_climbing_start(*ref_rms*)

clear()

concurrent_force_calcs(*images_to_calculate*, *image_indices*)

property coords

Return a flat 1d array containing the coordinates of all images.

property coords3d

describe()

property energy

property forces

get_climbing_forces(*ind*)

get_climbing_indices()

get_dask_client()

get_fixed_indices()

get_hei_index(*energies=None*)
Return index of highest energy image.

get_image_calc_counter_sum()

get_perpendicular_forces(*i*)
[1] Eq. 12

get_splined_hei()

get_tangent(*i, kind='upwinding', lanczos_guess=None, disable_lanczos=False*)
[1] Equations (8) - (11)

get_tangents()

property gradient

property image_coords

property image_inds

property last_index

log(*message*)

logger = <Logger cos (DEBUG)>

property masses_rep

property max_image_num

property moving_images

property moving_indices
Returns the indices of the images that aren't fixed and can be optimized.

par_image_calc(*image*)

property perpendicular_forces

prepare_opt_cycle(*last_coords, last_energies, last_forces*)
Implements additional logic in preparation of the next optimization cycle.

Should be called by the optimizer at the beginning of a new optimization cycle. Can be used to implement additional logic as needed for AdaptiveNEB etc.

property results

rms(*arr*)

Root mean square

Returns the root mean square of the given array.

Parameters

arr (*iterable of numbers*)

Returns

rms -- Root mean square of the given array.

Return type

float

set_climbing_forces(*forces*)

set_coords_at(*i, coords*)

Called from helpers.procrustes with cartesian coordinates. Then tries to set cartesian coordinate as self.images[i].coords which will raise an error when coord_type != "cart".

set_images(*indices, images*)

set_vector(*name, vector, clear=False*)

set_zero_forces_for_fixed_images()

This is always done in cartesian coordinates, independent of the actual coord_type of the images as setting forces only work with cartesian forces.

valid_coord_types = ('cart', 'cartesian', 'dlc')

zero_fixed_vector(*vector*)

pysisyphus.cos.FreeEndNEB module

class pysisyphus.cos.FreeEndNEB.**FreeEndNEB**(*args, fix_first=False, fix_last=False, **kwargs)

Bases: [AdaptiveNEB](#)

__init__(*args, fix_first=False, fix_last=False, **kwargs)

Simple Free-End-NEB method.

Derived from AdaptiveNEB with disabled adaptation. Only implements Eq. (7) from [2]. For other implementations please see the commit 01bc8812ca6f1cd3645d43e0337d9e3c5fb0ba55. There the other variants are present but I think Eq. (7) in [2] is the simplest & best bet.

pysisyphus.cos.FreezingString module

class pysisyphus.cos.FreezingString.**FreezingString**(*images, calc_getter, max_nodes=10, opt_steps=3*)

Bases: object

property allcoords

as_xyz()

property coords

property energy
property forces
property fully_grown
get_new_image(*coords*, *index*)
get_tangent()
property left_frontier
reparametrize()
property right_frontier
set_new_frontier_nodes()

pysisyphus.cos.GrowingChainOfStates module

```
class pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates(images, calc_getter,  
                                                                max_nodes=10, **kwargs)
```

Bases: [ChainOfStates](#)

property arc_dims
get_new_image_from_coords(*coords*, *index*)
property max_image_num
new_node_coords(*k*)
prepare_opt_cycle(**args*, ***kwargs*)
 Implements additional logic in preparation of the next optimization cycle.
 Should be called by the optimizer at the beginning of a new optimization cycle. Can be used to implement
 additional logic as needed for AdaptiveNEB etc.
set_new_node(*k*)

pysisyphus.cos.GrowingNT module

```
class pysisyphus.cos.GrowingNT.GrowingNT(geom, step_len=0.5, rms_thresh=0.0017, r=None,  
                                          final_geom=None, between=None, bonds=None,  
                                          r_update=True, r_update_thresh=1.0, stop_after_ts=False,  
                                          require_imag_freq=0.0, hessian_at_ts=False, out_dir='.',  
                                          dump=True)
```

Bases: object

property P
 Projector that keeps perpendicular component.
as_xyz()
property atoms

```
calc_hessian_for(other_geom)

property cart_coords
property cart_forces

check_convergence(*args, **kwargs)

clear_passed()

property coords
property energy
property forces

get_additional_print()

get_energy_and_forces_at(coords)

get_energy_at(coords)

get_path(fn)

static get_r(geom, final_geom, bonds, r)

grow_image()

initialize()

log(message)

logger = <Logger cos (DEBUG)>

property r
    Parallel/search direction.

reparametrize()
    Check if GNT can be grown.
```

pysisyphus.cos.GrowingString module

```
class pysisyphus.cos.GrowingString.GrowingString(images, calc_getter, perp_thresh=0.05,
                                                  param='equi', reparam_every=2,
                                                  reparam_every_full=3, reparam_tol=None,
                                                  reparam_check='rms', max_micro_cycles=5,
                                                  reset_dlc=True, climb=False, **kwargs)
```

Bases: [GrowingChainOfStates](#)

```
property forces

property full_string_image_inds

property fully_grown
    Returns whether the string is fully grown. Don't count the first and last node.

get_additional_print()
```

get_cur_param_density(*kind=None*)

get_new_image(*ref_index*)

Get new image by taking a step from self.images[ref_index] towards the center of the string.

get_tangent(*i*)

[1] Equations (8) - (11)

property image_inds

property left_size

property lf_ind

Index of the left frontier node in self.images.

property nodes_missing

Returns the number of nodes to be grown.

reparam_cart(*desired_param_density*)

reparam_dlc(*desired_param_density, thresh=0.001*)

reparametrize()

reset_geometries(*ref_geometry*)

property rf_ind

Index of the right frontier node in self.images.

property right_size

set_coords(*image, coords*)

spline(*tangents=False*)

property string_size

pysisyphus.cos.NEB module

class pysisyphus.cos.NEB.**NEB**(*images, variable_springs=False, k_max=0.3, k_min=0.1, perp_spring_forces=None, bandwidth=None, **kwargs*)

Bases: [ChainOfStates](#)

fmt_k()

property forces

get_parallel_forces(*i*)

get_quenched_dneb_forces(*i*)

See [3], Sec. VI and [4] Sec. D.

get_spring_forces(*i*)

property parallel_forces

set_variable_springs()

update_springs()

pysisyphus.cos.SimpleZTS module

```
class pysisyphus.cos.SimpleZTS.SimpleZTS(images, param='equal', **kwargs)
    Bases: ChainOfStates
    reparametrize()
```

Module contents

17.1.1.4 pysisyphus.db package

Submodules

pysisyphus.db.generate_db module

```
pysisyphus.db.generate_db.clean()
pysisyphus.db.generate_db.generate_db()
pysisyphus.db.generate_db.parse_args(args)
pysisyphus.db.generate_db.run()
```

pysisyphus.db.helpers module

```
pysisyphus.db.helpers.full_name(molecule_name, level_name)
pysisyphus.db.helpers.get_molecules_levels()
pysisyphus.db.helpers.get_path(molecule_name, level_name)
```

pysisyphus.db.level module

pysisyphus.db.molecules module

```
class pysisyphus.db.molecules.Molecule(name, fn, charge, mult, density, resname)
    Bases: tuple
    charge
        Alias for field number 2
    density
        Alias for field number 4
    fn
        Alias for field number 1
    mult
        Alias for field number 3
```

name

Alias for field number 0

resname

Alias for field number 5

Module contents

17.1.1.5 pysisyphus.drivers package

Submodules

pysisyphus.drivers.afir module

```
class pysisyphus.drivers.afir.AFIRPath(atoms, cart_coords, energies, forces, charge, mult,  
                                         opt_is_converged=None, gamma=None, path_indices=None)
```

Bases: object

atoms: tuple

cart_coords: ndarray

charge: int

compatible(*other*)

dump_trj(*fn*)

energies: ndarray

forces: ndarray

gamma: Optional[float] = None

mult: int

opt_is_converged: Optional[bool] = None

path_indices: Optional[List[int]] = None

pysisyphus.drivers.afir.analyze_afir_path(*energies*)

pysisyphus.drivers.afir.automatic_fragmentation(*atoms, coords3d, frag1, frag2, cycles=2, p=6,*
 bond_factor=1.25)

Automatic fragmentation scheme as described in SC-AFIR paper [1].

pysisyphus.drivers.afir.coordinates_similar(*test_coords3d, ref_coords3d, rmsd_thresh=0.01*)

Return type

Tuple[bool, int]

pysisyphus.drivers.afir.decrease_distance(*coords3d, m, n, frac=0.8*)

`pysisyphus.drivers.afir.determine_target_pairs(atoms, coords3d, min_=1.25, max_=5.0, active_atoms=None)`

Determine possible target m, n atom pairs for SC-AFIR calculations.

Return type

List[Tuple[int]]

`pysisyphus.drivers.afir.determine_target_pairs_for_geom(geom, **kwargs)`

Determine possible target m, n atom pairs for SC-AFIR calculations from geom.

Return type

List[Tuple[int]]

`pysisyphus.drivers.afir.find_candidates(center, bond_sets)`

`pysisyphus.drivers.afir.generate_random_union(geoms, offset=2.0, rng=None)`

Unite fragments into one Geometry with random fragment orientations.

Center, rotate and translate from origin according to approximate radius and an offset. Displace along +x, -x, +y, -y, +z, -z.

Results for > 3 fragments don't look so pretty ;).

`pysisyphus.drivers.afir.generate_random_union_ref(geoms, rng=None, opt_kwargs=None)`

Unite fragments into one Geometry with random fragment orientations.

`pysisyphus.drivers.afir.geom_similar(test_geom, ref_geoms, **kwargs)`

Return type

bool

`pysisyphus.drivers.afir.lstsqsWithReference(coords3d, ref_coords3d, freeze_atoms=None)`

Least-squares w.r.t. reference coordinates while keeping some atoms frozen.

`pysisyphus.drivers.afir.opt_afir_path(geom, calc_getter, afir_kwargs, opt_kwargs=None, out_dir=None)`

Minimize geometry with AFIR calculator.

`pysisyphus.drivers.afir.prepare_mc_afir(geoms, rng=None, **kwargs)`

Wrapper for generate_random_union(_ref).

`pysisyphus.drivers.afir.prepare_sc_afir(geom, m, n, bond_factor=1.2)`

Create perturbed geometry, determine fragments and set AFIR calculator.

`pysisyphus.drivers.afir.relax_afir_path(atoms, cart_coords, calc_getter, images=15, out_dir=None)`

Sample imagef from AFIR path and do COS relaxation.

`pysisyphus.drivers.afir.run_afir_path(geom, calc_getter, out_dir, gamma_max, gamma_interval, rng, ignore_bonds=None, bond_factor=1.2, afir_kwargs=None, opt_kwargs=None)`

Driver for AFIR minimizations with increasing gamma values.

`pysisyphus.drivers.afir.run_afir_paths(afir_key, geoms, calc_getter, afir_kwargs=None, opt_kwargs=None, seed=None, N_sample=None, rmsd_thresh=0.25, **kwargs)`

`pysisyphus.drivers.afir.run_mc_afir_paths(geoms, calc_getter, gamma_max, rng, N_max=5, gamma_interval=(0.0, 1.0), afir_kwargs=None, opt_kwargs=None)`

```
pysisyphus.drivers.afir.run_sc_afir_paths(geom, calc_getter, gamma_max, rng, N_max=5,  
                                          N_sample=0, gamma_interval=(1.0, 1.0), afir_kwargs=None,  
                                          opt_kwargs=None, target_pairs=None)
```

```
pysisyphus.drivers.afir.weight_function(atoms, coords3d, i, j, p=6)
```

pysisyphus.drivers.barriers module

```
pysisyphus.drivers.barriers.do_endopt_ts_barriers(ts_geom, left_geoms, right_geoms=None,  
                                                  left_fns=None, right_fns=None, do_thermo=False,  
                                                  T=298.15, p=101325, calc_getter=None,  
                                                  solv_calc_getter=None,  
                                                  do_standard_state_corr=False)
```

pysisyphus.drivers.birkholz module

```
pysisyphus.drivers.birkholz.birkholz_interpolation(geoms, calc_getter, recreate=True)
```

```
pysisyphus.drivers.birkholz.bond_order(r, r0, b=2)
```

Bond order for given bond length and reference length.

Eq. (3) in [1].

```
pysisyphus.drivers.birkholz.bond_orders(coords3d, bond_indices, r0s)
```

List of bond orders.

```
pysisyphus.drivers.birkholz.bond_orders_for_geom(geom, bond_indices)
```

Wrapper for bond_orders for simple use with Geometry.

```
pysisyphus.drivers.birkholz.get_r0s(geom, bond_indices)
```

Reference bond lengths as sum of covalent radii.

```
pysisyphus.drivers.birkholz.length_for_bond_order(bo, r0, b=2)
```

Return bond length for given bond order and reference length.

Eq. (3) in [1].

pysisyphus.drivers.merge module

```
pysisyphus.drivers.merge.align_on_subset(geom1, union, del=None)
```

Align 'union' onto subset of 'geom1'

```
pysisyphus.drivers.merge.hardsphere_merge(geom1, geom2)
```

```
pysisyphus.drivers.merge.merge_geoms(geom1, geom2, geom1_del=None, geom2_del=None,  
                                      make_bonds=None)
```

Merge geom1 and geom2 while keeping the original coordinates.

Supports deleting certain atoms.

```
pysisyphus.drivers.merge.merge_opt(union, bond_diff, ff='mmff94')
```

Fragment merging along given bond by forcefield optimization.


```
pysisyphus.drivers.merge.merge_with_frozen_geom(frozen_geom, lig_geom, make_bonds, frozen_del,
                                                lig_del, ff='mmff94')
```

```
pysisyphus.drivers.merge.parse_args(args)
```

```
pysisyphus.drivers.merge.prepare_merge(geom1, bond_diff, geom2=None, del1=None, del2=None,
                                       dump=False)
```

```
pysisyphus.drivers.merge.run_merge()
```

pysisyphus.drivers.opt module

```
class pysisyphus.drivers.opt.OptResult(opt, geom, fn)
```

Bases: object

fn: Path

geom: Geometry

opt: Optimizer

```
pysisyphus.drivers.opt.get_optimal_bias(ref_geom, calc_getter, opt_key, opt_kwargs, k_max, k_min=0.0,
                                       rmsd_target=0.188973, rmsd_thresh=None,
                                       rmsd_kwargs=None, k_thresh=0.001, strict=True)
```

Driver to determine optimal bias value k for RMSD restraint.

Parameters

- **ref_geom** (Geometry) -- Reference geometry. Starting point of the optimizations and reference for RMSD calculations.
- **calc_getter** (Callable) -- Function that returns the actual calculator, providing the energy and its derivatives.
- **opt_key** (str) -- Determines optimizer type. See pysisyphus.optimizers.cls_map.
- **opt_kwargs** (Dict) -- Optional dict of arguments passed to the optimizer.
- **k_max** (float) -- Maximum absolute value of bias factor k. Must be a > k_min.
- **k_min** -- Minimum absolute value of bias factor k. Must be a positive number >= 0.0. Defaults to 0.0.
- **rmsd_target** (float, default: 0.188973) -- Target RMSD value in au. Defaults to 0.188973 a0 (approx. 0.1).
- **rmsd_thresh** (Optional[float], default: None) -- Allowed deviation from rmsd_target in au. If omitted, 5% of rmsd_target are used.
- **rmsd_kwargs** (Optional[Dict], default: None) -- Additional keyword arguments that are passed to the RMSD class, e.g., atom_indices.
- **k_thresh** (float, default: 0.001) -- When the absolute value of k_bias - k_min or k_max becomes smaller than k_thresh, the bisection is aborted.
- **strict** (default: True) -- If True, AssertionError is raised when an optimization did not converged.

Return type

Tuple[OptResult, float, bool]

Returns

- *opt_result* -- OptimizationResult object containig the Optimizer object.
- *k_opt* -- Optimal value of *k_bias*.
- *valid_k* -- Whether an appropriate bias value *k* was found.

```
pysisyphus.drivers.opt.opt_davidson(opt, topt=True, res_rms_thresh=0.0001)
```

```
pysisyphus.drivers.opt.run_opt(geom, calc_getter, opt_key, opt_kwargs=None, iterative=False,  
                               iterative_max_cycles=5, iterative_thresh=-15, iterative_scale=2.0,  
                               cart_hessian=None, print_thermo=False, title='Optimization',  
                               copy_final_geom=None, level=0)
```

pysisyphus.drivers.perf module

```
pysisyphus.drivers.perf.print_perf_results(results)
```

```
pysisyphus.drivers.perf.run_perf(geom, calc_getter, pals=None, mems=None, pal_range=None,  
                                mem_range=None, repeat=1, kind='forces')
```

pysisyphus.drivers.pka module

```
pysisyphus.drivers.pka.G_aq_from_h5_hessian(h5_hessian, solv_en, T=298.15, p=101325)
```

```
pysisyphus.drivers.pka.direct_cycle(acid_h5, base_h5, acid_solv_en, base_solv_en, G_aq_H=None,  
                                   G_gas_H=-6.28, dG_solv_H=-265.9, T=298.15, p=101325)
```

pysisyphus.drivers.precon_pos_rot module

prp a901cdfacc579eb63b193cbc9043212e8b57746f pysis 340ab6105ac4156f0613b4d0e8f080d9f195530c do_trans
accidentally disabled in transtorque

```
class pysisyphus.drivers.precon_pos_rot.SteepestDescent(geom, max_cycles=1000, max_step=0.05,  
                                                       rms_force=0.05, rms_force_only=True,  
                                                       prefix=None, dump=False,  
                                                       dump_every=100, print_every=100)
```

Bases: object

run()

```
pysisyphus.drivers.precon_pos_rot.center_fragments(frag_list, geom)
```

```
pysisyphus.drivers.precon_pos_rot.form_A(frags, which_frag, formed_bonds)
```

Construct the A-matrices.

AR[(m, n)] (AP[(m, n)]) contains the subset of atoms in Rm (Pm) that forms bonds with Rn (Pn).

```
pysisyphus.drivers.precon_pos_rot.get_fragments_and_bonds(geoms)
```

```
pysisyphus.drivers.precon_pos_rot.get_rot_mat(coords3d_1, coords3d_2, center=False)
```

```

pysisyphus.drivers.precon_pos_rot.get_steps_to_active_atom_mean(frag_lists, iter_frag_lists,
                                                                ind_dict, coords3d, skip=True)

pysisyphus.drivers.precon_pos_rot.get_which_frag(frags)

pysisyphus.drivers.precon_pos_rot.precon_pos_rot(reactants, products, prefix=None,
                                                config={'s2_hs_kappa': 1.0, 's4_hs_kappa': 50.0,
                                                  's4_v_kappa': 1.0, 's4_w_kappa': 1.0,
                                                  's5_hs_kappa': 10.0, 's5_rms_force': 0.01,
                                                  's5_trans': True, 's5_v_kappa': 1.0, 's5_w_kappa':
                                                  3.0, 's5_z_kappa': 2.0})

pysisyphus.drivers.precon_pos_rot.report_frags(rgeom, pgeom, rfrags, pfrags, rbond_diff, pbond_diff)

pysisyphus.drivers.precon_pos_rot.report_mats(name, mats)

pysisyphus.drivers.precon_pos_rot.rotate_inplace(frags, union, bonds)

pysisyphus.drivers.precon_pos_rot.run_precontr(reactant_geom, product_geom, **kwargs)

```

pysisyphus.drivers.rates module

```

class pysisyphus.drivers.rates.ReactionRates(from_, barrier, barrier_si, temperature,
                                              imag_wavenumber, imag_frequency, rate_eyring,
                                              kappa_eyring, rate_wigner=None, kappa_wigner=None,
                                              rate_bell=None, kappa_bell=None, rate_eckart=None,
                                              kappa_eckart=None)

Bases: object

barrier: float

barrier_si: float

from_: str

imag_frequency: float

imag_wavenumber: float

kappa_bell: Optional[float] = None

kappa_eckart: Optional[float] = None

kappa_eyring: float

kappa_wigner: Optional[float] = None

rate_bell: Optional[float] = None

rate_eckart: Optional[float] = None

rate_eyring: float

rate_wigner: Optional[float] = None

temperature: float

```

`pysisyphus.drivers.rates.bell_corr(temperature, imag_frequency)`

Tunneling correction according to Bell.

See <https://onlinelibrary.wiley.com/doi/10.1002/anie.201708489> eq. (1) and eq. (2).

Parameters

- **temperature** (float) -- Temperature in Kelvin.
- **imag_frequency** (float) -- Imaginary frequency in 1/s.

Returns

Unitless tunneling correction according to Bell. Negative kappas are meaningless.

Return type

kappa

`pysisyphus.drivers.rates.eckart_corr(fw_barrier_height, bw_barrier_height, temperature, imag_frequency)`

Tunneling correction according to Eckart.

See [3], [4] and [5]. The correction should be independent of the order fw_barrier_height/bw_barrier_height.

Parameters

- **fw_barrier_height** (float) -- Barrier height in forward direction in Hartree.
- **bw_barrier_height** (float) -- Barrier height in backward direction in Hartree.
- **temperature** (float) -- Temperature in Kelvin.
- **imag_frequency** (float) -- Frequency in 1/s of the imaginary mode at the TS.

Returns

Unitless tunneling correction according to Eckart.

Return type

kappa

`pysisyphus.drivers.rates.eckart_corr_brown(fw_barrier_height, bw_barrier_height, temperature, imag_frequency)`

Tunneling correction according to Eckart.

Wrapper for the TUNL subroutine as given in the appendix of [5].

Parameters

- **fw_barrier_height** (float) -- Barrier height in forward direction in Hartree.
- **bw_barrier_height** (float) -- Barrier height in backward direction in Hartree.
- **temperature** (float) -- Temperature in Kelvin.
- **imag_frequency** (float) -- Frequency in 1/s of the imaginary mode at the TS.

Returns

Unitless tunneling correction according to Eckart.

Return type

kappa

`pysisyphus.drivers.rates.eyring_rate(barrier_height, temperature, trans_coeff=1.0)`

Rate constant in 1/s from the Eyring equation.

See <https://pubs.acs.org/doi/10.1021/acs.organomet.8b00456> "Reaction Rates and Barriers" on p. 3234 and eq. (8).

Parameters

- **barrier_height** (float) -- Barrier height (energy, enthalpy, gibbs energy, ...) in Hartree.
- **temperature** (Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]) -- Temperature in Kelvin.
- **trans_coeff** (float, default: 1.0) -- Unitless transmission coefficient, e.g., obtained from Wigner or Eckart correction.

Returns

Eyring reaction rate in 1/s.

Return type

rate

`pysisyphus.drivers.rates.get_rates(temperature, reactant_thermos, ts_thermo, product_thermos=None)`

`pysisyphus.drivers.rates.get_rates_for_geoms(temperature, reactant_geoms, ts_geom, product_geoms)`

`pysisyphus.drivers.rates.get_rates_for_hessians(temperature, reactant_h5s, ts_h5, product_h5s)`

`pysisyphus.drivers.rates.harmonic_tst_rate(barrier_height, temperature, rs_part_func, ts_part_func, trans_coeff=1.0)`

Rate constant in 1/s from harmonic TST.

See <http://dx.doi.org/10.18419/opus-9841>, chapter 5. Contrary to the Eyring rate this function does only takes a scalar temperature as the partition functions are also functions of the temperature and would have to be recalculated for different temperatures.

A possible extension would be to also support multiple rs/ts partition functions, one for each temperature.

Parameters

- **barrier_height** (float) -- Barrier height (energy, enthalpy, gibbs energy, ...) in Hartree.
- **rs_part_func** (float) -- Partition function of the reactant state.
- **ts_part_func** (float) -- Partition function of the transition state.
- **temperature** (float) -- Temperature in Kelvin.
- **trans_coeff** (float, default: 1.0) -- Unitless transmission coefficient, e.g., obtained from Wigner or Eckart correction.

Returns

HTST reaction rate in 1/s.

Return type

rate

`pysisyphus.drivers.rates.render_rx_rates(rx_rates)`

Return type

str

`pysisyphus.drivers.rates.tunl(alph1, alph2, U, strict=False)`

Eckart correction factor for rate constants according to Brown.

Python adaption of the TUNL subroutine in 4. Appendix of [5].

Parameters

- **alph1** (float) -- Unitless barrier height descriptor. $2 V_1 / (h \nu^*)$; see (2) in [5].

- **alph2** (float) -- Unitless barrier height descriptor. $2 V_2 / (h \nu^*)$; see (2) in [5].
- **u** -- Unitless curvature descriptor. $h \nu^* / kT$; see (2) in [5].
- **strict** (bool, default: False) -- If enabled, arguments are bound checked. Will raise AssertionError if they are out of bounds. TUNL was found to yield accurate results when the arguments are within bounds.

Returns

Unitless tunneling correction according to Eckart.

Return type

G

`pysisyphus.drivers.rates.wigner_corr(temperature, imag_frequency)`

Tunneling correction according to Wigner.

See <https://doi.org/10.1002/qua.25686> eq. (12) and https://doi.org/10.1007/978-3-642-59033-7_9 for the original publication.

Parameters

- **temperature** (float) -- Temperature in Kelvin.
- **imag_frequency** (float) -- Imaginary frequency in 1/s.

Returns

Unitless tunneling correction according to Wigner.

Return type

kappa

pysisyphus.drivers.replace module

`pysisyphus.drivers.replace.get_bond_subgeom(geom, ind, invert=False)`

`pysisyphus.drivers.replace.normalize_replacements(replacements)`

`pysisyphus.drivers.replace.parse_args(args)`

`pysisyphus.drivers.replace.replace_atom(geom, ind, repl_geom, repl_ind, return_full=True, opt=False, use_xtb=False, charge=0, mult=1)`

Replace atom with fragment.

`pysisyphus.drivers.replace.replace_atoms(geom, replacements, opt=False, use_xtb=False, charge=0, mult=1)`

`pysisyphus.drivers.replace.run()`

`pysisyphus.drivers.replace.run_opt(geom, freeze_inds, ff='uff', use_xtb=False, charge=0, mult=1)`

pysisyphus.drivers.scan module

`pysisyphus.drivers.scan.relaxed_1d_scan`(*geom*, *calc_getter*, *constrain_prims*, *start*, *step_size*, *steps*, *opt_key*, *opt_kwargs*, *pref=None*, *callback=None*)

`pysisyphus.drivers.scan.relaxed_scan`(*geom*, *calc_getter*, *constrain_prims*, *target_values*, *title*, *max_cycles=25*, *trust_radius=0.5*, *thresh=0.01*, *dump=True*)

Relaxed scan, allowing fixing of multiple primitive internals.

pysisyphus.drivers.thermo module

`pysisyphus.drivers.thermo.parse_args`(*args*)

`pysisyphus.drivers.thermo.run_thermo`()

Module contents

17.1.1.6 pysisyphus.dynamics package

Submodules

pysisyphus.dynamics.Gaussian module

`class pysisyphus.dynamics.Gaussian.DummyColvar`

Bases: object

`eval`(*x*)

`class pysisyphus.dynamics.Gaussian.Gaussian`(*w=1*, *s=1*, *x0=None*, *colvar=None*, *dump_name=None*)

Bases: object

`__init__`(*w=1*, *s=1*, *x0=None*, *colvar=None*, *dump_name=None*)

See:

<https://doi.org/10.1016/j.cpc.2018.02.017>

$V(f(x)) = w * \exp(-(f(x) - f_0)**2 / (2*s**2))$

$F(x) = -dV/dx = -dV/df * df/dx$

See also:

<https://doi.org/10.1103/PhysRevLett.90.238302>

`calculate`(*coords*, *x0=None*, *gradient=False*)

Return potential and gradient for Gaussian(s) centered at *x0*.

`dump`(*step*, *s*, *w*, *center*)

`eval`(*coords*, *x0=None*)

`gradient`(*coords*, *x0=None*)

property *s*

value(*coords*, *x0=None*)

property *w*

pysisyphus.dynamics.colvars module

class pysisyphus.dynamics.colvars.CVBend(*indices*, ***kwargs*)

Bases: *Colvar*

value(*c3d*)

class pysisyphus.dynamics.colvars.CVDistance(*indices*, ***kwargs*)

Bases: *Colvar*

value(*c3d*)

class pysisyphus.dynamics.colvars.CVTorsion(*indices*, ***kwargs*)

Bases: *Colvar*

value(*c3d*)

class pysisyphus.dynamics.colvars.Colvar(*force_agrad=False*)

Bases: *object*

eval(*coords*)

gradient(*c3d*)

abstract value(*c3d*)

pysisyphus.dynamics.colvars.get_colvar(*key*, *kwargs*)

pysisyphus.dynamics.driver module

class pysisyphus.dynamics.driver.MDResult(*coords*, *t_ps*, *step*, *terminated*, *T*, *E_tot*)

Bases: *tuple*

E_tot

Alias for field number 5

T

Alias for field number 4

coords

Alias for field number 0

step

Alias for field number 2

t_ps

Alias for field number 1

terminated

Alias for field number 3

`pysisyphus.dynamics.driver.get_data_model(atoms, dump_steps)`

`pysisyphus.dynamics.driver.md(geom, v0, steps, dt, remove_com_v=True, thermostat=None, T=298.15, timecon=100, term_funcs=None, constraints=None, constraint_kwargs=None, gaussians=None, verbose=True, print_stride=50, dump_stride=None, h5_group_name='run')`

Velocity verlet integrator.

Parameters

- **geom** (*Geometry*) -- The system for which the dynamics are to be run.
- **v0** (*np.array, floats*) -- Initial velocities in Bohr/fs.
- **steps** (*float*) -- Number of simulation steps.
- **dt** (*float*) -- Timestep in fs.
- **remove_com_v** (*bool, default=True*) -- Remove center-of-mass velocity.
- **thermostat** (*str, optional, default None*) -- Which and whether to use a thermostat.
- **T** (*float, optional, default=None*) -- Desired temperature in thermostated runs.
- **timecon** (*float*) -- Timeconsanst of the thermostat in fs.
- **term_funcs** (*dict, optional*) -- Iterable of functions that are called with the atomic coordinates in every MD cycle and result in termination
- **constraints** (*2d iterable, optional*) -- 2D iterable containing atom indices describing constrained bond lengths. of the MD integration when they evaluate to true.
- **constraint_kwargs** (*dict, optional*) -- Keyword arguments for the constraint algorithm.
- **gaussians** (*list, optional, default=None*) -- List of Gaussians to be used in a meta-dynamics run.
- **verbose** (*bool, default=True*) -- Do additional printing when True.
- **print_stride** (*int, default=50*) -- Report every n-th step.
- **dump_stride** (*int, default=None*) -- If given, MD data will be dumped to a HDF5 file every n-th step.
- **str** (*h5_group_name =*) -- Name of the HDF5 group, used for dumping.
- **optional** -- Name of the HDF5 group, used for dumping.

pysisyphus.dynamics.helpers module

`pysisyphus.dynamics.helpers.dump_coords(atoms, coords, trj_fn)`

`pysisyphus.dynamics.helpers.energy_forces_getter_closure(geom)`

`pysisyphus.dynamics.helpers.get_mb_velocities(masses, cart_coords, T, remove_com_v=True, remove_rot_v=True, seed=None)`

Initial velocities from Maxwell-Boltzmann distribution.

Parameters

- **masses** (*np.array, 1d, shape (number of atoms,)*) -- Atomic masses in amu.

- **cart_coords** (*iterable, 1d, shape (3 * number of atoms,)*) -- Atomic cartesian coordinates. Needed for removal of rotation.
- **T** (*float*) -- Temperature in Kelvin.
- **remove_com_v** (*bool, default=True, optional*) -- Whether to remove center-of-mass velocity.
- **remove_rot_v** (*bool, default=True, optional*) -- Whether to remove rotational velocity.
- **seed** (*int, default=None, optional*) -- Seed for the random-number-generator.

Returns

v -- Initial velocities in Bohr/fs.

Return type

np.array, 2d, shape (number of atoms, 3)

pysisyphus.dynamics.helpers.**get_mb_velocities_for_geom**(*geom, T, remove_com_v=True, remove_rot_v=True, seed=None*)

Initial velocities from Maxwell-Boltzmann distribution.

See 'get_mb_velocities' for explanation.

pysisyphus.dynamics.helpers.**kinetic_energy_for_temperature**(*atom_number, T, fixed_dof=0*)

Kinetic energy for given temperature and number of atoms.

Each atom has three degrees of freedom ($1/2 * 3 == 3/2$).

Parameters

- **atom_number** (*int*) -- Number of atoms. Each atom has three degrees of freedom.
- **T** (*float*) -- Temperature in Kelvin.
- **fixed_dof** (*int, optional, default=0*) -- Number of fixed degrees of freedom, e.g. 3 when the center-of-mass velocity is removed.

Returns

E_kin -- Kinetic energy in Hartree.

Return type

float

pysisyphus.dynamics.helpers.**kinetic_energy_from_velocities**(*masses, velocities*)

Kinetic energy for given velocities and masses.

Parameters

- **masses** (*1d array, shape (number of atoms,)*) -- Atomic masses in amu.
- **velocities** (*2d array, (number of atoms, 3)*) -- Atomic velocities in Bohr/fs.

Returns

E_kin -- Kinetic energy in Hartree.

Return type

float

pysisyphus.dynamics.helpers.**remove_com_velocity**(*v, masses, keep_norm=True*)

Remove center-of-mass velocity.

Returned units vary with the given input units.

Parameters

- **v** (*np.array, 2d, shape (number of atoms, 3)*) -- Velocities.
- **masses** (*np.array, 1d, shape (number of atoms,)*) -- Atomic masses.
- **keep_norm** (*bool, default=True*) -- Whether to rescale v to its original norm, after removal of v_com.

Returns

v -- Velocities without center-of-mass velocity.

Return type

np.array

pysisyphus.dynamics.helpers.**scale_velocities_to_energy**(*masses, v, E_desired*)

Scale velocities to a given temperature.

Parameters

- **masses** (*np.array, 1d, shape (number of atoms,)*) -- Atomic masses in amu.
- **v** (*np.array, 2d, shape (number of atoms, 3)*) -- (Unscaled) velocities in Bohr/fs.
- **E_desired** (*float*) -- Desired kinetic energy in Hartree.

Returns

v -- Scaled velocities in Bohr/fs.

Return type

np.array, 2d, shape (number of atoms, 3)

pysisyphus.dynamics.helpers.**scale_velocities_to_temperatue**(*masses, v, T_desired, fixed_dof=0*)

Scale velocities to a given temperature.

Parameters

- **masses** (*np.array, 1d, shape (number of atoms,)*) -- Atomic masses in amu.
- **v** (*np.array, 2d, shape (number of atoms, 3)*) -- (Unscaled) velocities in Bohr/fs.
- **T_desired** (*float*) -- Desired temperature in Kelvin.
- **fixed_dof** (*int, optional, default=0*) -- Number of fixed degrees of freedom, e.g. 3 when the center-of-mass velocity is removed.

Returns

v -- Scaled velocities in Bohr/fs.

Return type

np.array, 2d, shape (number of atoms, 3)

pysisyphus.dynamics.helpers.**temperature_for_kinetic_energy**(*atom_number, E_kin, fixed_dof=0*)

Temperature for given kinetic energy and atom number.

Each atom has three degrees of freedom ($1/2 * 3 == 3/2$).

Parameters

- **atom_number** (*int*) -- Number of atoms. Each atom has three degrees of freedom.
- **E_kin** (*float*) -- Kinetic energy in Hartree.
- **fixed_dof** (*int, optional, default=0*) -- Number of fixed degrees of freedom, e.g. 3 when the center-of-mass velocity is removed.

Returns

Temperature -- Temperature in Kelvin.

Return type

float

`pysisyphus.dynamics.helpers.unscaled_velocity_distribution(masses, T, seed=None)`

$\exp(-v^2 * m / (2 * kT)) \ln() - 1/2 * v^2 * m / kT \quad v^2 - 2 * \ln() * kT / m \quad v \pm (-2 * \ln() * kT / m)^{0.5} \quad v \pm (-2 * k)^{0.5} * (\ln() * T / m)^{0.5}$

The first term of the RHS is constant. As we later scale the velocities we neglect it. Don't use these velocities unscaled!

$v \pm (\ln() * T / m)^{0.5}$

pysisyphus.dynamics.lincs module

`pysisyphus.dynamics.lincs.lincs_closure(geom, constraints, order=4)`

Drop `conn_nums` and keep `conns_` in a list of list, instead of an array. We could do everything with

```
for i, conn_constr in enumerate(conns)
    pass
```

instead of the pseudo-code way as given in the paper.

pysisyphus.dynamics.mdp module

`pysisyphus.dynamics.mdp.MDPResult`

alias of `MDResult`

`pysisyphus.dynamics.mdp.mdp(geom, steps, dt, term_funcs=None, steps_init=None, E_excess=0.0, displ_length=0.1, epsilon=0.0005, ascent_alpha=0.05, max_ascent_steps=25, max_init_trajs=10, dump=True, seed=None, external_md=False)`

`pysisyphus.dynamics.mdp.parse_raw_term_func(raw_term_func)`

`pysisyphus.dynamics.mdp.parse_raw_term_funcs(raw_term_funcs)`

`pysisyphus.dynamics.mdp.run_md(geom, dt, steps, v0=None, term_funcs=None, external=False)`

pysisyphus.dynamics.rattle module

`pysisyphus.dynamics.rattle.rattle_closure(geom, constraints, dt, tol=0.001, max_cycles=25, energy_forces_getter=None, remove_com_v=True)`

pysisyphus.dynamics.thermostats module

pysisyphus.dynamics.thermostats.**berendsen_closure**(*sigma*, *dof*, *dt*, *tau*=100, *rng*=None)

<https://doi.org/10.1063/1.448118>

pysisyphus.dynamics.thermostats.**csvr_closure**(*sigma*, *dof*, *dt*, *tau*=100, *rng*=None)

Parameters

- **sigma** (*float*) -- Target average value of the kinetic energy ($1/2 \text{ dof } k_B T$) in the same units as `cur_kinetic_energy`.
- **dof** (*int*) -- Degrees of freedom.
- **tau** (*float*) -- Timeconstant of the thermostat. `tau` : float Timeconstant of the thermostat.
- **rng** (*numpy.random.Generator, optional*) -- Instances of a random number generator (RNG). If it is not provided the module-level RNG will be used.

pysisyphus.dynamics.thermostats.**csvr_closure_2**(*sigma*, *dof*, *dt*, *tau*=100, *rng*=None)

pysisyphus.dynamics.thermostats.**sum_noises**(*num*, *rng*=None)

Parameters

- **num** (*int*) -- Number of independent Gaussian noises to be squared.
- **rng** (*numpy.random.Generator, optional*) -- Instances of a random number generator (RNG). If it is not provided the module-level RNG will be used.

Module contents

17.1.1.7 pysisyphus.intcoords package

Submodules

pysisyphus.intcoords.Bend module

class pysisyphus.intcoords.Bend.**Bend**(*indices*, *periodic*=False, *calc_kwargs*=None, *cache*=False)

Bases: *Primitive*

pysisyphus.intcoords.Bend2 module

class pysisyphus.intcoords.Bend2.**Bend2**(*indices*, *periodic*=False, *calc_kwargs*=None, *cache*=False)

Bases: *Bend*

pysisyphus.intcoords.BondedFragment module

```
class pysisyphus.intcoords.BondedFragment.BondedFragment(indices, bond_indices, **kwargs)
    Bases: Primitive
```

pysisyphus.intcoords.Cartesian module

```
class pysisyphus.intcoords.Cartesian.Cartesian(*args, **kwargs)
    Bases: Primitive

class pysisyphus.intcoords.Cartesian.CartesianX(*args, **kwargs)
    Bases: Cartesian
    cart_axis = 0

class pysisyphus.intcoords.Cartesian.CartesianY(*args, **kwargs)
    Bases: Cartesian
    cart_axis = 1

class pysisyphus.intcoords.Cartesian.CartesianZ(*args, **kwargs)
    Bases: Cartesian
    cart_axis = 2
```

pysisyphus.intcoords.CartesianCoords module

```
class pysisyphus.intcoords.CartesianCoords.CartesianCoords(atoms, coords3d, masses,
                                                             freeze_atoms=None, *,
                                                             mass_weighted=False, **kwargs)

    Bases: CoordSys

    property coords: ndarray[Any, dtype[_ScalarType_co]]
        Getter for coordinates in this coordinate system.

    property coords3d: ndarray[Any, dtype[_ScalarType_co]]
        Getter for 3d Cartesian coordinates.

    property inv_masses_rep_sqrt: ndarray[Any, dtype[_ScalarType_co]]

    property masses: ndarray[Any, dtype[_ScalarType_co]]

    property masses_sqrt: ndarray[Any, dtype[_ScalarType_co]]

    project_hessian(hessian)
        Project Hessian in the current coordinate system.

    transform_forces(cart_forces)
        Transform Cartesian forces to this coordinate system.

        Return type
        ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

    transform_hessian(cart_hessian, int_gradient=None)
        Transform Cartesian Hessian to this coordinate system.
```

transform_int_step(*step*, *update_constraints=False*, *pure=False*)

Transform step in this coordinate system to Cartesians.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

property typed_prims: List

List of (primitive) internal coordinates.

May be empty, e.g., when the coordinate system is Cartesian.

class pysisyphus.intcoords.CartesianCoords.**MWCartesianCoords**(*args, **kwargs)

Bases: [CartesianCoords](#)

pysisyphus.intcoords.Coords module

class pysisyphus.intcoords.Coords.**CoordSys**(*args, **kwargs)

Bases: Protocol

abstract property coords: ndarray[Any, dtype[_ScalarType_co]]

Getter for coordinates in this coordinate system.

abstract property coords3d: ndarray[Any, dtype[_ScalarType_co]]

Getter for 3d Cartesian coordinates.

abstract project_hessian(hessian)

Project Hessian in the current coordinate system.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

abstract transform_forces(cart_forces)

Transform Cartesian forces to this coordinate system.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

abstract transform_hessian(cart_hessian, int_gradient)

Transform Cartesian Hessian to this coordinate system.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

abstract transform_int_step(step, update_constraints=False, pure=False)

Transform step in this coordinate system to Cartesians.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

abstract property typed_prims: List

List of (primitive) internal coordinates.

May be empty, e.g., when the coordinate system is Cartesian.

pysisyphus.intcoords.DLC module

class pysisyphus.intcoords.DLC.DLC(*args, full_set=True, **kwargs)

Bases: [RedundantCoords](#)

property B

Wilson B-Matrix in the non-redundant subspace.

property U

backtransform_hessian(*args, **kwargs)

Transform Hessian in internal coordinates to Cartesians.

property constraints

property coords

freeze_primitives(typed_prims)

Freeze primitive internal coordinates.

Parameters

typed_prims (iterable of typed primitives) -- Iterable containing typed_primitives, starting with a PrimType and followed by atom indices.

get_active_set(B, inv_thresh=None)

See [5] between Eq. (7) and Eq. (8) for advice regarding the threshold.

get_constrained_U(constraint_vecs)

project_hessian(H)

As we already work in the non-redundant subspace we don't have to project/shift the hessian as we do it for simple redundant internal coordinates.

project_primitive_on_active_set(prim_ind)

reset_constraints()

set_active_set()

transform_hessian(cart_hessian, int_gradient=None)

Transform Cartesian Hessian to DLC.

transform_int_step(step, *args, **kwargs)

As the transformation is done in primitive internal coordinates we convert the DLC back to primitive coordinates.

class pysisyphus.intcoords.DLC.HDLC(*args, **kwargs)

Bases: [DLC](#)

pysisyphus.intcoords.DistanceFunction module

```
class pysisyphus.intcoords.DistanceFunction.DistanceFunction(indices, coeff=-1, **kwargs)
    Bases: Primitive
```

pysisyphus.intcoords.DummyTorsion module

```
class pysisyphus.intcoords.DummyTorsion.DummyTorsion(indices, *args, fix_inner=True, **kwargs)
    Bases: Torsion
    static get_coords3d_and_indices_ext(coords3d, indices)
    static get_fourth_coords(coords3d, indices, r=1.889, theta=90)

    M N <- add
        /
        u v
        /
        P
    m, o, p, n = indices
```

pysisyphus.intcoords.LinearBend module

```
class pysisyphus.intcoords.LinearBend.LinearBend(*args, complement=False, **kwargs)
    Bases: Primitive
    calculate(coords3d, indices=None, gradient=False)
    jacobian(coords3d, indices=None)
```

pysisyphus.intcoords.LinearDisplacement module

```
class pysisyphus.intcoords.LinearDisplacement.LinearDisplacement(*args, complement=False,
                                                                    **kwargs)
    Bases: Primitive
    calculate(coords3d, indices=None, gradient=False)
    jacobian(coords3d, indices=None)
```

pysisyphus.intcoords.OutOfPlane module

class pysisyphus.intcoords.OutOfPlane.**OutOfPlane**(*indices, periodic=False, calc_kwargs=None, cache=False*)

Bases: *Primitive*

[1] [https://doi.org/10.1002/\(SICI\)1096-987X\(19990730\)20:10<1067::AID-JCC9>3.0.CO;2-V](https://doi.org/10.1002/(SICI)1096-987X(19990730)20:10<1067::AID-JCC9>3.0.CO;2-V)
Lee, 1999

pysisyphus.intcoords.PrimTypes module

pysisyphus.intcoords.PrimTypes.**PT**

alias of *PrimTypes*

class pysisyphus.intcoords.PrimTypes.**PrimTypes**(*value*)

Bases: *OrderedEnum*

An enumeration.

AUX_BOND = 1

AUX_INTERFRAG_BOND = 4

BEND = 5

BEND2 = 28

BOND = 0

BONDED_FRAGMENT = 25

CARTESIAN = 21

CARTESIAN_X = 22

CARTESIAN_Y = 23

CARTESIAN_Z = 24

DISTANCE_FUNCTION = 27

DUMMY_IMPROPER = 30

DUMMY_TORSION = 26

HYDROGEN_BOND = 2

IMPROPER_DIHEDRAL = 9

INTERFRAG_BOND = 3

LINEAR_BEND = 6

LINEAR_BEND_COMPLEMENT = 7

LINEAR_DISPLACEMENT = 11

LINEAR_DISPLACEMENT_COMPLEMENT = 12

```
OUT_OF_PLANE = 10
PROPER_DIHEDRAL = 8
PROPER_DIHEDRAL2 = 29
ROBUST_TORSION1 = 31
ROBUST_TORSION2 = 32
ROTATION = 17
ROTATION_A = 18
ROTATION_B = 19
ROTATION_C = 20
TRANSLATION = 13
TRANSLATION_X = 14
TRANSLATION_Y = 15
TRANSLATION_Z = 16
```

```
pysisyphus.intcoords.PrimTypes.get_bonded_frag_coord()
```

```
pysisyphus.intcoords.PrimTypes.get_dist_func()
```

```
pysisyphus.intcoords.PrimTypes.get_rot_coord(cls)
```

```
pysisyphus.intcoords.PrimTypes.normalize_prim_input(prim_inp)
```

Normalize input for define_prims and constrain_prims

The `intcoords.RedundantCoords` constructor expects lists of integer lists (tuples) for arguments like 'define_prims' and 'constrain_prims'. The first item of every list determines the type of primitive coordinate. Currently there are about 20 different types and it is hard to remember all of them.

So we also allow a more human friendly input, that is normalized here. The most common primitives are:

0: BOND 5: BEND 8: PROPER_DIHEDRAL

This function maps inputs like ["BOND", 1, 2] to [PrimTypes.BOND, 1, 2] etc.

Always returns a list of tuples, as some `prim_inps` expand to multiple coordinates, e.g., XYZ or ATOM.

```
pysisyphus.intcoords.PrimTypes.normalize_prim_inputs(prim_inps)
```

```
pysisyphus.intcoords.PrimTypes.prim_for_human(prim_type, val)
```

```
pysisyphus.intcoords.PrimTypes.prims_from_prim_inputs(prim_inps)
```

pysisyphus.intcoords.Primitive module

```
class pysisyphus.intcoords.Primitive.Primitive(indices, periodic=False, calc_kwargs=None,
                                                cache=False)
```

Bases: object

calculate(coords3d, indices=None, gradient=False)

jacobian(coords3d, indices=None)

log(msg, lvl=10)

log_dbg(msg)

static parallel(u, v, thresh=1e-06)

static rho(atoms, coords3d, indices)

set_cross_vec(coords3d, indices)

weight(atoms, coords3d, f_damping=0.12)

pysisyphus.intcoords.RedundantCoords module

```
class pysisyphus.intcoords.RedundantCoords.HybridRedundantCoords(*args, **kwargs)
```

Bases: [RedundantCoords](#)

```
class pysisyphus.intcoords.RedundantCoords.RedundantCoords(atoms, coords3d, masses=None,
                                                             bond_factor=1.3, typed_prims=None,
                                                             define_prims=None,
                                                             constrain_prims=None,
                                                             freeze_atoms=None,
                                                             freeze_atoms_exclude=False,
                                                             internals_with_frozen=False,
                                                             define_for=None, bonds_only=False,
                                                             check_bends=True, rebuild=True,
                                                             bend_min_deg=15,
                                                             dihed_max_deg=175,
                                                             lb_min_deg=175, weighted=False,
                                                             min_weight=0.3,
                                                             svd_inv_thresh=0.000316,
                                                             recalc_B=False, tric=False,
                                                             hybrid=False, hbond_angles=False,
                                                             rm_for_frag=None)
```

Bases: object

property B

Wilson B-Matrix

property B_inv

Generalized inverse of the Wilson B-Matrix.

property B_inv_prim

Generalized inverse of the primitive Wilson B-Matrix.

property B_prim

Wilson B-Matrix

property Bt_inv

Transposed generalized inverse of the Wilson B-Matrix.

property Bt_inv_prim

Transposed generalized inverse of the primitive Wilson B-Matrix.

property C

Diagonal matrix. Entries for constraints are set to one.

property P

Projection matrix onto B. See [1] Eq. (4).

backtransform_hessian(*redund_hessian*, *int_gradient=None*)

Transform Hessian in internal coordinates to Cartesians.

property bend_atom_indices**property bend_indices****property bond_atom_indices****property bond_indices****property bond_typed_prims****property cartesian_indices****clear()****property constrained_indices****property coords****property coords3d****property dihedral_atom_indices****property dihedral_indices****eval(*coords3d*, *attr=None*)****get_K_matrix(*int_gradient=None*)****get_index_of_typed_prim(*typed_prim*)**

Index in self.typed_prims for the supplied typed_prim.

get_prim_internals_by_indices(*indices*)**inv_B(*B*)****inv_Bt(*B*)****property linear_bend_indices****log(*message*)****log_int_grad_msg(*int_gradient*)**

property outofplane_indices

property prim_coords

property prim_indices_set

property prim_internals

property primitives

print_typed_prims()

project_hessian(*H*, *shift=1000*)

Expects a hessian in internal coordinates. See Eq. (11) in [1].

project_vector(*vector*)

Project supplied vector onto range of B.

return_inds(*slice_*)

property rotation_indices

set_inds_from_typed_prims(*typed_prims*)

set_primitive_indices(*atoms*, *coords3d*)

transform_forces(*cart_forces*)

Combination of Eq. (9) and (11) in [1].

transform_hessian(*cart_hessian*, *int_gradient=None*)

Transform Cartesian Hessian to internal coordinates.

transform_int_step(*int_step*, *update_constraints=False*, *pure=False*)

property translation_indices

property typed_prims

class pysisyphus.intcoords.RedundantCoords.**TMTRIC**(*atoms*, **args*, ***kwargs*)

Bases: [*TRIC*](#)

class pysisyphus.intcoords.RedundantCoords.**TRIC**(**args*, ***kwargs*)

Bases: [*RedundantCoords*](#)

pysisyphus.intcoords.Rotation module

class pysisyphus.intcoords.Rotation.**Rotation**(*indices*, **args*, *ref_coords3d*, ***kwargs*)

Bases: [*Primitive*](#)

See (II. Theory) in [1], Eq. (3) - (14)

index = **None**

class pysisyphus.intcoords.Rotation.**RotationA**(*indices*, **args*, *ref_coords3d*, ***kwargs*)

Bases: [*Rotation*](#)

index = **0**

```
class pysisyphus.intcoords.Rotation.RotationB(indices, *args, ref_coords3d, **kwargs)
```

Bases: *Rotation*

index = 1

```
class pysisyphus.intcoords.Rotation.RotationC(indices, *args, ref_coords3d, **kwargs)
```

Bases: *Rotation*

index = 2

```
pysisyphus.intcoords.Rotation.compare_to_geometric(c3d, ref_c3d, dR, dF, dqdx, dvdx, atol=1e-14)
```

pysisyphus.intcoords.Stretch module

```
class pysisyphus.intcoords.Stretch.Stretch(indices, periodic=False, calc_kwargs=None, cache=False)
```

Bases: *Primitive*

pysisyphus.intcoords.Torsion module

```
class pysisyphus.intcoords.Torsion.Torsion(indices, periodic=False, calc_kwargs=None, cache=False)
```

Bases: *Primitive*

pysisyphus.intcoords.Torsion2 module

```
class pysisyphus.intcoords.Torsion2.Torsion2(indices, periodic=False, calc_kwargs=None,
                                              cache=False)
```

Bases: *Torsion*

pysisyphus.intcoords.Translation module

```
class pysisyphus.intcoords.Translation.Translation(*args, **kwargs)
```

Bases: *Primitive*

See (II. Theory) in [1], Eq. (2)

```
class pysisyphus.intcoords.Translation.TranslationX(*args, **kwargs)
```

Bases: *Translation*

cart_axis = 0

```
class pysisyphus.intcoords.Translation.TranslationY(*args, **kwargs)
```

Bases: *Translation*

cart_axis = 1

```
class pysisyphus.intcoords.Translation.TranslationZ(*args, **kwargs)
```

Bases: *Translation*

cart_axis = 2

pysisyphus.intcoords.augment_bonds module

pysisyphus.intcoords.augment_bonds.**augment_bonds**(geom, root=0, proj=False)

pysisyphus.intcoords.augment_bonds.**find_missing_bonds_by_projection**(geom, hessian,
bond_factor=2.0,
bond_thresh=0.35,
concerted_thresh=0.35,
root=0)

pysisyphus.intcoords.augment_bonds.**find_missing_strong_bonds**(geom, hessian, bond_factor=1.7,
thresh=0.3, root=0)

pysisyphus.intcoords.derivatives module

pysisyphus.intcoords.derivatives.**d2q_a**(m0, m1, m2, o0, o1, o2, n0, n1, n2)
Bend, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_a2**(m0, m1, m2, o0, o1, o2, n0, n1, n2)
Bend2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_b**(m0, m1, m2, n0, n1, n2)
Stretch, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_d**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)
Torsion, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_d2**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)
Torsion2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_lb**(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)
Linear Bend, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_ld**(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)
Linear Displacement, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_oop**(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)
OutOfPlane, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_rd1**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)
RobustTorsion1, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**d2q_rd2**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)
RobustTorsion2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**dq_a**(m0, m1, m2, o0, o1, o2, n0, n1, n2)
Bend, first derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**dq_a2**(m0, m1, m2, o0, o1, o2, n0, n1, n2)
Bend2, first derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**dq_b**(m0, m1, m2, n0, n1, n2)
Stretch, first derivative wrt. Cartesians

pysisyphus.intcoords.derivatives.**dq_d**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)
Torsion, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_d2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

Torsion2, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_lb(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Bend, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_ld(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Displacement, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_oop(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)`

OutOfPlane, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_rd1(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion1, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.dq_rd2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion2, first derivative wrt. Cartesians

`pysisyphus.intcoords.derivatives.q_a(m0, m1, m2, o0, o1, o2, n0, n1, n2)`

Bend

`pysisyphus.intcoords.derivatives.q_a2(m0, m1, m2, o0, o1, o2, n0, n1, n2)`

Bend2

`pysisyphus.intcoords.derivatives.q_b(m0, m1, m2, n0, n1, n2)`

Stretch

`pysisyphus.intcoords.derivatives.q_d(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

Torsion

`pysisyphus.intcoords.derivatives.q_d2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

Torsion2

`pysisyphus.intcoords.derivatives.q_lb(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Bend

`pysisyphus.intcoords.derivatives.q_ld(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Displacement

`pysisyphus.intcoords.derivatives.q_oop(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)`

OutOfPlane

`pysisyphus.intcoords.derivatives.q_rd1(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion1

`pysisyphus.intcoords.derivatives.q_rd2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion2

pysisyphus.intcoords.eval module

class `pysisyphus.intcoords.eval.PrimInternal(inds, val, grad=None)`

Bases: object

`pysisyphus.intcoords.eval.augment_primitives(missing_prims, coords3d, prim_indices, fragments)`

`pysisyphus.intcoords.eval.check_primitives(coords3d, primitives, B=None, thresh=1e-06, logger=None)`

pysisyphus.intcoords.eval.**eval_B**(coords3d, primitives)

pysisyphus.intcoords.eval.**eval_primitives**(coords3d, primitives)

pysisyphus.intcoords.exceptions module

exception pysisyphus.intcoords.exceptions.DifferentCoordLengthsException

Bases: Exception

exception pysisyphus.intcoords.exceptions.DifferentPrimitivesException

Bases: Exception

exception pysisyphus.intcoords.exceptions.NeedNewInternalsException(*coords3d*, *args,
invalid_inds=None,
invalid_prims=None,
**kwargs)

Bases: Exception

exception pysisyphus.intcoords.exceptions.PrimitiveNotDefinedException(*typed_prim*, *args,
**kwargs)

Bases: Exception

exception pysisyphus.intcoords.exceptions.RebuiltInternalsException(*args, *typed_prims*=None,
**kwargs)

Bases: Exception

pysisyphus.intcoords.findiffs module

pysisyphus.intcoords.findiffs.**fin_diff_B**(primitive, coords3d, delta=1e-06)

Derivatives of a primitive internal gradient wrt its defining cartesian coordinates.

pysisyphus.intcoords.findiffs.**fin_diff_prim**(primitive, coords3d, delta=1e-06)

Derivatives of a primitive internal gradient wrt its defining cartesian coordinates.

pysisyphus.intcoords.generate_derivatives module

class pysisyphus.intcoords.generate_derivatives.FuncResult(*d0*, *d1*, *d2*, *f0*, *f1*, *f2*)

Bases: tuple

d0

Alias for field number 0

d1

Alias for field number 1

d2

Alias for field number 2

f0

Alias for field number 3

f1

Alias for field number 4

f2

Alias for field number 5

```
pysisyphus.intcoords.generate_derivatives.generate_wilson(generate=None, out_fn='derivatives.py',
                                                         use_mpmath=False)
```

```
pysisyphus.intcoords.generate_derivatives.make_deriv_funcs(base_expr, dx, args, names, comment,
                                                         use_mpmath=True)
```

```
pysisyphus.intcoords.generate_derivatives.make_py_func(exprs, args=None, name=None, comment="",
                                                         use_mpmath=False)
```

pysisyphus.intcoords.helpers module

```
pysisyphus.intcoords.helpers.form_coordinate_union(geom1, geom2)
```

```
pysisyphus.intcoords.helpers.get_bond_difference(geom1, geom2, bond_factor=1.3)
```

Return formed and broken bonds when going from geom1 to geom2.

```
pysisyphus.intcoords.helpers.get_bond_differences_verbose(geom1, geom2, bond_factor=1.3,
                                                         key1='geom1', key2='geom2')
```

```
pysisyphus.intcoords.helpers.get_step(geom, coords)
```

```
pysisyphus.intcoords.helpers.get_tangent(prims1, prims2, dihedral_inds, normalize=False)
```

Normalized tangent between primitive internal coordinates.

Tangent pointing from prims1 to prims2 in primitive internal coordinates, taking into account the periodicity of dihedral angles.

Parameters

- **prims1** (*np.array*) -- 1d-array of primitive internal coordinates in the order (stretches, bends, dihedrals).
- **prims2** (*np.array*) -- See prims1.
- **dihedral_inds** (*list of int*) -- Dihedral indices in prims1 and prims2.

Returns

tangent -- 1d array containing the normalized tangent pointing from prims1 to prims2.

Return type

np.array

```
pysisyphus.intcoords.helpers.get_weighted_bond_mode(weighted_bonds, coords3d,
                                                         remove_translation=True)
```

```
pysisyphus.intcoords.helpers.get_weighted_bond_mode_getter(target_weighted_bonds,
                                                         bond_factor=1.2, fractional=False)
```

Create input for `intcoords.helpers.get_weighted_bond_mode`.

Compared to the rest of `pysisyphus` this method uses a slightly lowered bond factor, so it is more strict regarding what is considered a bond and what not.

pysisyphus.intcoords.helpers.**interfragment_distance**(frag1, frag2, coords3d)

Return type

float

pysisyphus.intcoords.helpers.**merge_coordinate_definitions**(geom1, geom2)

pysisyphus.intcoords.helpers.**verbose_bond_difference**(formed, broken, key1, key2, atoms=None)

pysisyphus.intcoords.mp_derivatives module

pysisyphus.intcoords.mp_derivatives.**d2q_a**(m0, m1, m2, o0, o1, o2, n0, n1, n2)

Bend, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_a2**(m0, m1, m2, o0, o1, o2, n0, n1, n2)

Bend2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_b**(m0, m1, m2, n0, n1, n2)

Stretch, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_d**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

Torsion, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_d2**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

Torsion2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_lb**(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)

Linear Bend, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_ld**(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)

Linear Displacement, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_oop**(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)

OutOfPlane, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_rd1**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

RobustTorsion1, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**d2q_rd2**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

RobustTorsion2, 2nd derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**dq_a**(m0, m1, m2, o0, o1, o2, n0, n1, n2)

Bend, first derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**dq_a2**(m0, m1, m2, o0, o1, o2, n0, n1, n2)

Bend2, first derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**dq_b**(m0, m1, m2, n0, n1, n2)

Stretch, first derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**dq_d**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

Torsion, first derivative wrt. Cartesians

pysisyphus.intcoords.mp_derivatives.**dq_d2**(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)

Torsion2, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.dq_lb(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Bend, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.dq_ld(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Displacement, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.dq_oop(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)`

OutOfPlane, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.dq_rd1(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion1, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.dq_rd2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion2, first derivative wrt. Cartesians

`pysisyphus.intcoords.mp_derivatives.q_a(m0, m1, m2, o0, o1, o2, n0, n1, n2)`

Bend

`pysisyphus.intcoords.mp_derivatives.q_a2(m0, m1, m2, o0, o1, o2, n0, n1, n2)`

Bend2

`pysisyphus.intcoords.mp_derivatives.q_b(m0, m1, m2, n0, n1, n2)`

Stretch

`pysisyphus.intcoords.mp_derivatives.q_d(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

Torsion

`pysisyphus.intcoords.mp_derivatives.q_d2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

Torsion2

`pysisyphus.intcoords.mp_derivatives.q_lb(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Bend

`pysisyphus.intcoords.mp_derivatives.q_ld(m0, m1, m2, o0, o1, o2, n0, n1, n2, p0, p1, p2)`

Linear Displacement

`pysisyphus.intcoords.mp_derivatives.q_oop(m0, m1, m2, n0, n1, n2, o0, o1, o2, p0, p1, p2)`

OutOfPlane

`pysisyphus.intcoords.mp_derivatives.q_rd1(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion1

`pysisyphus.intcoords.mp_derivatives.q_rd2(m0, m1, m2, o0, o1, o2, p0, p1, p2, n0, n1, n2)`

RobustTorsion2

pysisyphus.intcoords.setup module

```
class pysisyphus.intcoords.setup.CoordInfo(bonds, hydrogen_bonds, interfrag_bonds,
                                           aux_interfrag_bonds, bends, linear_bends,
                                           linear_bend_complements, proper_dihedrals,
                                           improper_dihedrals, translation_inds, rotation_inds,
                                           cartesian_inds, typed_prims, fragments)
```

Bases: tuple

aux_interfrag_bonds

Alias for field number 3

bends

Alias for field number 4

bonds

Alias for field number 0

cartesian_inds

Alias for field number 11

fragments

Alias for field number 13

hydrogen_bonds

Alias for field number 1

improper_dihedrals

Alias for field number 8

interfrag_bonds

Alias for field number 2

linear_bend_complements

Alias for field number 6

linear_bends

Alias for field number 5

proper_dihedrals

Alias for field number 7

rotation_inds

Alias for field number 10

translation_inds

Alias for field number 9

typed_prims

Alias for field number 12

```
pysisyphus.intcoords.setup.connect_fragments(cdm, fragments, max_aux=3.78, aux_factor=1.3,  
                                              logger=None)
```

Determine the smallest interfragment bond for a list of fragments and a condensed distance matrix. For more than a few fragments this function performs poorly, as each fragment is connected to all remaining fragments, leading to an explosion of bonds, bends and dihedrals.

```
pysisyphus.intcoords.setup.connect_fragments_ahlricks(cdm, fragments, atoms, min_dist_scale=1.1,  
                                                      scale=1.2, avoid_h=False, logger=None)
```

```
pysisyphus.intcoords.setup.connect_fragments_kmeans(cdm, fragments, atoms,  
                                                    aux_below_thresh=3.7807,  
                                                    aux_add_dist=2.8356, aux_keep=5,  
                                                    aux_no_hh=True, min_dist_thresh=5.0,  
                                                    min_scale=1.2, logger=None)
```

Generate (auxiliary) interfragment bonds.

In the a first step, minimum distance interfragment bonds (IFBs) are determined between all possible fragment pairs. Similarly, possible auxiliary IFBs are determined. Candidates for auxiliary IFBs are:

IFB \leq aux_below_thresh, default 2 Å IFB \leq (minimum distance IFB + aux_add_dist), default 1.5 Å

By default, only the first aux_keep (default = 5) auxiliary IFBs are kept.

Connecting all fragments can lead to bonds between very distant atoms. If more than two fragments are present we cluster the minimum distance IFB distances using KMeans, to determine a reasonable length for valid IFBs. We start out with two clusters and increase the number of cluster until the center of one cluster is around the scaled global minimum distance between the fragments. The center of this cluster is then used as a cutoff vor valid IFBs.

After pruning all possible IFBs we can determine the fragment pairs, that are actually connected. This information is then used to also prune possible interfragment bonds. Only auxiliary IFBs between fragments that are actually connected via IFBs are kept.

```
pysisyphus.intcoords.setup.get_bend_inds(coords3d, bond_inds, min_deg, max_deg, logger=None)
```

```
pysisyphus.intcoords.setup.get_bond_mat(geom, bond_factor=1.3)
```

```
pysisyphus.intcoords.setup.get_bond_sets(atoms, coords3d, bond_factor=1.3, return_cdm=False,
                                          return_cbm=False)
```

I'm sorry, but this function does not return sets, but an int ndarray.

```
pysisyphus.intcoords.setup.get_dihedral_inds(coords3d, bond_inds, bend_inds, max_deg,
                                             logger=None)
```

```
pysisyphus.intcoords.setup.get_fragments(atoms, coords, bond_inds=None, bond_factor=1.3)
```

This misses unconnected single atoms!

```
pysisyphus.intcoords.setup.get_hydrogen_bond_inds(atoms, coords3d, bond_inds, logger=None)
```

```
pysisyphus.intcoords.setup.get_hydrogen_bond_inds_v2(atoms, coords3d, bond_inds, logger=None)
```

```
pysisyphus.intcoords.setup.get_linear_bend_inds(coords3d, cbm, bends, min_deg=175, max_bonds=4,
                                                logger=None)
```

```
pysisyphus.intcoords.setup.get_pair_covalent_radii(atoms)
```

```
pysisyphus.intcoords.setup.get_primitives(coords3d, typed_prims, logger=None)
```

```
pysisyphus.intcoords.setup.setup_redundant(atoms, coords3d, factor=1.3, define_prims=None,
                                           min_deg=15, dihed_max_deg=175, lb_min_deg=None,
                                           lb_max_bonds=4, min_weight=None, tric=False,
                                           hybrid=False, interfrag_hbonds=True,
                                           hbond_angles=False, freeze_atoms=None, define_for=None,
                                           internals_with_frozen=False, rm_for_frag=None,
                                           logger=None)
```

```
pysisyphus.intcoords.setup.setup_redundant_from_geom(geom, *args, **kwargs)
```

```
pysisyphus.intcoords.setup.sort_by_prim_type(to_sort=None)
```

pysisyphus.intcoords.setup_fast module

```
pysisyphus.intcoords.setup_fast.find_bonds(coords3d, bonds, min_deg, max_deg, logger=None)

pysisyphus.intcoords.setup_fast.find_bonds(atoms, coords3d, covalent_radii=None, bond_factor=1.3,
                                             min_dist=0.1)

pysisyphus.intcoords.setup_fast.find_bonds_bends(geom, bond_factor=1.3, min_deg=15,
                                                  max_deg=175)

pysisyphus.intcoords.setup_fast.find_bonds_for_geom(geom, bond_factor=1.3)

pysisyphus.intcoords.setup_fast.find_dihedrals(coords3d, bonds, bends, max_deg, logger=None)

pysisyphus.intcoords.setup_fast.get_bend_candidates(bonds)
    Also yields duplicates [a, b, c] and [c, b, a].

pysisyphus.intcoords.setup_fast.get_bond_vec_getter(atoms, covalent_radii, bonds_for_inds,
                                                    no_bonds_with=None, bond_factor=1.3)

pysisyphus.intcoords.setup_fast.get_max_bond_dists(atoms, bond_factor, covalent_radii=None)
```

pysisyphus.intcoords.update module

```
pysisyphus.intcoords.update.correct_dihedrals(new_dihedrals, old_dihedrals)
```

Dihedrals are periodic. Going from -179° to 179° is not a step of 358° , but a step of 2° . By considering the actual distance of the dihedrals from the correct step can be calculated.

$$\text{dihedral step length} = \text{abs}(\text{abs}(\text{new_dihedral}) -) + \text{abs}(\text{abs}(\text{old_dihedral}) -)$$

or put differently

$$\text{dihedral step length} = \text{abs}(\text{abs}(\text{new_dihedral} - \text{old_dihedral}) - 2^*)$$

The sign is left to be determined. Going from -179° to 179° (roughly $-- = 2$) is a counter clockwise rotation and the dihedral has to decrease below $-$. Going from 179° to -179° (roughly $-- = -2$) is a clockwise rotation and the dihedral increases above $-$. So the correct sign corresponds to the negative sign of the original difference.

original difference 2 \rightarrow dihedral must decrease \rightarrow sign = -1 original difference -2 \rightarrow dihedral must increase \rightarrow sign = +1

Overall, the old dihedral is modified by the actual step length with the correct sign.

```
pysisyphus.intcoords.update.inds_from_prim_types(typed_prims, prim_types)
```

```
pysisyphus.intcoords.update.transform_int_step(int_step, old_cart_coords, cur_internals, Bt_inv_prim,
                                                primitives, typed_prims=None, dihedral_inds=None,
                                                rotation_inds=None, bend_inds=None,
                                                check_dihedrals=False, check_bends=False,
                                                bend_min_deg=15, bend_max_deg=175,
                                                freeze_atoms=None, constrained_inds=None,
                                                update_constraints=False, cart_rms_thresh=1e-06,
                                                Bt_inv_prim_getter=None, max_cycles=25,
                                                logger=None)
```

Transformation is done in primitive internals, so int_step must be given in primitive internals and not in DLC!

`pysisyphus.intcoords.update.update_internals`(*new_coords3d, old_internals, primitives, dihedral_inds, rotation_inds, bend_inds, check_dihedrals=False, check_bends=False, bend_min_deg=15, bend_max_deg=175, rotation_thresh=0.9, logger=None*)

pysisyphus.intcoords.valid module

`pysisyphus.intcoords.valid.are_collinear`(*vec1, vec2, deg_thresh=179.5*)

`pysisyphus.intcoords.valid.bend_valid`(*coords3d, indices, min_deg, max_deg*)

`pysisyphus.intcoords.valid.check_typed_prims`(*coords3d, typed_prims, bend_min_deg, dihed_max_deg, lb_min_deg, logger=None, check_bends=True*)

`pysisyphus.intcoords.valid.dihedral_valid`(*coords3d, inds, deg_thresh=177.5*)

`pysisyphus.intcoords.valid.dihedrals_are_valid`(*coords3d, dihedral_inds, logger=None*)

Module contents

class `pysisyphus.intcoords.Bend`(*indices, periodic=False, calc_kwargs=None, cache=False*)

Bases: [*Primitive*](#)

class `pysisyphus.intcoords.Bend2`(*indices, periodic=False, calc_kwargs=None, cache=False*)

Bases: [*Bend*](#)

class `pysisyphus.intcoords.CartesianCoords`(*atoms, coords3d, masses, freeze_atoms=None, *, mass_weighted=False, **kwargs*)

Bases: [*CoordSys*](#)

property `coords`: `ndarray[Any, dtype[_ScalarType_co]]`

Getter for coordinates in this coordinate system.

property `coords3d`: `ndarray[Any, dtype[_ScalarType_co]]`

Getter for 3d Cartesian coordinates.

property `inv_masses_rep_sqrt`: `ndarray[Any, dtype[_ScalarType_co]]`

property `masses`: `ndarray[Any, dtype[_ScalarType_co]]`

property `masses_sqrt`: `ndarray[Any, dtype[_ScalarType_co]]`

project_hessian(*hessian*)

Project Hessian in the current coordinate system.

transform_forces(*cart_forces*)

Transform Cartesian forces to this coordinate system.

Return type

`ndarray[Any, dtype[TypeVar[_ScalarType_co, bound= generic, covariant=True]]]`

transform_hessian(*cart_hessian, int_gradient=None*)

Transform Cartesian Hessian to this coordinate system.

transform_int_step(*step*, *update_constraints=False*, *pure=False*)

Transform step in this coordinate system to Cartesians.

Return type

ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]

property typed_prims: List

List of (primitive) internal coordinates.

May be empty, e.g., when the coordinate system is Cartesian.

class pysisyphus.intcoords.**CartesianX**(*args, **kwargs)

Bases: [Cartesian](#)

cart_axis = 0

class pysisyphus.intcoords.**CartesianY**(*args, **kwargs)

Bases: [Cartesian](#)

cart_axis = 1

class pysisyphus.intcoords.**CartesianZ**(*args, **kwargs)

Bases: [Cartesian](#)

cart_axis = 2

class pysisyphus.intcoords.**DLC**(*args, *full_set=True*, **kwargs)

Bases: [RedundantCoords](#)

property B

Wilson B-Matrix in the non-redundant subspace.

property U

backtransform_hessian(*args, **kwargs)

Transform Hessian in internal coordinates to Cartesians.

property constraints

property coords

freeze_primitives(*typed_prims*)

Freeze primitive internal coordinates.

Parameters

typed_prims (*iterable of typed primitives*) -- Iterable containing typed_primitives, starting with a PrimType and followed by atom indices.

get_active_set(*B*, *inv_thresh=None*)

See [5] between Eq. (7) and Eq. (8) for advice regarding the threshold.

get_constrained_U(*constraint_vecs*)

project_hessian(*H*)

As we already work in the non-redundant subspace we don't have to project/shift the hessian as we do it for simple redundant internal coordinates.

project_primitive_on_active_set(*prim_ind*)

reset_constraints()

```

set_active_set()

transform_hessian(cart_hessian, int_gradient=None)
    Transform Cartesian Hessian to DLC.

transform_int_step(step, *args, **kwargs)
    As the transformation is done in primitive internal coordinates we convert the DLC back to primitive co-ordinates.

class pysisyphus.intcoords.DistanceFunction(indices, coeff=-1, **kwargs)
    Bases: Primitive

class pysisyphus.intcoords.DummyImproper(indices, *args, fix_inner=True, **kwargs)
    Bases: Torsion

    static get_coords3d_and_indices_ext(coords3d, indices)

class pysisyphus.intcoords.DummyTorsion(indices, *args, fix_inner=True, **kwargs)
    Bases: Torsion

    static get_coords3d_and_indices_ext(coords3d, indices)

    static get_fourth_coords(coords3d, indices, r=1.889, theta=90)

    
$$\frac{\mathbf{M} \mathbf{N} \cdot \mathbf{u} \mathbf{v}}{P}$$

    m, o, p, n = indices

class pysisyphus.intcoords.HDLC(*args, **kwargs)
    Bases: DLC

class pysisyphus.intcoords.HybridRedundantCoords(*args, **kwargs)
    Bases: RedundantCoords

class pysisyphus.intcoords.LinearBend(*args, complement=False, **kwargs)
    Bases: Primitive

    calculate(coords3d, indices=None, gradient=False)

    jacobian(coords3d, indices=None)

class pysisyphus.intcoords.LinearDisplacement(*args, complement=False, **kwargs)
    Bases: Primitive

    calculate(coords3d, indices=None, gradient=False)

    jacobian(coords3d, indices=None)

class pysisyphus.intcoords.MWCartesianCoords(*args, **kwargs)
    Bases: CartesianCoords

```

class pysisyphus.intcoords.**OutOfPlane**(*indices, periodic=False, calc_kwargs=None, cache=False*)

Bases: *Primitive*

[1] [https://doi.org/10.1002/\(SICI\)1096-987X\(19990730\)20:10<1067::AID-JCC9>3.0.CO;2-V](https://doi.org/10.1002/(SICI)1096-987X(19990730)20:10<1067::AID-JCC9>3.0.CO;2-V)

Lee, 1999

exception pysisyphus.intcoords.**PrimitiveNotDefinedException**(*typed_prim, *args, **kwargs*)

Bases: *Exception*

class pysisyphus.intcoords.**RedundantCoords**(*atoms, coords3d, masses=None, bond_factor=1.3, typed_prims=None, define_prims=None, constrain_prims=None, freeze_atoms=None, freeze_atoms_exclude=False, internals_with_frozen=False, define_for=None, bonds_only=False, check_bends=True, rebuild=True, bend_min_deg=15, dihed_max_deg=175, lb_min_deg=175, weighted=False, min_weight=0.3, svd_inv_thresh=0.000316, recalc_B=False, tric=False, hybrid=False, hbond_angles=False, rm_for_frag=None*)

Bases: *object*

property **B**

Wilson B-Matrix

property **B_inv**

Generalized inverse of the Wilson B-Matrix.

property **B_inv_prim**

Generalized inverse of the primitive Wilson B-Matrix.

property **B_prim**

Wilson B-Matrix

property **Bt_inv**

Transposed generalized inverse of the Wilson B-Matrix.

property **Bt_inv_prim**

Transposed generalized inverse of the primitive Wilson B-Matrix.

property **C**

Diagonal matrix. Entries for constraints are set to one.

property **P**

Projection matrix onto B. See [1] Eq. (4).

backtransform_hessian(*redund_hessian, int_gradient=None*)

Transform Hessian in internal coordinates to Cartesians.

property **bend_atom_indices**

property **bend_indices**

property **bond_atom_indices**

property **bond_indices**

property **bond_typed_prims**

property **cartesian_indices**

`clear()`

`property constrained_indices`

`property coords`

`property coords3d`

`property dihedral_atom_indices`

`property dihedral_indices`

`eval(coords3d, attr=None)`

`get_K_matrix(int_gradient=None)`

`get_index_of_typed_prim(typed_prim)`
Index in self.typed_prims for the supplied typed_prim.

`get_prim_internals_by_indices(indices)`

`inv_B(B)`

`inv_Bt(B)`

`property linear_bend_indices`

`log(message)`

`log_int_grad_msg(int_gradient)`

`property outofplane_indices`

`property prim_coords`

`property prim_indices_set`

`property prim_internals`

`property primitives`

`print_typed_prims()`

`project_hessian(H, shift=1000)`
Expects a hessian in internal coordinates. See Eq. (11) in [1].

`project_vector(vector)`
Project supplied vector onto range of B.

`return_inds(slice_)`

`property rotation_indices`

`set_inds_from_typed_prims(typed_prims)`

`set_primitive_indices(atoms, coords3d)`

`transform_forces(cart_forces)`
Combination of Eq. (9) and (11) in [1].

```
transform_hessian(cart_hessian, int_gradient=None)
    Transform Cartesian Hessian to internal coordinates.

transform_int_step(int_step, update_constraints=False, pure=False)

property translation_indices

property typed_prims

class pysisyphus.intcoords.RobustTorsion1(indices, periodic=False, calc_kwargs=None, cache=False)
    Bases: Primitive

class pysisyphus.intcoords.RobustTorsion2(indices, periodic=False, calc_kwargs=None, cache=False)
    Bases: Primitive

class pysisyphus.intcoords.RotationA(indices, *args, ref_coords3d, **kwargs)
    Bases: Rotation

    index = 0

class pysisyphus.intcoords.RotationB(indices, *args, ref_coords3d, **kwargs)
    Bases: Rotation

    index = 1

class pysisyphus.intcoords.RotationC(indices, *args, ref_coords3d, **kwargs)
    Bases: Rotation

    index = 2

class pysisyphus.intcoords.Stretch(indices, periodic=False, calc_kwargs=None, cache=False)
    Bases: Primitive

class pysisyphus.intcoords.TMTRIC(atoms, *args, **kwargs)
    Bases: TRIC

class pysisyphus.intcoords.TRIC(*args, **kwargs)
    Bases: RedundantCoords

class pysisyphus.intcoords.Torsion(indices, periodic=False, calc_kwargs=None, cache=False)
    Bases: Primitive

class pysisyphus.intcoords.Torsion2(indices, periodic=False, calc_kwargs=None, cache=False)
    Bases: Torsion

class pysisyphus.intcoords.TranslationX(*args, **kwargs)
    Bases: Translation

    cart_axis = 0

class pysisyphus.intcoords.TranslationY(*args, **kwargs)
    Bases: Translation

    cart_axis = 1

class pysisyphus.intcoords.TranslationZ(*args, **kwargs)
    Bases: Translation

    cart_axis = 2
```

17.1.1.8 pysisyphus.interpolate package

Submodules

pysisyphus.interpolate.IDPP module

```
class pysisyphus.interpolate.IDPP.IDPP(geoms, between, extrapolate=0, extrapolate_before=0,  
                                         extrapolate_after=0, extrapolate_damp=1.0, align=False)
```

Bases: [Interpolator](#)

```
interpolate(initial_geom, final_geom, **kwargs)
```

pysisyphus.interpolate.Interpolator module

```
class pysisyphus.interpolate.Interpolator.Interpolator(geoms, between, extrapolate=0,  
                                                         extrapolate_before=0, extrapolate_after=0,  
                                                         extrapolate_damp=1.0, align=False)
```

Bases: object

```
all_geoms_to_trj(trj_fn)
```

```
interpolate(initial_geom, final_geom, interpolate_only=0, extrapolate=False)
```

```
interpolate_all()
```

pysisyphus.interpolate.LST module

```
class pysisyphus.interpolate.LST.LST(*args, align=True, gtol=0.0001, silent=False, **kwargs)
```

Bases: [Interpolator](#)

```
cost_function(wa_c, f, rab, wab)
```

```
interpolate(initial_geom, final_geom, **kwargs)
```

pysisyphus.interpolate.Redund module

```
class pysisyphus.interpolate.Redund.Redund(*args, align=True, rebuild_geoms=True, **kwargs)
```

Bases: [Interpolator](#)

```
dump_progress(geoms, out_fn='redund_interpol_fail.trj')
```

```
interpolate(initial_geom, final_geom, interpolate_only=0, extrapolate=False, typed_prims=None)
```

```
restart_interpolate(geoms, new_geom, interpolate_kwargs)
```

```
step_along_tangent(geom, prim_tangent, step_size)
```

pysisyphus.interpolate.helpers module

pysisyphus.interpolate.helpers.**interpolate**(*initial_geom*, *final_geom*, *between*, *kind*='linear',
only_between=False, *align*=False, *interpol_kwargs*=None)

pysisyphus.interpolate.helpers.**interpolate_all**(*geoms*, *between*, *kind*='linear', *align*=False,
***interpol_kwargs*)

Module contents

17.1.1.9 pysisyphus.io package

Submodules

pysisyphus.io.cjson module

pysisyphus.io.cjson.**geom_from_cjson**(*fn*, ***kwargs*)

pysisyphus.io.cjson.**parse_cjson**(*fn*)

pysisyphus.io.crd module

pysisyphus.io.crd.**a8**(*str_*)

pysisyphus.io.crd.**atoms_coords_to_crd_str**(*atoms*, *coords*, *resno*=1, *res*='UNL1', *segid*=None, *resid*=1,
ref_atoms=None, *del_atoms*=None)

pysisyphus.io.crd.**f20**(*float_*)

pysisyphus.io.crd.**geom_from_crd**(*fn*, ***kwargs*)

pysisyphus.io.crd.**geom_to_crd_str**(*geom*, ***kwargs*)

pysisyphus.io.crd.**i10**(*int_*)

pysisyphus.io.hdf5 module

pysisyphus.io.hdf5.**get_h5_group**(*fn*, *group_name*, *data_model*=None, *reset*=False)

Return (and create if necessary) group with given name and data model.

pysisyphus.io.hdf5.**init_h5_group**(*f*, *group_name*, *data_model*)

Create group with given name and data model.

pysisyphus.io.hdf5.**resize_h5_group**(*group*, *max_cycles*)

Increase size of first dimension of datasets in the given group.

pysisyphus.io.hessian module

```
pysisyphus.io.hessian.geom_from_hessian(h5_fn, **geom_kwargs)
pysisyphus.io.hessian.save_hessian(h5_fn, geom, cart_hessian=None, energy=None, mult=None)
pysisyphus.io.hessian.save_third_deriv(h5_fn, geom, third_deriv_result, H_mw, H_proj)
```

pysisyphus.io.mol2 module

```
pysisyphus.io.mol2.dict_to_mol2_string(as_dict)
pysisyphus.io.mol2.geom_from_mol2(text, **kwargs)
pysisyphus.io.mol2.render_xyz(xyz)
```

pysisyphus.io.molden module

```
pysisyphus.io.molden.geoms_from_molden(fn, **kwargs)
pysisyphus.io.molden.parse_mo(mo_lines)
pysisyphus.io.molden.parse_molden_atoms(data)
```

pysisyphus.io.pdb module

```
pysisyphus.io.pdb.atoms_coords_to_pdb_str(atoms, coords, fragments=None, resname="", conect=True)
pysisyphus.io.pdb.geom_from_pdb(fn, **kwargs)
pysisyphus.io.pdb.geom_to_pdb_str(geom, detect_fragments=False, **kwargs)
pysisyphus.io.pdb.get_conect_lines(atoms, coords)
pysisyphus.io.pdb.get_parser(widths)
pysisyphus.io.pdb.parse_atom_name(name)
```

pysisyphus.io.pubchem module

```
pysisyphus.io.pubchem.cid_from_name(name)
pysisyphus.io.pubchem.geom_from_pubchem_name(name, **kwargs)
pysisyphus.io.pubchem.sdf_from_cid(cid)
```

pysisyphus.io.sdf module

pysisyphus.io.sdf.**geom_from_sdf**(text, **kwargs)

pysisyphus.io.sdf.**parse_coord_line**(line)

pysisyphus.io.sdf.**parse_sdf**(text)

pysisyphus.io.xyz module

pysisyphus.io.xyz.**geom_from_xyz**(fn, **kwargs)

pysisyphus.io.xyz.**geoms_from_inline_xyz**(inline_xyz, **kwargs)

pysisyphus.io.xyz.**geoms_from_xyz**(fn, **kwargs)

pysisyphus.io.zmat module

class pysisyphus.io.zmat.**ZLine**(atom, rind, r, aind, a, dind, d)

Bases: tuple

a

Alias for field number 4

aind

Alias for field number 3

atom

Alias for field number 0

d

Alias for field number 6

dind

Alias for field number 5

r

Alias for field number 2

rind

Alias for field number 1

pysisyphus.io.zmat.**geom_from_zmat**(zmat, atoms=None, coords3d=None, geom=None, start_at=None, drop_dummy=True, **geom_kwargs)

Adapted from <https://github.com/robashaw/geomConvert> by Robert Shaw.

pysisyphus.io.zmat.**geom_from_zmat_fn**(fn, **geom_kwargs)

pysisyphus.io.zmat.**geom_from_zmat_str**(text, coord_type='cart', coord_kwargs=None)

pysisyphus.io.zmat.**zmat_from_fn**(fn)

pysisyphus.io.zmat.**zmat_from_str**(text)

Module contents

```

pysisyphus.io.geom_from_cjson(fn, **kwargs)
pysisyphus.io.geom_from_crd(fn, **kwargs)
pysisyphus.io.geom_from_cube(inp, *args, **kwargs)
pysisyphus.io.geom_from_fchk(inp, *args, **kwargs)
pysisyphus.io.geom_from_hessian(h5_fn, **geom_kwargs)
pysisyphus.io.geom_from_mol2(text, **kwargs)
pysisyphus.io.geom_from_pdb(fn, **kwargs)
pysisyphus.io.geom_from_pubchem_name(name, **kwargs)
pysisyphus.io.geom_from_qcschema(qcschema, **geom_kwargs)
pysisyphus.io.geom_from_zmat(zmat, atoms=None, coords3d=None, geom=None, start_at=None,
                             drop_dummy=True, **geom_kwargs)
    Adapted from https://github.com/robashaw/geomConvert by Robert Shaw.
pysisyphus.io.geoms_from_molden(fn, **kwargs)
pysisyphus.io.geoms_from_xyz(fn, **kwargs)
pysisyphus.io.save_hessian(h5_fn, geom, cart_hessian=None, energy=None, mult=None)
pysisyphus.io.save_third_deriv(h5_fn, geom, third_deriv_result, H_mw, H_proj)

```

17.1.1.10 pysisyphus.irc package

Submodules

pysisyphus.irc.DWI module

```

class pysisyphus.irc.DWI.DWI(n=4, maxlen=2)
    Bases: object
    __init__(n=4, maxlen=2)
        Distance weighted interpolation.
    dump(fn)
    static from_h5(fn)
    interpolate(at_coords, gradient=False)
        See [1] Eq. (25) - (29)
    update(coords, energy, gradient, hessian)

pysisyphus.irc.DWI.taylor(energy, gradient, hessian, step)
    Taylor series expansion of the energy to second order.

pysisyphus.irc.DWI.taylor_grad(gradient, hessian, step)
    Gradient of a Taylor series expansion of the energy to second order.

```

pysisyphus.irc.DampedVelocityVerlet module

```
class pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet(geometry, v0=0.04, dt0=0.5,
                                                                error_tol=0.003,
                                                                max_cycles=150, **kwargs)
```

Bases: [IRC](#)

damp_velocity(*velocity*)

estimate_error(*new_mw_coords*)

mw_grad_to_acc(*mw_grad*)

Takes care of the units for the mass-weighted gradient. Converts units of a mass-weighted gradient [Hartree/(Bohr*amu)] to units of acceleration [$\sqrt{\text{amu}} \cdot \text{Bohr}/\text{fs}^2$]. The 1e30 comes from converting second² to femto second².

prepare(*direction*)

step()

pysisyphus.irc.Euler module

```
class pysisyphus.irc.Euler.Euler(geometry, step_length=0.01, **kwargs)
```

Bases: [IRC](#)

step()

pysisyphus.irc.EulerPC module

```
class pysisyphus.irc.EulerPC.EulerPC(*args, hessian_recalc=None, hessian_update='bofill',
                                       hessian_init='calc', max_pred_steps=500, loose_cycles=3,
                                       dump_dwi=False, scipy_method=None, corr_func='mbs',
                                       **kwargs)
```

Bases: [IRC](#)

corrector_step(*init_mw_coords*, *step_length*, *dwi*)

get_integration_length_func(*init_mw_coords*)

prepare(**args*, ***kwargs*)

scipy_corrector_step(*init_mw_coords*, *step_length*, *dwi*)

Solve IRC equation $dx/ds = -g/|g|$ on DWI PES in mass-weighted coordinates. Integration done until self.step_length in unweighted coordinates is achieved.

step()

pysisyphus.irc.GonzalezSchlegel module

```
class pysisyphus.irc.GonzalezSchlegel.GonzalezSchlegel(geometry, max_micro_cycles=20,
                                                    micro_step_thresh=0.001,
                                                    hessian_recalc=None, line_search=False,
                                                    **kwargs)
```

Bases: [IRC](#)

micro_step(counter)

Constrained optimization on a hypersphere.

perp_component(vec, perp_to)

postprocess()

step()

pysisyphus.irc.IMKMod module

```
class pysisyphus.irc.IMKMod.IMKMod(geometry, corr_first=True, corr_first_size=0.5,
                                   corr_first_energy=True, corr_bisec_size=0.0025,
                                   corr_bisec_energy=True, **kwargs)
```

Bases: [IRC](#)

fit_grad_and_energies(energy0, grad0, energy1, direction)

fit_parabola(x, y)

step()

pysisyphus.irc.IRC module

```
class pysisyphus.irc.IRC.IRC(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True,
                              backward=True, root=0, hessian_init=None, displ='energy',
                              displ_energy=0.001, displ_length=0.1, displ_third_h5=None,
                              rms_grad_thresh=0.001, hard_rms_grad_thresh=None, energy_thresh=1e-06,
                              imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',
                              prefix='', dump_fn='irc_data.h5', dump_every=5)
```

Bases: object

```
__init__(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True, backward=True,
         root=0, hessian_init=None, displ='energy', displ_energy=0.001, displ_length=0.1,
         displ_third_h5=None, rms_grad_thresh=0.001, hard_rms_grad_thresh=None,
         energy_thresh=1e-06, imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',
         prefix='', dump_fn='irc_data.h5', dump_every=5)
```

Base class for IRC calculations.

Parameters

- **geometry** ([Geometry](#)) -- Transtion state geometry, or initial geometry for downhill run.
- **step_length** (*float*, *optional*) -- Step length in unweighted coordinates.
- **max_cycles** (*int*, *optional*) -- Positive integer, controllong the maximum number of IRC steps taken in a direction (forward/backward/downhill).

- **downhill** (*bool*, *default=False*) -- Downhill run from a non-stationary point with non-vanishing gradient. Disables forward and backward runs.
- **forward** (*bool*, *default=True*) -- Integrate IRC in positive *s* direction.
- **backward** (*bool*, *default=True*) -- Integrate IRC in negative *s* direction.
- **root** (*int*, *default=0*) -- Use *n*-th root for initial displacement from TS.
- **hessian_init** (*str*, *default=None*) -- Path to Hessian HDF5 file, e.g., from a previous TS calculation.
- **displ** (*str*, *one of ("energy", "length", "energy_cubic")*) -- Controls initial displacement from the TS. 'energy' assumes a quadratic model, from which a step length for a given energy lowering (see 'displ_energy') is determined. 'length' corresponds to a displacement along the transition vector. 'energy_cubic' considers 3rd derivatives of the energy along the transition vector.
- **displ_energy** (*float*, *default=1e-3*) -- Required energy lowering from the TS in au (Hartree). Used with 'displ: energy|energy_cubic'.
- **displ_length** (*float*, *default=0.1*) -- Step length along the transition vector. Used only with 'displ: length'.
- **displ_third_h5** (*str*, *optional*) -- Path to HDF5 file containing 3rd derivative information. Used with 'displ: energy_cubic'.
- **rms_grad_thresh** (*float*, *default=1e-3*,) -- Convergence is signalled when to root mean square of the unweighted gradient is less than or equal to this value
- **energy_thresh** (*float*, *default=1e-6*,) -- Signal convergence when the energy difference between two points is equal to or less than 'energy_thresh'.
- **imag_below** (*float*, *default=0.0*) -- Require the wavenumber of the imaginary mode to be below the given threshold. If given, it should be a negative number.
- **force_inflection** (*bool*, *optional*) -- Don't indicate convergence before passing an inflection point.
- **check_bonds** (*bool*, *optional*, *default=True*) -- Report whether bonds are formed/broken along the IRC, w.r.t the TS.
- **out_dir** (*str*, *optional*) -- Dump everything into 'out_dir' directory instead of the CWD.
- **prefix** (*str*, *optional*) -- Short string that is prepended to all files that are created by this class, e.g., trajectories and HDF5 dumps.
- **dump_fn** (*str*, *optional*) -- Base name for the HDF5 files.
- **dump_every** (*int*, *optional*) -- Dump to HDF5 every *n*-th cycle.

property coords

dump_data(*dump_fn=None*, *full=False*)

dump_ends(*path*, *prefix*, *coords=None*, *trj=False*)

property energy

get_conv_fact(*mw_grad*, *min_fact=2.0*)

get_endpoint_and_ts_geoms()

get_full_irc_data()

get_irc_data()

get_path_for_fn(*fn*)

property gradient

initial_displacement()

Returns non-mass-weighted steps in +s and -s direction for initial displacement from the TS. Earlier version only returned one step, that was later multiplied by either 1 or -1, depending on the desired IRC direction (forward/backward). The current implementation directly returns two steps for forward and backward direction. Whereas for plus and minus steps for displ 'length' and displ 'energy'

$\text{step_plus} = -\text{step_minus}$

is valid, it is not valid for displ 'energy_cubic' anymore. The latter step is formed as

$x(ds) = ds * v0 + ds**2 * v1$

so

$x(ds) \neq -x(ds)$

as

$ds * v0 + ds**2 * v1 \neq -ds * v0 - ds**2 * v1$.

So, all required step are formed directly and later used as appropriate.

See

<https://aip.scitation.org/doi/pdf/10.1063/1.454172>

<https://pubs.acs.org/doi/10.1021/j100338a027>

<https://aip.scitation.org/doi/pdf/10.1063/1.459634>

irc(*direction*)

log(*msg*)

property m_sqrt

mass_weigh_hessian(*hessian*)

property mw_coords

property mw_gradient

postprocess()

prepare(*direction*)

report_bonds(*prefix, bonds*)

report_conv_thresholds()

run()

set_data(*prefix*)

unweight_vec(*vec*)

valid_displs = ('energy', 'length', 'energy_cubic')

pysisyphus.irc.IRCDummy module

```
class pysisyphus.irc.IRCDummy.IRCDummy(all_coords, atoms, forward=True, backward=True, downhill=False)
```

Bases: object

all_coords: list

atoms: tuple

backward: bool = True

downhill: bool = False

forward: bool = True

pysisyphus.irc.Instanton module

```
class pysisyphus.irc.Instanton.Instanton(images, calc_getter, T)
```

Bases: object

property P

P_bh

the first image is connected to the last image. At k=0 the index k-1 will be -1, which points to the last image.

Below we pre-calculate some indices (assuming N images).

unshifted indices ks: k = {0, 1, .. , N-1} shifted indices ksm1: k-1 = {-1, 0, 1, .. , N-2} shifted indices ksp1: k+1 = {1, 2, .. , N-1, 0}

Type

The Instanton is periodic

action()

Action in au / fs, Hartree per femtosecond.

action_gradient()

kin_grad corresponds to the gradient of S_0 in (Eq. 1 in [1], or first term in Eq. 6 in [2].) It boils down to the derivative of a sum of vector norms

$$d \sum_k (\|y_k - y_{(k-1)}\|_2)^2 \text{ --- } d y_k$$

The derivative of a norm of a vector difference is quite simple, but care has to be taken to recognize, that y_k appears two times in the sum. It appears in the first summand for k and in the second summand for k+1.

$$\sum_k (\|y_k - y_{(k-1)}\|_2)^2$$

1. term 2. term

$$= (\|y_k - y_{(k-1)}\|_2)^2 + (\|y_{(k+1)} - y_k\|_2)^2 + \dots \text{ and so on}$$

The derivative of the first term is

$$2 * (y_k - y_{(k-1)})$$

and the derivative of the second term is

$$-2 * (y_{(k+1)} - y_k)$$

which is equal to

$$2 * (y_k - y_{(k+1)}) .$$

To summarize:

$$\begin{aligned} & d \sum_k (\|y_k - y_{(k-1)}\|_2)^2 \text{ --- } d y_k \\ & = 2 * (2 * y_k - y_{(k-1)} - y_{(k+1)}) . \end{aligned}$$

action_hessian()

as_xyz()

property cart_coords

property cart_forces

property cart_hessian

property coords

property energy

property forces

classmethod from_instanton(*other*, ****kwargs**)

classmethod from_ts(*ts_geom*, *P*, *dr=0.4*, *delta_T=25*, *cart_hessian=None*, ****kwargs**)

get_additional_print()

property gradient

property hessian

is_analytical_2d()

property path_length

`pysisyphus.irc.Instanton.T_crossover_from_eigval(eigval)`

`pysisyphus.irc.Instanton.T_crossover_from_ts(ts_geom)`

`pysisyphus.irc.Instanton.log_progress(val, key, i)`

pysisyphus.irc.LQA module

class pysisyphus.irc.LQA.LQA(*geometry*, *N_euler=5000*, ****kwargs**)

Bases: [IRC](#)

step()

pysisyphus.irc.ModeKill module

```
class pysisyphus.irc.ModeKill.ModeKill(geometry, kill_inds, nu_thresh=-5.0, do_hess=True,  
                                         hessian_update=True, **kwargs)
```

Bases: [IRC](#)

get_additional_print()

postprocess()

prepare(*args, **kwargs)

step()

update_mw_down_step()

pysisyphus.irc.ParamPlot module

```
class pysisyphus.irc.ParamPlot.ParamPlot(coords_list, param1_inds, param2_inds)
```

Bases: object

calc_angle(coords, ind1, ind2, ind3)

calc_bond(coords, ind1, ind2)

get_param(coords, param_inds)

plot()

save(path, prefix)

save_coords(path, prefix)

show()

pysisyphus.irc.RK4 module

```
class pysisyphus.irc.RK4.RK4(geometry, step_length=0.1, max_cycles=125, downhill=False, forward=True,  
                              backward=True, root=0, hessian_init=None, displ='energy',  
                              displ_energy=0.001, displ_length=0.1, displ_third_h5=None,  
                              rms_grad_thresh=0.001, hard_rms_grad_thresh=None, energy_thresh=1e-06,  
                              imag_below=0.0, force_inflection=True, check_bonds=True, out_dir='.',  
                              prefix='', dump_fn='irc_data.h5', dump_every=5)
```

Bases: [IRC](#)

get_k(mw_coords)

step()

pysisyphus.irc.initial_displ module

```
class pysisyphus.irc.initial_displ.ThirdDerivResult(coords_plus, energy_plus, H_plus,
                                                    coords_minus, energy_minus, H_minus, G_vec,
                                                    vec, ds)
```

Bases: tuple

G_vec

Alias for field number 6

H_minus

Alias for field number 5

H_plus

Alias for field number 2

coords_minus

Alias for field number 3

coords_plus

Alias for field number 0

ds

Alias for field number 8

energy_minus

Alias for field number 4

energy_plus

Alias for field number 1

vec

Alias for field number 7

```
pysisyphus.irc.initial_displ.cubic_displ(H, v0, w0, Gv, dE)
```

According to Eq. (26) in [2] v_1 does not depend on the sign of v_0 .

$$v_1 = (F_0 - 2v_0^T F_0 v_0 I)^1 \times ([G_0 v_0] - v_0^T [G_0 v_0] v_0 I) v_0$$

The first term is obviously independent of v_0 's sign. Using $v_0' = -v_0$ the second term becomes

$$([G_0 v_0'] - v_0'^T [G_0 v_0'] v_0' I) v_0' - ([G_0 v_0] - v_0^T [G_0 v_0] v_0 I) v_0' - ([G_0 v_0] + v_0^T [G_0 v_0] v_0 I) v_0' - (-[G_0 v_0] + v_0^T [G_0 v_0] v_0 I) v_0 ([G_0 v_0] - v_0^T [G_0 v_0] v_0 I) v_0$$

Strictly speaking the contraction $[G_0 v_0]$ should depend on the sign of v_0 . In the current implementation, the direction of v_0 is not taken into account, but

$$\text{get_curv_vec}(H, Gv, v_0, w_0) == \text{get_curv_vec}(H, -Gv, -v_0, w_0) .$$

But somehow the Taylor expansion gives bogus results when called with $-Gv$ and $-v_0$...

```
pysisyphus.irc.initial_displ.cubic_displ_for_geom(geom, dE=-0.0005)
```

```
pysisyphus.irc.initial_displ.cubic_displ_for_h5(h5_fn='third_deriv.h5', dE=-0.0005)
```

```
pysisyphus.irc.initial_displ.get_curv_vec(H, Gv, v0, w0)
```

`pysisyphus.irc.initial_displ.taylor_closure(H, Gv, v0, v1, w0)`

Taylor expansion of energy to 3rd order.

$$dx(ds) = ds*v0 + ds**2*v1 / 2 \quad dE(ds) = dx^T H dx / 2 + dx^T [Gv] dx / 6$$

H = Hessian Gv = 3rd derivative of energy along v0 v0 = Transition vector v1 = Curvature vector w0 = Eigenvalue belonging to v0

`pysisyphus.irc.initial_displ.third_deriv_fd(geom, vec, ds=0.001)`

Third derivative of the energy in direction 'vec'.

Module contents

`class pysisyphus.irc.DampedVelocityVerlet(geometry, v0=0.04, dt0=0.5, error_tol=0.003, max_cycles=150, **kwargs)`

Bases: [IRC](#)

`damp_velocity(velocity)`

`estimate_error(new_mw_coords)`

`mw_grad_to_acc(mw_grad)`

Takes care of the units for the mass-weighted gradient. Converts units of a mass-weighted gradient [Hartree/(Bohr*amu)] to units of acceleration [sqrt(amu)*Bohr/fs²]. The 1e30 comes from converting second² to femto second².

`prepare(direction)`

`step()`

`class pysisyphus.irc.Euler(geometry, step_length=0.01, **kwargs)`

Bases: [IRC](#)

`step()`

`class pysisyphus.irc.EulerPC(*args, hessian_recalc=None, hessian_update='bofill', hessian_init='calc', max_pred_steps=500, loose_cycles=3, dump_dwi=False, scipy_method=None, corr_func='mbs', **kwargs)`

Bases: [IRC](#)

`corrector_step(init_mw_coords, step_length, dwi)`

`get_integration_length_func(init_mw_coords)`

`prepare(*args, **kwargs)`

`scipy_corrector_step(init_mw_coords, step_length, dwi)`

Solve IRC equation $dx/ds = -g/|g|$ on DWI PES in mass-weighted coordinates. Integration done until self.step_length in unweighted coordinates is achieved.

`step()`

`class pysisyphus.irc.GonzalezSchlegel(geometry, max_micro_cycles=20, micro_step_thresh=0.001, hessian_recalc=None, line_search=False, **kwargs)`

Bases: [IRC](#)

micro_step(*counter*)

Constrained optimization on a hypersphere.

perp_component(*vec*, *perp_to*)

postprocess()

step()

class pysisyphus.irc.**IMKMod**(*geometry*, *corr_first*=True, *corr_first_size*=0.5, *corr_first_energy*=True, *corr_bisec_size*=0.0025, *corr_bisec_energy*=True, ***kwargs*)

Bases: [IRC](#)

fit_grad_and_energies(*energy0*, *grad0*, *energy1*, *direction*)

fit_parabola(*x*, *y*)

step()

class pysisyphus.irc.**LQA**(*geometry*, *N_euler*=5000, ***kwargs*)

Bases: [IRC](#)

step()

class pysisyphus.irc.**ModeKill**(*geometry*, *kill_inds*, *nu_thresh*=-5.0, *do_hess*=True, *hessian_update*=True, ***kwargs*)

Bases: [IRC](#)

get_additional_print()

postprocess()

prepare(**args*, ***kwargs*)

step()

update_mw_down_step()

class pysisyphus.irc.**RK4**(*geometry*, *step_length*=0.1, *max_cycles*=125, *downhill*=False, *forward*=True, *backward*=True, *root*=0, *hessian_init*=None, *displ*='energy', *displ_energy*=0.001, *displ_length*=0.1, *displ_third_h5*=None, *rms_grad_thresh*=0.001, *hard_rms_grad_thresh*=None, *energy_thresh*=1e-06, *imag_below*=0.0, *force_inflection*=True, *check_bonds*=True, *out_dir*='.', *prefix*='', *dump_fn*='irc_data.h5', *dump_every*=5)

Bases: [IRC](#)

get_k(*mw_coords*)

step()

17.1.1.11 pysisyphus.line_searches package

Submodules

pysisyphus.line_searches.Backtracking module

```
class pysisyphus.line_searches.Backtracking.Backtracking(*args, rho_lo=0.05, rho_hi=0.9,
                                                         use_grad=False, **kwargs)
```

Bases: [LineSearch](#)

```
__init__(*args, rho_lo=0.05, rho_hi=0.9, use_grad=False, **kwargs)
```

Backtracking line search enforcing Armijo conditions.

Uses only energy evaluations.

See [1], Chapter 3, Line Search methods, Section 3.1 p. 31 and Section 3.5 p. 56.

```
alpha_new_from_phi(cycle, phi0, dphi0, alpha, alpha_prev)
```

```
alpha_new_from_phi_dphi(cycle, phi0, dphi0, alpha)
```

```
run_line_search()
```

pysisyphus.line_searches.HagerZhang module

```
class pysisyphus.line_searches.HagerZhang.HagerZhang(*args, alpha_prev=None, f_prev=None,
                                                       dphi0_prev=None, quad_step=False,
                                                       eps=1e-06, theta=0.5, gamma=0.5, rho=5,
                                                       psi_0=0.01, psi_1=0.1, psi_2=2.0,
                                                       psi_low=0.1, psi_hi=10, Delta=0.7,
                                                       omega=0.001, max_bisects=10, **kwargs)
```

Bases: [LineSearch](#)

```
bisect(a, b)
```

Bisect interval [a, b].

```
bracket(c)
```

Generate initial interval [a, b] that satisfies the opposite slope condition ($d\phi(a) < 0$, $d\phi(b) > 0$).

```
double_secant(a, b)
```

Take secant² step.

```
initial()
```

Get an initial guess for alpha.

```
interval_update(a, b, c)
```

Narrows down the bracketing interval.

```
norm_inf(arr)
```

Returns infinity norm of given array.

```
prepare_line_search()
```

```
run_line_search()
```

secant(*a*, *b*)

Take secant step.

take_quad_step(*alpha*, *g0_*)

Try to get alpha for minimum step from quadratic interpolation.

pysisyphus.line_searches.LineSearch module

```
class pysisyphus.line_searches.LineSearch.LineSearch(p, cond='armijo', x0=None, geometry=None,
                                                    f=None, df=None, alpha_init=None, f0=None,
                                                    g0=None, c1=0.1, c2=0.9, max_cycles=10,
                                                    alpha_min=1e-06)
```

Bases: object

check_alpha(*alpha*)

curvature_condition(*alpha*)

get_fg(*what*, *alpha*)

Lookup raw function/gradient values for a given alpha.

get_phi_dphi(*what*, *alpha*, *check*=True)

Wrapper that handles function/gradient evaluations.

got_alpha_phi_dphi(*alpha*)

log(*message*)

prepare_line_search()

run()

abstract run_line_search()

strong_curvature_condition(*alpha*)

strong_wolfe_condition(*alpha*)

Strong wolfe condition.

sufficiently_decreased(*alpha*)

Sufficient decrease/Armijo condition.

wolfe_condition(*alpha*)

Normal, not strong, Wolfe condition.

```
exception pysisyphus.line_searches.LineSearch.LineSearchConverged(alpha)
```

Bases: Exception

```
exception pysisyphus.line_searches.LineSearch.LineSearchNotConverged
```

Bases: Exception

```
class pysisyphus.line_searches.LineSearch.LineSearchResult(converged, alpha, f_new, g_new,
                                                         f_evals, df_evals, dphi0)
```

Bases: tuple

alpha

Alias for field number 1

converged

Alias for field number 0

df_evals

Alias for field number 5

dphi0

Alias for field number 6

f_evals

Alias for field number 4

f_new

Alias for field number 2

g_new

Alias for field number 3

pysisyphus.line_searches.StrongWolfe module

```
class pysisyphus.line_searches.StrongWolfe.StrongWolfe(*args, alpha_max=10.0, fac=2, **kwargs)
```

Bases: [LineSearch](#)

```
__init__(*args, alpha_max=10.0, fac=2, **kwargs)
```

Wolfe line search.

Uses only energy & gradient evaluations.

See [1], Chapter 3, Line Search methods, Section 3.5 p. 60.

```
run_line_search()
```

```
zoom(alpha_lo, alpha_hi, phi_lo, phi_alpha_=None, alpha_0_=None, max_cycles=10)
```

pysisyphus.line_searches.interpol module

```
pysisyphus.line_searches.interpol.interpol_alpha_cubic(f_0, df_0, f_alpha_0, f_alpha_1, alpha_0,  
alpha_1)
```

```
pysisyphus.line_searches.interpol.interpol_alpha_quad(f_0, df_0, f_alpha_0, alpha_0)
```

Module contents

```
class pysisyphus.line_searches.Backtracking(*args, rho_lo=0.05, rho_hi=0.9, use_grad=False,  
**kwargs)
```

Bases: [LineSearch](#)

```
__init__(*args, rho_lo=0.05, rho_hi=0.9, use_grad=False, **kwargs)
```

Backtracking line search enforcing Armijo conditions.

Uses only energy evaluations.

See [1], Chapter 3, Line Search methods, Section 3.1 p. 31 and Section 3.5 p. 56.

alpha_new_from_phi(cycle, phi0, dphi0, alpha, alpha_prev)

alpha_new_from_phi_dphi(cycle, phi0, dphi0, alpha)

run_line_search()

class pysisyphus.line_searches.**HagerZhang**(*args, alpha_prev=None, f_prev=None, dphi0_prev=None, quad_step=False, eps=1e-06, theta=0.5, gamma=0.5, rho=5, psi_0=0.01, psi_l=0.1, psi_2=2.0, psi_low=0.1, psi_hi=10, Delta=0.7, omega=0.001, max_bisects=10, **kwargs)

Bases: [LineSearch](#)

bisect(a, b)

Bisect interval [a, b].

bracket(c)

Generate initial interval [a, b] that satisfies the opposite slope condition ($d\phi(a) < 0$, $d\phi(b) > 0$).

double_secant(a, b)

Take secant² step.

initial()

Get an initial guess for alpha.

interval_update(a, b, c)

Narrows down the bracketing interval.

norm_inf(arr)

Returns infinity norm of given array.

prepare_line_search()

run_line_search()

secant(a, b)

Take secant step.

take_quad_step(alpha, g0_)

Try to get alpha for minimum step from quadratic interpolation.

class pysisyphus.line_searches.**StrongWolfe**(*args, alpha_max=10.0, fac=2, **kwargs)

Bases: [LineSearch](#)

__init__(*args, alpha_max=10.0, fac=2, **kwargs)

Wolfe line search.

Uses only energy & gradient evaluations.

See [1], Chapter 3, Line Search methods, Section 3.5 p. 60.

run_line_search()

zoom(alpha_lo, alpha_hi, phi_lo, phi_alpha_=None, alpha_0_=None, max_cycles=10)

17.1.1.12 pysisyphus.modelfollow package

Submodules

pysisyphus.modelfollow.NormalMode module

class pysisyphus.modelfollow.NormalMode.**NormalMode**(*l*, *masses*)

Bases: object

See <http://gaussian.com/vib/>

__init__(*l*, *masses*)

NormalMode class.

Cartesian displacements are normalized to 1.

Parameters

- **l** (*np.array*) -- Cartesian, non-mass-weighted displacements.
- **masses** (*np.array*) -- Atomic masses.

property **l_mw**

mw_norm_for_norm(*norm=0.01*)

property **red_mass**

pysisyphus.modelfollow.davidson module

class pysisyphus.modelfollow.davidson.**DavidsonResult**(*cur_cycle*, *converged*, *final_modes*, *qs*, *nus*, *mode_inds*, *res_rms*)

Bases: tuple

converged

Alias for field number 1

cur_cycle

Alias for field number 0

final_modes

Alias for field number 2

mode_inds

Alias for field number 5

nus

Alias for field number 4

qs

Alias for field number 3

res_rms

Alias for field number 6

```
pysisyphus.modefollow.davidson.block_davidson(cart_coords, masses, forces_getter, guess_modes,  
                                                lowest=None, trial_step_size=0.01,  
                                                hessian_precon=None, max_cycles=25,  
                                                res_rms_thresh=0.0001, start_precon=5,  
                                                remove_trans_rot=True, print_level=1)
```

```
pysisyphus.modefollow.davidson.forces_fin_diff(forces_getter, coords, b, step_size)
```

```
pysisyphus.modefollow.davidson.geom_davidson(geom, *args, **kwargs)
```

pysisyphus.modefollow.lanczos module

```
pysisyphus.modefollow.lanczos.geom_lanczos(geom, *args, **kwargs)
```

Wraps Lanczos algorithm for use with Geometry objects.

```
pysisyphus.modefollow.lanczos.lanczos(coords, grad_getter, dx=0.005, dl=0.01, guess=None,  
                                       max_cycles=25, reortho=True, logger=None)
```

Lanczos method to determine smallest eigenvalue & -vector.

See [1] for description of algorithm.

Module contents

```
class pysisyphus.modefollow.NormalMode(l, masses)
```

Bases: object

See <http://gaussian.com/vib/>

```
__init__(l, masses)
```

NormalMode class.

Cartesian displacements are normalized to 1.

Parameters

- **l** (*np.array*) -- Cartesian, non-mass-weighted displacements.
- **masses** (*np.array*) -- Atomic masses.

```
property l_mw
```

```
mw_norm_for_norm(norm=0.01)
```

```
property red_mass
```

```
pysisyphus.modefollow.geom_davidson(geom, *args, **kwargs)
```

```
pysisyphus.modefollow.geom_lanczos(geom, *args, **kwargs)
```

Wraps Lanczos algorithm for use with Geometry objects.

17.1.1.13 pysisyphus.optimizers package

Submodules

pysisyphus.optimizers.BFGS module

class pysisyphus.optimizers.BFGS.BFGS(*geometry*, **args*, *update*='bfgs', ***kwargs*)

Bases: *Optimizer*

bfgs_update(*s*, *y*)

damped_bfgs_update(*s*, *y*, *mu_1*=0.2)

Damped BFGS update of inverse Hessian.

Potentially updates *s*. See Section 3.2 of [2], Eq. (30) - (33). There is a typo ;) It should be

$$H_{k+1} = V_k H_k V_k^T + \dots$$

instead of

$$H_{k+1} = V_k^T H_k V_k + \dots$$

double_damped_bfgs_update(*s*, *y*, *mu_1*=0.2, *mu_2*=0.2)

Double damped BFGS update of inverse Hessian.

See [3]. Potentially updates *s* and *y*.

property eye

optimize()

prepare_opt()

pysisyphus.optimizers.BacktrackingOptimizer module

class pysisyphus.optimizers.BacktrackingOptimizer.BacktrackingOptimizer(*geometry*, *alpha*,
bt_force=5,
dont_skip_after=2,
bt_max_scale=4,
bt_disable=False,
***kwargs*)

Bases: *Optimizer*

backtrack(*cur_forces*, *prev_forces*, *reset_hessian*=None)

Accelerated backtracking line search.

reset()

pysisyphus.optimizers.ConjugateGradient module

```
class pysisyphus.optimizers.ConjugateGradient.ConjugateGradient(geometry, alpha=0.1,  
                                                                formula='FR', dont_skip=True,  
                                                                **kwargs)
```

Bases: *BacktrackingOptimizer*

get_beta(cur_forces, prev_forces)

optimize()

prepare_opt()

reset()

pysisyphus.optimizers.CubicNewton module

```
class pysisyphus.optimizers.CubicNewton.CubicNewton(geometry, **kwargs)
```

Bases: *HessianOptimizer*

optimize()

postprocess_opt()

pysisyphus.optimizers.FIRE module

```
class pysisyphus.optimizers.FIRE.FIRE(geometry, dt=0.1, dt_max=1, N_acc=2, f_inc=1.1, f_acc=0.99,  
                                         f_dec=0.5, n_reset=0, a_start=0.1, **kwargs)
```

Bases: *Optimizer*

optimize()

reset()

pysisyphus.optimizers.HessianOptimizer module

```
class pysisyphus.optimizers.HessianOptimizer.HessianOptimizer(geometry, trust_radius=0.5,  
                                                                trust_update=True, trust_min=0.1,  
                                                                trust_max=1,  
                                                                max_energy_incr=None,  
                                                                hessian_update='bfgs',  
                                                                hessian_init='fischer',  
                                                                hessian_recalc=None,  
                                                                hessian_recalc_adapt=None,  
                                                                hessian_xtb=False,  
                                                                hessian_recalc_reset=False,  
                                                                small_eigval_thresh=1e-08,  
                                                                line_search=False, alpha0=1.0,  
                                                                max_micro_cycles=25,  
                                                                rfo_overlaps=False, **kwargs)
```

Bases: *Optimizer*

```
__init__(geometry, trust_radius=0.5, trust_update=True, trust_min=0.1, trust_max=1,
          max_energy_incr=None, hessian_update='bfgs', hessian_init='fischer', hessian_recalc=None,
          hessian_recalc_adapt=None, hessian_xtb=False, hessian_recalc_reset=False,
          small_eigval_thresh=1e-08, line_search=False, alpha0=1.0, max_micro_cycles=25,
          rfo_overlaps=False, **kwargs)
```

Baseclass for optimizers utilizing Hessian information.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.
- **trust_radius** (float, default: 0.5) -- Initial trust radius in whatever unit the optimization is carried out.
- **trust_update** (bool, default: True) -- Whether to update the trust radius throughout the optimization.
- **trust_min** (float, default: 0.1) -- Minimum trust radius.
- **trust_max** (float, default: 1) -- Maximum trust radius.
- **max_energy_incr** (Optional[float], default: None) -- Maximum allowed energy increased after a faulty step. Optimization is aborted when the threshold is exceeded.
- **hessian_update** (Literal['none', None, False, 'bfgs', 'damped_bfgs', 'flowchart', 'bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev'], default: 'bfgs') -- Type of Hessian update. Defaults to BFGS for minimizations and Bofill for saddle point searches.
- **hessian_init** (Literal['calc', 'unit', 'fischer', 'lindh', 'simple', 'swart', 'xtb', 'xtb1', 'xtbff'], default: 'fischer') -- Type of initial model Hessian.
- **hessian_recalc** (Optional[int], default: None) -- Recalculate exact Hessian every n-th cycle instead of updating it.
- **hessian_recalc_adapt** (Optional[float], default: None) -- Use a more flexible scheme to determine Hessian recalculation. Undocumented.
- **hessian_xtb** (bool, default: False) -- Recalculate the Hessian at the GFN2-XTB level of theory.
- **hessian_recalc_reset** (bool, default: False) -- Whether to skip Hessian recalculation after reset. Undocumented.
- **small_eigval_thresh** (float, default: 1e-08) -- Threshold for small eigenvalues. Eigenvectors belonging to eigenvalues below this threshold are discarded.
- **line_search** (bool, default: False) -- Whether to carry out a line search. Not implemented by a subclassing optimizers.
- **alpha0** (float, default: 1.0) -- Initial alpha for restricted-step (RS) procedure.
- **max_micro_cycles** (int, default: 25) -- Maximum number of RS iterations.
- **rfo_overlaps** (bool, default: False) -- Enable mode-following in RS procedure.
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

```
filter_small_eigvals(eigvals, eigvecs, mask=False)
```

```
get_alpha_step(cur_alpha, rfo_eigval, step_norm, eigvals, gradient)
```

```
get_augmented_hessian(eigvals, gradient, alpha=1.0)
```

```

static get_newton_step(eigvals, eigvecs, gradient)

get_newton_step_on_trust(eigvals, eigvecs, gradient, transform=True)
    Step on trust-radius.

    See Nocedal 4.3 Iterative solutions of the subproblem
get_rs_step(eigvals, eigvecs, gradient, name='RS')

static get_shifted_step_trans(eigvals, gradient_trans, shift)

get_step_func(eigvals, gradient, grad_rms_thresh=0.01)

housekeeping()
    Calculate gradient and energy. Update trust radius and hessian if needed. Return energy, gradient and
    hessian for the current cycle.

log_negative_eigenvalues(eigvals, pre_str='')

prepare_opt(hessian_init=None)

property prev_eigvec_max

property prev_eigvec_min

static quadratic_model(gradient, hessian, step)

reset()

rfo_dict = {'max': (-1, 'max'), 'min': (0, 'min')}

static rfo_model(gradient, hessian, step)

save_hessian()

set_new_trust_radius(coeff, last_step_norm)

solve_rfo(rfo_mat, kind='min', prev_eigvec=None)

update_hessian()

update_trust_radius()

pysisyphus.optimizers.HessianOptimizer.dummy_hessian_update(H, dx, dg)

```

pysisyphus.optimizers.LBFGS module

```

class pysisyphus.optimizers.LBFGS.LBFGS(geometry, keep_last=7, beta=1, max_step=0.2,
    double_damp=True, gamma_mult=False, line_search=False,
    mu_reg=None, max_mu_reg_adaptions=10, control_step=True,
    **kwargs)

Bases: Optimizer

```

```
__init__(geometry, keep_last=7, beta=1, max_step=0.2, double_damp=True, gamma_mult=False,
         line_search=False, mu_reg=None, max_mu_reg_adaptions=10, control_step=True, **kwargs)
```

Limited-memory BFGS optimizer.

See [1] Nocedal, Wright - Numerical Optimization, 2006 for a general discussion of LBFGS. See `pysisyphus.optimizers.hessian_updates` for the references related to double damping and `pysisyphus.optimizers.closures` for references related to regularized LBFGS.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.
- **keep_last** (int, default: 7) -- History size. Keep last 'keep_last' steps and gradient differences.
- **beta** (float, default: 1) -- Force constant in $-(H + I)^l g$.
- **max_step** (float, default: 0.2) -- Upper limit for the absolute component of the step vector in whatever unit the optimization is carried out.
- **double_damp** (bool, default: True) -- Use double damping procedure to modify steps s and gradient differences y to ensure $s^T y > 0$.
- **gamma_mult** (bool, default: False) -- Estimate γ from previous cycle. Eq. (7.20) in [1]. See 'beta' argument.
- **line_search** (bool, default: False) -- Enable implicit line searches.
- **mu_reg** (Optional[float], default: None) -- Initial guess for regularization constant in regularized LBFGS.
- **max_mu_reg_adaptions** (int, default: 10) -- Maximum number of trial steps in regularized LBFGS.
- **control_step** (bool, default: True) -- Whether to scale down the proposed step its biggest absolute component is equal to or below 'max_step'
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

```
get_lbfgs_step(forces)
```

```
optimize()
```

```
postprocess_opt()
```

```
reset()
```

pysisyphus.optimizers.LayerOpt module

```
class pysisyphus.optimizers.LayerOpt.LayerOpt(geometry, layers=None, **kwargs)
```

Bases: *Optimizer*

```
check_convergence(*args, **kwargs)
```

Check if we must use the model optimizer to signal convergence.

```
property layer_num: int
```

```
property model_opt
```

Return the persistent optimizer belonging to the model system. We don't have to supply any coordinates to the optimizer of the most expensive layer, as it is persistent, as well as the associated geometry.

optimize()

Return type

None

postprocess_opt()

Return type

None

print_opt_progress(*args, **kwargs)

Pick the correct method to report opt_progress.

When the model optimizer decides convergence we also report optimization progress using its data and not the data from LayerOpt, where the total ONIOM gradient is stored.

class pysisyphus.optimizers.LayerOpt.**Layers**(*geometry, opt_thresh, layers=None*)

Bases: object

classmethod **from_oniom_calculator**(*geometry, oniom_calc=None, layers=None, **kwargs*)

pysisyphus.optimizers.LayerOpt.**get_geom_kwargs**(*layer_ind, layer_mask*)

pysisyphus.optimizers.LayerOpt.**get_opt_kwargs**(*opt_key, layer_ind, thresh*)

pysisyphus.optimizers.MicroOptimizer module

class pysisyphus.optimizers.MicroOptimizer.**MicroOptimizer**(*geom, step='lbfgs', line_search=True, max_cycles=100000000, max_step=0.2, keep_last=10, rms_force=None, double_damp=True, dump=False, **kwargs*)

Bases: object

cg_step(*forces*)

lbfgs_step(*forces*)

log(*msg*)

optimize()

run()

sd_step(*forces*)

take_step(*energy, forces, return_step=False*)

pysisyphus.optimizers.NCOptimizer module

```
class pysisyphus.optimizers.NCOptimizer.NCOptimizer(geometry, *args, freeze_modes=None,
                                                    **kwargs)
```

Bases: [HessianOptimizer](#)

```
optimize()
```

pysisyphus.optimizers.Optimizer module

```
class pysisyphus.optimizers.Optimizer.ConvInfo(cur_cycle, energy_converged, max_force_converged,
                                                rms_force_converged, max_step_converged,
                                                rms_step_converged, desired_eigval_structure)
```

Bases: object

```
cur_cycle: int
```

```
desired_eigval_structure: bool
```

```
energy_converged: bool
```

```
get_convergence()
```

```
is_converged()
```

```
max_force_converged: bool
```

```
max_step_converged: bool
```

```
rms_force_converged: bool
```

```
rms_step_converged: bool
```

```
class pysisyphus.optimizers.Optimizer.Optimizer(geometry, thresh='gau_loose', max_step=0.04,
                                                max_cycles=150, min_step_norm=1e-08,
                                                assert_min_step=True, rms_force=None,
                                                rms_force_only=False, max_force_only=False,
                                                force_only=False,
                                                converge_to_geom_rms_thresh=0.05, align=False,
                                                align_factor=1.0, dump=False, dump_restart=False,
                                                print_every=1, prefix="", reparam_thresh=0.001,
                                                reparam_check_rms=True, reparam_when='after',
                                                overachieve_factor=0.0,
                                                check_eigval_structure=False, restart_info=None,
                                                check_coord_diffs=True, coord_diff_thresh=0.01,
                                                fragments=None, monitor_frag_dists=0, out_dir='.',
                                                h5_fn='optimization.h5', h5_group_name='opt')
```

Bases: object

```
__init__(geometry, thresh='gau_loose', max_step=0.04, max_cycles=150, min_step_norm=1e-08,
        assert_min_step=True, rms_force=None, rms_force_only=False, max_force_only=False,
        force_only=False, converge_to_geom_rms_thresh=0.05, align=False, align_factor=1.0,
        dump=False, dump_restart=False, print_every=1, prefix="", reparam_thresh=0.001,
        reparam_check_rms=True, reparam_when='after', overachieve_factor=0.0,
        check_eigval_structure=False, restart_info=None, check_coord_diffs=True,
        coord_diff_thresh=0.01, fragments=None, monitor_frag_dists=0, out_dir='.',
        h5_fn='optimization.h5', h5_group_name='opt')
```

Optimizer baseclass. Meant to be subclassed.

Parameters

- **geometry** (*Geometry*) -- Geometry to be optimized.
- **thresh** (Literal['gau_loose', 'gau', 'gau_tight', 'gau_vtight', 'baker', 'never'], default: 'gau_loose') -- Convergence threshold.
- **max_step** (float, default: 0.04) -- Maximum absolute component of the allowed step vector. Utilized in optimizers that don't support a trust region or line search.
- **max_cycles** (int, default: 150) -- Maximum number of allowed optimization cycles.
- **min_step_norm** (float, default: 1e-08) -- Minimum norm of an allowed step. If the step norm drops below this value a ZeroStepLength-exception is raised. The unit depends on the coordinate system of the supplied geometry.
- **assert_min_step** (bool, default: True) -- Flag that controls whether the norm of the proposed step is check for being too small.
- **rms_force** (Optional[float], default: None) -- Root-mean-square of the force from which user-defined thresholds are derived. When 'rms_force' is given 'thresh' is ignored.
- **rms_force_only** (bool, default: False) -- When set, convergence is signalled only based on rms(forces).
- **max_force_only** (bool, default: False) -- When set, convergence is signalled only based on max(|forces|).
- **force_only** (bool, default: False) -- When set, convergence is signalled only based on max(|forces|) and rms(forces).
- **converge_to_geom_rms_thresh** (float, default: 0.05) -- Threshold for the RMSD with another geometry. When the RMSD drops below this threshold convergence is signalled. Only used with Growing Newton trajectories.
- **align** (bool, default: False) -- Flag that controls whether the geometry is aligned in every step onto the coordinates of the previous step. Must not be used with internal coordinates.
- **align_factor** (float, default: 1.0) -- Factor that controls the strength of the alignment. 1.0 means full alignment, 0.0 means no alignment. The factor mixes the rotation matrix of the alignment with the identity matrix.
- **dump** (bool, default: False) -- Flag to control dumping/writing of optimization progress to the filesystem
- **dump_restart** (bool, default: False) -- Flag to control whether restart information is dumped to the filesystem.
- **print_every** (int, default: 1) -- Report optimization progress every nth cycle.
- **prefix** (str, default: '') -- Short string that is prepended to several files created by the optimizer. Allows distinguishing several optimizations carried out in the same directory.
- **reparam_thresh** (float, default: 0.001) -- Controls the minimal allowed similarity between coordinates after two successive reparametrizations. Convergence is signalled if the coordinates did not change significantly.
- **reparam_check_rms** (bool, default: True) -- Whether to check for (too) similar coordinates after reparametrization.

- **reparam_when** (Optional[Literal['before', 'after']], default: 'after') -- Reparametrize before or after calculating the step. Can also be turned off by setting it to None.
- **overachieve_factor** (float, default: 0.0) -- Signal convergence when max(forces) and rms(forces) fall below the chosen threshold, divided by this factor. Convergence of max(step) and rms(step) is ignored.
- **check_eigval_structure** (bool, default: False) -- Check the eigenvalues of the modes we maximize along. Convergence requires them to be negative. Useful if TS searches are started from geometries close to a minimum.
- **restart_info** (default: None) -- Restart information. Undocumented.
- **check_coord_diffs** (bool, default: True) -- Whether coordinates of chain-of-sates images are checked for being too similar.
- **coord_diff_thresh** (float, default: 0.01) -- Unitless threshold for similiary checking of COS image coordinates. The first image is assigned 0, the last image is assigned to 1.
- **fragments** (Optional[Tuple], default: None) -- Tuple of lists containing atom indices, defining two fragments.
- **monitor_frag_dists** (int, default: 0) -- Monitor fragment distances for N cycles. The optimization is terminated when the interfragment distances falls below the initial value after N cycles.
- **out_dir** (str, default: '. ') -- String poiting to a directory where optimization progress is dumped.
- **h5_fn** (str, default: 'optimization.h5') -- Basename of the HDF5 file used for dumping.
- **h5_group_name** (str, default: 'opt') -- Groupname used for dumping of this optimization.

check_convergence(step=None, multiple=1.0, overachieve_factor=None)

Check if the current convergence of the optimization is equal to or below the required thresholds, or a multiple thereof. The latter may be used in initiating the climbing image.

dump_restart_info()

final_summary()

fit_rigid(*, vectors=None, vector_lists=None, hessian=None)

get_path_for_fn(fn, with_prefix=True)

get_restart_info()

log(message, level=50)

make_conv_dict(key, rms_force=None, rms_force_only=False, max_force_only=False, force_only=False)

abstract optimize()

postprocess_opt()

prepare_opt()

print_opt_progress(conv_info)

procrustes()

Wrapper for procrustes that passes additional arguments along.

report_conv_thresholds()

run()

scale_by_max_step(*steps*)

set_restart_info(*restart_info*)

write_cycle_to_file()

write_image_trjs()

write_results()

write_to_out_dir(*out_fn, content, mode='w'*)

`pysisyphus.optimizers.Optimizer.get_data_model(geometry, is_cos, max_cycles)`

pysisyphus.optimizers.PreconLBFGS module

```
class pysisyphus.optimizers.PreconLBFGS(geometry, alpha_init=1.0, history=7,
                                         precon=True, precon_update=1,
                                         precon_getter_update=None, precon_kind='full',
                                         max_step_element=None, line_search='armijo',
                                         c_stab=None, **kwargs)
```

Bases: [Optimizer](#)

```
__init__(geometry, alpha_init=1.0, history=7, precon=True, precon_update=1,
          precon_getter_update=None, precon_kind='full', max_step_element=None, line_search='armijo',
          c_stab=None, **kwargs)
```

Preconditioned limited-memory BFGS optimizer.

See `pysisyphus.optimizers.precon` for related references.

Parameters

- **geometry** ([Geometry](#)) -- Geometry to be optimized.
- **alpha_init** (float, default: 1.0) -- Initial scaling factor for the first trial step in the explicit line search.
- **history** (int, default: 7) -- History size. Keep last 'history' steps and gradient differences.
- **precon** (bool, default: True) -- Whether to use preconditioning or not.
- **precon_update** (int, default: 1) -- Recalculate preconditioner P in every n-th cycle with the same topology.
- **precon_getter_update** (Optional[int], default: None) -- Recalculate topology for preconditioner P in every n-th cycle. It is usually sufficient to only determine the topology once at the beginning.
- **precon_kind** (Literal['full', 'full_fast', 'bonds', 'bonds_bends'], default: 'full') -- What types of primitive internal coordinates to consider in the preconditioner.
- **max_step_element** (Optional[float], default: None) -- Maximum component of the absolute step vector when no line search is carried out.

- **line_search** (Literal['armijo', 'armijo_fg', 'strong_wolfe', 'hz', None, False], default: 'armijo') -- Whether to use explicit line searches and if so, which kind of line search.
- **c_stab** (Optional[float], default: None) -- Regularization constant c in $(H + cI)^{\dagger}$ in atomic units.
- ****kwargs** -- Keyword arguments passed to the Optimizer baseclass.

get_precon_getter()

optimize()

prepare_opt()

scale_max_element(*step*, *max_step_element*)

pysisyphus.optimizers.PreconSteepestDescent module

```
class pysisyphus.optimizers.PreconSteepestDescent.PreconSteepestDescent(geometry,  
                                                                           alpha_init=0.5,  
                                                                           **kwargs)
```

Bases: [PreconLBFGS](#)

pysisyphus.optimizers.QuickMin module

```
class pysisyphus.optimizers.QuickMin.QuickMin(geometry, dt=0.35, **kwargs)
```

Bases: [Optimizer](#)

optimize()

prepare_opt()

reset()

pysisyphus.optimizers.RFOptimizer module

```
class pysisyphus.optimizers.RFOptimizer.RFOptimizer(geometry, line_search=True, gediis=False,  
                                                      gdiis=True, gdiis_thresh=0.0025,  
                                                      gediis_thresh=0.01, gdiis_test_direction=True,  
                                                      max_micro_cycles=25, adapt_step_func=False,  
                                                      **kwargs)
```

Bases: [HessianOptimizer](#)

```
__init__(geometry, line_search=True, gediis=False, gdiis=True, gdiis_thresh=0.0025, gediis_thresh=0.01,  
         gdiis_test_direction=True, max_micro_cycles=25, adapt_step_func=False, **kwargs)
```

Rational function Optimizer.

Parameters

- **geometry** ([Geometry](#)) -- Geometry to be optimized.
- **line_search** (bool, default: True) -- Whether to carry out implicit line searches.
- **gediis** (bool, default: False) -- Whether to enable GEDIIS.

- **gdiis** (bool, default: True) -- Whether to enable GDIIS.
- **gdiis_thresh** (float, default: 0.0025) -- Threshold for rms(forces) to enable GDIIS.
- **gediis_thresh** (float, default: 0.01) -- Threshold for rms(step) to enable GEDIIS.
- **gdiis_test_direction** (bool, default: True) -- Whether to the overlap of the RFO step and the GDIIS step.
- **max_micro_cycles** (int, default: 25) -- Number of restricted-step microcycles. Disabled by default.
- **adapt_step_func** (bool, default: False) -- Whether to switch between shifted Newton and RFO-steps.
- ****kwargs** -- Keyword arguments passed to the Optimizer/HessianOptimizer baseclass.

optimize()

postprocess_opt()

pysisyphus.optimizers.RSA module

```
class pysisyphus.optimizers.RSA.RSA(geometry, trust_radius=0.5, trust_update=True, trust_min=0.1,
                                     trust_max=1, max_energy_incr=None, hessian_update='bfgs',
                                     hessian_init='fischer', hessian_recalc=None,
                                     hessian_recalc_adapt=None, hessian_xtb=False,
                                     hessian_recalc_reset=False, small_eigval_thresh=1e-08,
                                     line_search=False, alpha0=1.0, max_micro_cycles=25,
                                     rfo_overlaps=False, **kwargs)
```

Bases: [HessianOptimizer](#)

The Importance of Step Control in Optimization Methods, del Campo, 2009.

optimize()

pysisyphus.optimizers.StabilizedQNMethod module

```
class pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod(geometry, alpha=0.5,
                                                                      alpha_max=1,
                                                                      alpha_stretch=0.5,
                                                                      alpha_stretch_max=1,
                                                                      eps=0.0001, hist_max=10,
                                                                      E_thresh=1e-06, bio=True,
                                                                      trust_radius=0.1,
                                                                      linesearch=True, **kwargs)
```

Bases: [Optimizer](#)

adjust_alpha(gradient, precon_gradient)

adjust_alpha_stretch()

bio_mode(gradient)

property n_hist

```
optimize()  
precondition_gradient(gradient, steps, grad_diffs, eps)  
prepare_opt()
```

pysisyphus.optimizers.SteepestDescent module

```
class pysisyphus.optimizers.SteepestDescent.SteepestDescent(geometry, alpha=0.1, **kwargs)  
    Bases: BacktrackingOptimizer  
    optimize()  
    prepare_opt()
```

pysisyphus.optimizers.StringOptimizer module

```
class pysisyphus.optimizers.StringOptimizer.StringOptimizer(geometry, max_step=0.1,  
                                                            stop_in_when_full=-1, keep_last=10,  
                                                            lbfgs_when_full=True,  
                                                            gamma_mult=False,  
                                                            double_damp=True,  
                                                            scale_step='global', **kwargs)  
    Bases: Optimizer  
    check_convergence(*args, **kwargs)  
        Check if the current convergence of the optimization is equal to or below the required thresholds, or a  
        multiple thereof. The latter may be used in initiating the climbing image.  
    optimize()  
    prepare_opt()  
    reset()  
    restrict_step_components(steps)
```

pysisyphus.optimizers.closures module

```
pysisyphus.optimizers.closures.bfgs_multiply(s_list, y_list, vector, beta=1, P=None, logger=None,  
                                              gamma_mult=True, mu_reg=None, inds=None,  
                                              cur_size=None)
```

Matrix-vector product $H \cdot v$.

Multiplies given vector with inverse Hessian, obtained from repeated BFGS updates calculated from steps in 's_list' and gradient differences in 'y_list'.

Based on algorithm 7.4 Nocedal, Num. Opt., p. 178.

```
pysisyphus.optimizers.closures.get_update_mu_reg(mu_min=0.001, gamma_1=0.1, gamma_2=5.0,  
                                                eta_1=0.01, eta_2=0.9, logger=None)
```

See 5.1 in [1]


```
pysisyphus.optimizers.closures.lbfgs_closure(force_getter, M=10, beta=1, restrict_step=None)
```

```
pysisyphus.optimizers.closures.modified_broyden_closure(force_getter, M=5, beta=1,  
                                                         restrict_step=None)
```

<https://doi.org/10.1006/jcph.1996.0059> F corresponds to the residual gradient, so we after calling `force_getter` we multiply the force by -1 to get the gradient.

```
pysisyphus.optimizers.closures.small_lbfgs_closure(history=5, gamma_mult=True)
```

Compact LBFGS closure.

The returned function takes two arguments: `forces` and `prev_step`. `forces` are the forces at the current iterate and `prev_step` is the previous step that lead us to the current iterate. In this way step restriction/line search can be done outside of the `lbfgs` function.

pysisyphus.optimizers.cls_map module

```
pysisyphus.optimizers.cls_map.get_opt_cls(opt_key)
```

```
pysisyphus.optimizers.cls_map.key_is_tsopt(opt_key)
```

pysisyphus.optimizers.exceptions module

```
exception pysisyphus.optimizers.exceptions.OptimizationError
```

Bases: `Exception`

```
exception pysisyphus.optimizers.exceptions.ZeroStepLength
```

Bases: `Exception`

pysisyphus.optimizers.gdiis module

```
class pysisyphus.optimizers.gdiis.DIISResult(coeffs, coords, forces, energy, N, type)
```

Bases: `tuple`

N

Alias for field number 4

coeffs

Alias for field number 0

coords

Alias for field number 1

energy

Alias for field number 3

forces

Alias for field number 2

type

Alias for field number 5

```
pysisyphus.optimizers.gdiis.diis_result(coeffs, coords, forces, energy=None, prefix="")
```

```
pysisyphus.optimizers.gdiis.from_coeffs(vec, coeffs)
pysisyphus.optimizers.gdiis.gdiis(err_vecs, coords, forces, ref_step, max_vecs=5, test_direction=True)
pysisyphus.optimizers.gdiis.gediis(coords, energies, forces, hessian=None, max_vecs=3)
pysisyphus.optimizers.gdiis.log(msg)
pysisyphus.optimizers.gdiis.valid_diis_direction(diis_step, ref_step, use)
```

pysisyphus.optimizers.guess_hessians module

```
pysisyphus.optimizers.guess_hessians.fischer_guess(geom)
pysisyphus.optimizers.guess_hessians.get_guess_hessian(geometry, hessian_init, int_gradient=None,
                                                         cart_gradient=None, h5_fn=None)

    Obtain/calculate (model) Hessian.

    For hessian_init="calc" the Hessian will be in the coord_type of the geometry, otherwise a Hessian in primitive
    internals will be returned.

pysisyphus.optimizers.guess_hessians.get_lindh_alpha(atom1, atom2)
pysisyphus.optimizers.guess_hessians.improved_guess(geom, bond_func, bend_func, dihedral_func)
pysisyphus.optimizers.guess_hessians.lindh_guess(geom)
    Slightly modified Lindh model hessian as described in [1].

    Instead of using the tabulated r_ref,ij values from [1] we will use the 'true' covalent radii as pyberny. The tabulated
    r_ref,ij value for two carbons (2nd period) is 2.87 Bohr. Carbons covalent radius is ~ 1.44 Bohr, so 2*1.44 Bohr
    = 2.88 Bohr which fits nicely with the tabulate value. If values for elements > 3rd are requested the alpha values
    for the 3rd period will be (re)used.

pysisyphus.optimizers.guess_hessians.lindh_style_guess(geom, ks, rhos)
    Approximate force constants according to Lindh.[1]

    Bonds:  $k_{ij} = k_r * \rho_{ij}$  Bends:  $k_{ijk} = k_b * \rho_{ij} * \rho_{jk}$  Dihedrals:  $k_{ijkl} = k_d * \rho_{ij} * \rho_{jk} * \rho_{kl}$ 

pysisyphus.optimizers.guess_hessians.simple_guess(geom)
    Default force constants.

pysisyphus.optimizers.guess_hessians.swart_guess(geom)
pysisyphus.optimizers.guess_hessians.ts_hessian(hessian, coord_inds, damp=0.25)
    According to [3]
pysisyphus.optimizers.guess_hessians.xtb_hessian(geom, gfn=None)
```

pysisyphus.optimizers.hessian_updates module

`pysisyphus.optimizers.hessian_updates.bfgs_update(H, dx, dg)`

`pysisyphus.optimizers.hessian_updates.bofill_update(H, dx, dg)`

`pysisyphus.optimizers.hessian_updates.damped_bfgs_update(H, dx, dg)`

See [5]

`pysisyphus.optimizers.hessian_updates.double_damp(s, y, H=None, s_list=None, y_list=None, mu_1=0.2, mu_2=0.2, logger=None)`

Double damped step 's' and gradient differences 'y'.

H is the inverse Hessian! See [6]. Potentially updates s and y. y is only updated if mu_2 is not None.

Parameters

- **s** (*np.array, shape (N,), floats*) -- Coordiante differences/step.
- **y** (*np.array, shape (N,), floats*) -- Gradient differences
- **H** (*np.array, shape (N, N), floats, optional*) -- Inverse Hessian.
- **s_list** (*list of nd.array, shape (K, N), optional*) -- List of K previous steps. If no H is supplied and prev_ys is given the matrix-vector product Hy will be calculated through the two-loop LBFGS-recursion.
- **y_list** (*list of nd.array, shape (K, N), optional*) -- List of K previous gradient differences. See s_list.
- **mu_1** (*float, optional*) -- Parameter for 's' damping.
- **mu_2** (*float, optional*) -- Parameter for 'y' damping.
- **logger** (*logging.Logger, optional*) -- Logger to be used.

Returns

- **s** (*np.array, shape (N,), floats*) -- Damped coordiante differences/step.
- **y** (*np.array, shape (N,), floats*) -- Damped gradient differences

`pysisyphus.optimizers.hessian_updates.flowchart_update(H, dx, dg)`

`pysisyphus.optimizers.hessian_updates.mod_flowchart_update(H, dx, dg)`

`pysisyphus.optimizers.hessian_updates.psb_update(z, dx)`

`pysisyphus.optimizers.hessian_updates.sr1_update(z, dx)`

`pysisyphus.optimizers.hessian_updates.ts_bfgs_update(H, dx, dg)`

As described in [7]

`pysisyphus.optimizers.hessian_updates.ts_bfgs_update_org(H, dx, dg)`

Do not use! Implemented as described in the 1998 bofill paper [8].

This does not seem to work too well.

`pysisyphus.optimizers.hessian_updates.ts_bfgs_update_revised(H, dx, dg)`

TS-BFGS update as described in [9].

Better than the original formula of Bofill, worse than the implementation in [7]. a is caluclated as described in the footnote 1 on page 38. Eq. (8) looks suspicious as it contains the inverse of a vector?! As also outlined in the paper abs(a) is used (**|a|** in the paper).

pysisyphus.optimizers.poly_fit module**class** pysisyphus.optimizers.poly_fit.**FitResult**(*x, y, polys*)

Bases: tuple

polys

Alias for field number 2

x

Alias for field number 0

y

Alias for field number 1

pysisyphus.optimizers.poly_fit.**cubic_fit**(*e0, e1, g0, g1*)pysisyphus.optimizers.poly_fit.**gen_solutions**()Given two energies (*e0, e1*) and corresponding gradients (*g0, g1*) we can (try to) fit a quartic polynomial

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$$

s.t. the constraint $f'(x) \geq 0$, with the equality being fulfilled at only one point. There are five unknowns ($a_0 - a_4$) to be determined. Four equations can be derived from $f(x)$ and its first derivative

$$f'(x) = a_1 + 2a_2x + 3a_3x^2 + 4a_4x^3.$$

With (*e0, g0*) being given at $x=0$ and (*e1, g1*) being given at $x=1$ we can setup the following equations:

$$f(0) = a_0 \quad (1) \quad f'(0) = a_1 \quad (2)$$

using e_0 and g_0 at $x=0$, and

$$f(1) = a_0 + a_1 + a_2 + a_3 + a_4 \quad (3) \quad f'(1) = a_1 + 2a_2 + 3a_3 + 4a_4. \quad (4)$$

The missing last equation can be derived from the constraint. The second derivative of $f(x)$ is

$$f''(x) = 2a_2 + 6a_3x + 12a_4x^2$$

and shall be positive except at one point where it is allowed to be 0, that its two roots ($f''(x) = 0$) must be degenerate. This is fulfilled when the discriminant D of the quadratic polynomial $ax^2 + bx + c$ is zero.

$$D = b^2 - 4ac = 0$$

With

$$a = 12a_4 \quad b = 6a_3 \quad c = 2a_2$$

we get

$$0 = (6a_3)^2 - 4 \cdot 12a_4 \cdot 2a_2 = 36a_3^2 - 96a_4a_2 = 3a_3^2 - 8a_4a_2 \quad (5) \quad \text{or} \quad a_4 = \frac{3}{8} \cdot \frac{a_3^2}{a_2}$$

Using (1) - (5) we can solve the set of equations for $a_0 - a_4$.

pysisyphus.optimizers.poly_fit.**get_maximum**(*poly*)pysisyphus.optimizers.poly_fit.**get_minimum**(*poly*)pysisyphus.optimizers.poly_fit.**poly_line_search**(*cur_energy, prev_energy, cur_grad, prev_grad, prev_step, cubic_max_x=2.0, quartic_max_x=4.0, logger=None*)

Generate directional gradients by projecting them on the previous step.

```
pysisyphus.optimizers.poly_fit.quartic_fit(e0, e1, g0, g1, maximize=False)
```

See `gen_solutions()` for derivation.

```
pysisyphus.optimizers.poly_fit.quintic_fit(e0, e1, g0, g1, H0, H1)
```

pysisyphus.optimizers.precon module

```
pysisyphus.optimizers.precon.get_lindh_k(atoms, coords3d, bonds=None, angles=None, torsions=None)
```

```
pysisyphus.optimizers.precon.get_lindh_precon(atoms, coords, bonds=None, bends=None,  
                                              dihedrals=None, c_stab=0.0103, logger=None)
```

`c_stab = 0.00103 hartree/bohr2` corresponds to 0.1 eV/Å² as given in the paper.

```
pysisyphus.optimizers.precon.precon_getter(geom, c_stab=0.0103, kind='full', logger=None)
```

pysisyphus.optimizers.restrict_step module

```
pysisyphus.optimizers.restrict_step.get_scale_max(max_element)
```

```
pysisyphus.optimizers.restrict_step.restrict_step(steps, max_step)
```

```
pysisyphus.optimizers.restrict_step.scale_by_max_step(steps, max_step)
```

Module contents

```
class pysisyphus.optimizers.CubicNewton(geometry, **kwargs)
```

Bases: [*HessianOptimizer*](#)

```
optimize()
```

```
postprocess_opt()
```

```
class pysisyphus.optimizers.MicroOptimizer(geom, step='lbfgs', line_search=True,  
                                           max_cycles=100000000, max_step=0.2, keep_last=10,  
                                           rms_force=None, double_damp=True, dump=False,  
                                           **kwargs)
```

Bases: `object`

```
cg_step(forces)
```

```
lbfgs_step(forces)
```

```
log(msg)
```

```
optimize()
```

```
run()
```

```
sd_step(forces)
```

```
take_step(energy, forces, return_step=False)
```

17.1.1.14 pysisyphus.plotters package

Submodules

pysisyphus.plotters.AnimPlot module

```
class pysisyphus.plotters.AnimPlot.AnimPlot(calculator, optimizer, xlim=None, ylim=None,  
                                              levels=None, num=100, figsize=(8, 8), interval=250,  
                                              energy_profile=True, colorbar=True, save=None,  
                                              title=True, tight_layout=False)
```

Bases: object

animate()

as_html5(*out_fn*)

func(*frame*)

on_keypress(*event*)

Pause on SPACE press.

Module contents

17.1.1.15 pysisyphus.stochastic package

Submodules

pysisyphus.stochastic.FragmentKick module

```
class pysisyphus.stochastic.FragmentKick.FragmentKick(geom, fragments, fix_fragments=None,  
                                                         displace_from=None,  
                                                         random_displacement=False, **kwargs)
```

Bases: *Kick*

get_fragments(*fragments*)

get_input_geom(*geom*)

get_origin()

kick_fragment(*frag_coords*)

print_fragments()

pysisyphus.stochastic.Kick module

```
class pysisyphus.stochastic.Kick.Kick(geom, radius=0.5, **kwargs)
```

Bases: [Pipeline](#)

```
get_input_geom(geom)
```

```
get_kick()
```

```
run_cycle(geom)
```

pysisyphus.stochastic.Pipeline module

```
class pysisyphus.stochastic.Pipeline.Pipeline(geom, calc_getter=None, seed=None, max_cycles=5,
                                              cycle_size=15, rmsd_thresh=0.1, energy_thresh=0.001,
                                              energy_range=0.125, compare_num=25, break_after=2,
                                              calc_kwargs=None)
```

Bases: object

```
atoms_are_too_close(geom, factor=0.7)
```

Determine if atoms are too close.

```
geom_is_close_in_energy(geom)
```

```
geom_is_new(geom)
```

Determine if geometry is not already known.

```
geom_is_valid(geom)
```

Filter out geometries that are None, or were the atoms are too close or when they are already known.

```
get_input_geom(geom)
```

```
get_unique_geometries(geoms)
```

```
get_valid_index_set(to_intersect)
```

```
log(message)
```

Write a log message.

Wraps the logger variable.

Parameters

message (*str*) -- Message to be logged.

```
run()
```

```
run_geom_opt(geom)
```

```
write_geoms_to_trj(geoms, fn, comments=None)
```

pysisyphus.stochastic.align module

`pysisyphus.stochastic.align.apply_transform(coords3d, swap, reflection)`

Apply axis swaps and reflections to a coordinate array.

`pysisyphus.stochastic.align.match_geom_atoms(ref_geom, geom_to_match, hydrogen=True)`

Apply the hungarian method to geom_to_match.

Uses the scipy implementation of the Hungarian method/ Kuhn-Munkres algorithm in `scipy.optimize.linear_sum_assignment`.

See

[1] 10.1021/ci400534h [2] 10.1021/acs.jcim.6b00516

`pysisyphus.stochastic.align.matched_rmsd(geom1, geom2, thresh=0.05)`

RMSD for optimally aligned and matched geometries.

Returns

- **matched_rmsd** (*float*) -- RMSD of optimally aligned and matched geometries.
- **matched_geoms** (*tuple(Geometry, Geometry)*) -- Tuple of the optimally aligned and matched geometries.

Module contents

`class pysisyphus.stochastic.FragmentKick(geom, fragments, fix_fragments=None, displace_from=None, random_displacement=False, **kwargs)`

Bases: [*Kick*](#)

`get_fragments(fragments)`

`get_input_geom(geom)`

`get_origin()`

`kick_fragment(frag_coords)`

`print_fragments()`

`class pysisyphus.stochastic.Kick(geom, radius=0.5, **kwargs)`

Bases: [*Pipeline*](#)

`get_input_geom(geom)`

`get_kick()`

`run_cycle(geom)`

17.1.1.16 pysisyphus.tests package

Submodules

pysisyphus.tests.test_calculators module

Module contents

17.1.1.17 pysisyphus.tsoptimizers package

Submodules

pysisyphus.tsoptimizers.RSIRFOptimizer module

```
class pysisyphus.tsoptimizers.RSIRFOptimizer.RSIRFOptimizer(geometry, roots=None, root=0,
                                                             hessian_ref=None, rx_modes=None,
                                                             prim_coord=None, rx_coords=None,
                                                             hessian_init='calc',
                                                             hessian_update='bofill',
                                                             hessian_recalc_reset=True,
                                                             max_micro_cycles=50,
                                                             trust_radius=0.3, trust_max=0.5,
                                                             augment_bonds=False,
                                                             min_line_search=False,
                                                             max_line_search=False,
                                                             assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

pysisyphus.tsoptimizers.RSPRFOptimizer module

```
class pysisyphus.tsoptimizers.RSPRFOptimizer.RSPRFOptimizer(geometry, roots=None, root=0,
                                                             hessian_ref=None, rx_modes=None,
                                                             prim_coord=None, rx_coords=None,
                                                             hessian_init='calc',
                                                             hessian_update='bofill',
                                                             hessian_recalc_reset=True,
                                                             max_micro_cycles=50,
                                                             trust_radius=0.3, trust_max=0.5,
                                                             augment_bonds=False,
                                                             min_line_search=False,
                                                             max_line_search=False,
                                                             assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

pysisyphus.tsoptimizers.TRIM module

```
class pysisyphus.tsoptimizers.TRIM.TRIM(geometry, roots=None, root=0, hessian_ref=None,
                                         rx_modes=None, prim_coord=None, rx_coords=None,
                                         hessian_init='calc', hessian_update='bofill',
                                         hessian_recalc_reset=True, max_micro_cycles=50,
                                         trust_radius=0.3, trust_max=0.5, augment_bonds=False,
                                         min_line_search=False, max_line_search=False,
                                         assert_neg_eigval=False, **kwargs)
```

Bases: [TSHessianOptimizer](#)

optimize()

pysisyphus.tsoptimizers.TSHessianOptimizer module

```
class pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer(geometry, roots=None,
                                                                       root=0, hessian_ref=None,
                                                                       rx_modes=None,
                                                                       prim_coord=None,
                                                                       rx_coords=None,
                                                                       hessian_init='calc',
                                                                       hessian_update='bofill',
                                                                       hessian_recalc_reset=True,
                                                                       max_micro_cycles=50,
                                                                       trust_radius=0.3,
                                                                       trust_max=0.5,
                                                                       augment_bonds=False,
                                                                       min_line_search=False,
                                                                       max_line_search=False,
                                                                       assert_neg_eigval=False,
                                                                       **kwargs)
```

Bases: [HessianOptimizer](#)

Optimizer to find first-order saddle points.

```
__init__(geometry, roots=None, root=0, hessian_ref=None, rx_modes=None, prim_coord=None,
          rx_coords=None, hessian_init='calc', hessian_update='bofill', hessian_recalc_reset=True,
          max_micro_cycles=50, trust_radius=0.3, trust_max=0.5, augment_bonds=False,
          min_line_search=False, max_line_search=False, assert_neg_eigval=False, **kwargs)
```

Baseclass for transition state optimizers utilizing Hessian information.

Several arguments expect a typed primitive or an iterable of typed primitives. A typed primitive is specified as (PrimType, int, int, ...), e.g., for a bond between atoms 0 and 1: (BOND, 0, 1) or for a bend between the atom triple 0, 1, 2 as (BEND, 0, 1, 2).

Parameters

- **geometry** ([Geometry](#)) -- Geometry to be optimized.
- **roots** (Optional[List[int]], default: None) -- Indices of modes to maximize along, e.g., to optimize saddle points of 2nd order. Overrides 'root'.
- **root** (int, default: 0) -- Index of imaginary mode to maximize along. Shortcut for 'roots' with only one root.

- **hessian_ref** (Optional[str], default: None) -- Filename pointing to a pysisyphus HDF5 Hessian.
- **rx_modes** (*iterable of (iterable of (typed_prim, phase_factor))*) -- Select initial root(s) by overlap with a modes constructed from the given typed primitives with respective phase factors.
- **prim_coord** (*typed_prim*) -- Select initial root/imaginary mode by overlap with this internal coordinate. Shortcut for 'rx_modes' with only one internal coordinate.
- **rx_coords** (*iterable of (typed_prim)*) -- Construct imaginary mode comprising the given typed prims by modifying a model Hessian.
- **hessian_init** (Literal['calc', 'unit', 'fischer', 'lindh', 'simple', 'swart', 'xtb', 'xtb1', 'xtbff'], default: 'calc') -- Type of initial model Hessian.
- **hessian_update** (Literal['none', None, False, 'bfgs', 'damped_bfgs', 'flowchart', 'bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev'], default: 'bofill') -- Type of Hessian update. Defaults to BFGS for minimizations and Bofill for saddle point searches.
- **hessian_recalc_reset** (bool, default: True) -- Whether to skip Hessian recalculation after reset. Undocumented.
- **max_micro_cycles** (int, default: 50) -- Maximum number of RS iterations.
- **trust_radius** (float, default: 0.3) -- Initial trust radius in whatever unit the optimization is carried out.
- **trust_max** (float, default: 0.5) -- Maximum trust radius.
- **augment_bonds** (bool, default: False) -- Try to derive additional stretching coordinates from the imaginary mode.
- **min_line_search** (bool, default: False) -- Carry out line search along the imaginary mode.
- **max_line_search** (bool, default: False) -- Carry out line search in the subspace that is minimized.
- **assert_neg_eigval** (bool, default: False) -- Check for the existences for at least one significant negative eigenvalue. If enabled and no negative eigenvalue is present the optimization will be aborted.
- ****kwargs** -- Keyword arguments passed to the HessianOptimizer/Optimizer baseclass.

```
static do_line_search(e0, e1, g0, g1, prev_step, maximize, logger=None)
```

```
prepare_opt(*args, **kwargs)
```

```
property root
```

```
property roots
```

```
step_and_grad_from_line_search(energy, gradient_trans, eigvecs, min_indices, max_indices)
```

```
update_ts_mode(eigvals, eigvecs)
```

```
valid_updates = ('bofill', 'ts_bfgs', 'ts_bfgs_org', 'ts_bfgs_rev')
```

Module contents

```
class pysisyphus.tsoptimizers.RSIRFOptimizer(geometry, roots=None, root=0, hessian_ref=None,
                                             rx_modes=None, prim_coord=None, rx_coords=None,
                                             hessian_init='calc', hessian_update='bofill',
                                             hessian_recalc_reset=True, max_micro_cycles=50,
                                             trust_radius=0.3, trust_max=0.5, augment_bonds=False,
                                             min_line_search=False, max_line_search=False,
                                             assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

```
class pysisyphus.tsoptimizers.RSPRFOptimizer(geometry, roots=None, root=0, hessian_ref=None,
                                             rx_modes=None, prim_coord=None, rx_coords=None,
                                             hessian_init='calc', hessian_update='bofill',
                                             hessian_recalc_reset=True, max_micro_cycles=50,
                                             trust_radius=0.3, trust_max=0.5, augment_bonds=False,
                                             min_line_search=False, max_line_search=False,
                                             assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

```
class pysisyphus.tsoptimizers.TRIM(geometry, roots=None, root=0, hessian_ref=None, rx_modes=None,
                                     prim_coord=None, rx_coords=None, hessian_init='calc',
                                     hessian_update='bofill', hessian_recalc_reset=True,
                                     max_micro_cycles=50, trust_radius=0.3, trust_max=0.5,
                                     augment_bonds=False, min_line_search=False,
                                     max_line_search=False, assert_neg_eigval=False, **kwargs)
```

Bases: *TSHessianOptimizer*

optimize()

17.1.1.18 pysisyphus.wrapper package

Submodules

pysisyphus.wrapper.jmol module

pysisyphus.wrapper.jmol.**call_jmol**(spt_str, show=False)

pysisyphus.wrapper.jmol.**render_cdd_cube**(cdd_cube, isoval=0.001, orient="")

pysisyphus.wrapper.jmol.**render_geom_and_charges**(geom, point_charges)

pysisyphus.wrapper.jmol.**view_cdd_cube**(cdd_cube, isoval=0.001, orient="")

pysisyphus.wrapper.mwfn module

`pysisyphus.wrapper.mwfn.call_mwfn(inp_fn, stdin, cwd=None)`

`pysisyphus.wrapper.mwfn.get_mwfn_exc_str(energies, Xa, Ya=None, Xb=None, Yb=None, thresh=0.001)`

Write plain text input for MWFN according to 3.21.

As of version 3.8 MWFN does not seem to handle this file when spin labels are present, even though it is given as an example in the manual.

`pysisyphus.wrapper.mwfn.log(msg)`

`pysisyphus.wrapper.mwfn.make_cdd(inp_fn, state, log_fn, cwd=None, keep=False, quality=2, prefix='S')`

Create CDD cube in cwd.

Parameters

- **inp_fn** (*str*) -- Filename of a .molden/.fchk file.
- **state** (*int*) -- CDD cubes will be generated up to this state.
- **log_fn** (*str*) -- Filename of the .log file.
- **cwd** (*str or Path, optional*) -- If a different cwd should be used.
- **keep** (*bool*) -- Wether to keep electron.cub and hole.cub, default is False.
- **quality** (*int*) -- Quality of the cube. (1=low, 2=medium, 3=high).

`pysisyphus.wrapper.mwfn.wrap_stdin(stdin)`

pysisyphus.wrapper.packmol module

`pysisyphus.wrapper.packmol.call_packmol(inp)`

`pysisyphus.wrapper.packmol.make_input(output_fn, solvent_fn, solvent_num, sphere_radius, solute_fn=None, solute_num=None, tolerance=2.0)`

Module contents

17.1.2 Submodules

17.1.3 pysisyphus.Geometry module

`class pysisyphus.Geometry.Geometry(atoms, coords, fragments=None, coord_type='cart', coord_kwargs=None, isotopes=None, freeze_atoms=None, comment="", name=")`

Bases: object

`__init__(atoms, coords, fragments=None, coord_type='cart', coord_kwargs=None, isotopes=None, freeze_atoms=None, comment="", name=")`

Object representing atoms in a coordinate system.

The Geometry represents atoms and their positions in coordinate system. By default cartesian coordinates are used, but internal coordinates are also possible.

Parameters

- **atoms** (*iterable*) -- Iterable of length N, containing element symbols.
- **coords** (*1d iterable*) -- 1d iterable of length 3N, containing the cartesian coordinates of N atoms.
- **fragments** (*dict, optional*) -- Dict with different keys denoting different fragments. The values contain lists of atom indices.
- **coord_type** (*{"cart", "redund"}, optional*) -- Type of coordinate system to use. Right now cartesian (cart) and redundand (redund) are supported.
- **coord_kwargs** (*dict, optional*) -- Dictionary containing additional arguments that get passed to the constructor of the internal coordinate class.
- **isotopes** (*iterable of pairs, optional*) -- Iterable of pairs consisting of 0-based atom index and either an integer or a float. If an integer is given the closest isotope mass will be selected. Given a float, this float will be directly used as mass.
- **freeze_atoms** (*iterable of integers*) -- Specifies which atoms should remain fixed at their initial positions.
- **comment** (*str, optional*) -- Comment string.
- **name** (*str, optional*) -- Verbose name of the geometry, e.g. methanal or water. Used for printing

align_principal_axes()

Align the principal axes to the cartesian axes.

<https://math.stackexchange.com/questions/145023>

property all_energies

Return energies of all states that were calculated.

This will also set self.energy, which may NOT be the ground state, but the state correspondig to the 'root' attribute of the calculator.

approximate_radius()

Approximate molecule radius from the biggest atom distance along an axis.

as_ase_atoms()

as_g98_list()

Returns data for fake Gaussian98 standard orientation output.

Returns

g98_list -- List with one row per atom. Every row contains [center number, atomic number, atomic type (always 0 for now), X Y Z coordinates in Angstrom.

Return type

list

as_xyz(comment="", atoms=None, cart_coords=None)

Current geometry as a string in XYZ-format.

Parameters

- **comment** (*str, optional*) -- Will be written in the second line (comment line) of the XYZ-string.
- **cart_coords** (*np.array, 1d, shape (3 * atoms.size,)*) -- Cartesians for dumping instead of self._coords.

Returns

xyz_str -- Current geometry as string in XYZ-format.

Return type

str

assert_cart_coords(*coords*)

assert_compatibility(*other*)

Assert that two Geometries can be subtracted from each other.

Parameters

other (*Geometry*) -- Geometry for comparison.

atom_indices()

Dict with atom types as key and corresponding indices as values.

Returns

inds_dict -- Unique atom types as keys, corresponding indices as values.

Return type

dict

property atom_types

atom_xyz_iter()

property atomic_numbers

property bond_sets

calc_double_ao_overlap(*geom2*)

calc_energy_and_forces()

Force a calculation of the current energy and forces.

property cart_coords

property cart_forces

property cart_gradient

property cart_hessian

center()

property center_of_mass

Returns the center of mass.

Returns

R -- Center of mass.

Return type

np.array, shape(3,)

center_of_mass_at(*coords3d*)

Returns the center of mass at given coords3d.

Parameters

coords3d (*np.array, shape(N, 3)*) -- Cartesian coordinates.

Returns

R -- Center of mass.

Return type

np.array, shape(3,)

property centroid

Geometric center of the Geometry.

Returns

R -- Geometric center of the Geometry.

Return type

np.array, shape(3,)

clear()

Reset the object state.

property comment

```
coord_types = {'cart': None, 'cartesian': <class
'pysisyphus.intcoords.CartesianCoords.CartesianCoords'>, 'dlc': <class
'pysisyphus.intcoords.DLC.DLC'>, 'hdlc': <class 'pysisyphus.intcoords.DLC.HDLC'>,
'hredund': <class 'pysisyphus.intcoords.RedundantCoords.HybridRedundantCoords'>,
'mwcartesian': <class 'pysisyphus.intcoords.CartesianCoords.MWCartesianCoords'>,
'redund': <class 'pysisyphus.intcoords.RedundantCoords.RedundantCoords'>, 'tmtric':
<class 'pysisyphus.intcoords.RedundantCoords.TMTRIC'>, 'tric': <class
'pysisyphus.intcoords.RedundantCoords.TRIC'>}
```

property coords

1d vector of atomic coordinates.

Returns

coords -- 1d array holding the current coordinates.

Return type

np.array

property coords3d

Coordinates in 3d.

Returns

coords3d -- Coordinates of the Geometry as 2D array.

Return type

np.array

property coords_by_type

Coordinates in 3d by atom type and their corresponding indices.

Returns

- **cbt** (*dict*) -- Dictionary with the unique atom types of the Geometry as keys. It's values are the 3d coordinates of the corresponding atom type.
- **inds** (*dict*) -- Dictionary with the unique atom types of the Geometry as keys. It's values are the original indices of the 3d coordinates in the whole coords3d array.

copy(coord_type=None, coord_kwargs=None)

Returns a new Geometry object with same atoms and coordinates.

Parameters

- **coord_type** (*str*) -- Desired coord_type, defaults to current coord_type.
- **coord_kwargs** (*dict*, *optional*) -- Any desired coord_kwargs that will be passed to the RedundantCoords object.

Returns

geom -- New Geometry object with the same atoms and coordinates.

Return type

Geometry

copy_all(*coord_type=None*, *coord_kwargs=None*)

property covalent_radii

del_atoms(*inds*, ***kwargs*)

describe()

dump_xyz(*fn*, *cart_coords=None*, ***kwargs*)

eckart_projection(*mw_hessian*, *return_P=False*, *full=False*)

property energy

Energy of the current atomic configuration.

Returns

energy -- Energy of the current atomic configuration.

Return type

float

fd_coords3d_gen(*step_size=0.001*)

Iterator returning 3d Cartesians for finite-differences.

property forces

Energy of the current atomic configuration.

Returns

force -- 1d array containing the forces acting on the atoms. Negative of the gradient.

Return type

np.array

get_energy_and_cart_forces_at(*cart_coords*)

get_energy_and_cart_hessian_at(*cart_coords*)

get_energy_and_forces_at(*coords*)

Calculate forces and energies at the given coordinates.

The results are not saved in the Geometry object.

get_energy_at(*coords*)

get_energy_at_cart_coords(*cart_coords*)

get_fragments(*regex*)

get_imag_frequencies(*hessian=None*, *thresh=1e-06*)

get_normal_modes(*cart_hessian=None, full=False*)

Normal mode wavenumbers, eigenvalues and Cartesian displacements Hessian.

get_restart_info()

get_sphere_radius(*offset=4*)

get_subgeom(*indices, coord_type='cart', sort=False*)

Return a Geometry containing a subset of the current Geometry.

Parameters

- **indices** (*iterable of ints*) -- Atomic indices that the define the subset of the current Geometry.
- **coord_type** (*str, ("cart", "redund"), optional*) -- Coordinate system of the new Geometry.

Returns

sub_geom -- Subset of the current Geometry.

Return type

Geometry

get_subgeom_without(*indices, **kwargs*)

get_temporary_coords(*coords*)

get_thermoanalysis(*energy=None, cart_hessian=None, T=298.15, p=101325, point_group='c1'*)

get_trans_rot_projector(*full=False*)

property gradient

Negative of the force.

Returns

gradient -- 1d array containing the negative of the current forces.

Return type

np.array

property hessian

Matrix of second derivatives of the energy in respect to atomic displacements.

Returns

hessian -- 2d array containing the second derivatives of the energy with respect to atomic/coordinate displacements depending on the type of coordiante system.

Return type

np.array

property inertia_tensor

property is_analytical_2d

jmol(*atoms=None, cart_coords=None*)

Show geometry in jmol.

property layers

mass_weigh_hessian(*hessian*)

property masses

property masses_rep

property mm_inv

Inverted mass matrix.

Returns a diagonal matrix containing the inverted atomic masses.

property mm_sqrt_inv

Inverted square root of the mass matrix.

modes3d()

property moving_atoms

moving_atoms_jmol()

property mw_coords

Mass-weighted coordinates.

Returns

mw_coords -- 1d array containing the mass-weighted cartesian coordinates.

Return type

np.array

property mw_gradient

Mass-weighted gradient.

Returns

mw_gradient -- Returns the mass-weighted gradient.

Return type

np.array

property mw_hessian

Mass-weighted hessian.

Returns

mw_hessian -- 2d array containing the mass-weighted hessian $M^{(-1/2)} H M^{(-1/2)}$.

Return type

np.array

principal_axes_are_aligned()

Check if the principal axes are aligned with the cartesian axes.

Returns

aligned -- Whether the principal axes are aligned or not.

Return type

bool

reparametrize()

reset_coords(*new_typed_prims=None*)

rmsd(*geom*)

rotate(*copy=False, rng=None*)

set_calculator(*calculator*, *clear=True*)

Reset the object and set a calculator.

set_coord(*ind*, *coord*)

Set a coordinate by index.

Parameters

- **ind** (*int*) -- Index in of the coordinate to set in the self.coords array.
- **coord** (*float*) -- Coordinate value.

set_coords(*coords*, *cartesian=False*, *update_constraints=False*)

set_h5_hessian(*fn*)

set_restart_info(*restart_info*)

set_results(*results*)

Save the results from a dictionary.

Parameters

results (*dict*) -- The keys in this dict will be set as attributes in the current object, with the corresponding item as value.

standard_orientation()

property **sum_formula**

tmp_xyz_handle(*atoms=None*, *cart_coords=None*)

property **total_mass**

unweight_mw_hessian(*mw_hessian*)

Unweight a mass-weighted hessian.

Parameters

mw_hessian (*np.array*) -- Mass-weighted hessian to be unweighted.

Returns

hessian -- 2d array containing the hessian.

Return type

np.array

property **vdw_radii**

vdw_volume(***kwargs*)

without_hydrogens()

zero_frozen_forces(*cart_forces*)

`pysisyphus.Geometry.get_trans_rot_projector`(*cart_coords*, *masses*, *full=False*)

`pysisyphus.Geometry.get_trans_rot_vectors`(*cart_coords*, *masses*, *rot_thresh=1e-06*)

Vectors describing translation and rotation.

These vectors are used for the Eckart projection by constructing a projector from them.

See Martin J. Field - A Practical Introduction to the simulation of Molecular Systems, 2007, Cambridge University Press, Eq. (8.23), (8.24) and (8.26) for the actual projection.

See also <https://chemistry.stackexchange.com/a/74923>.

Parameters

- **cart_coords** (*np.array, 1d, shape (3 * atoms.size,)*) -- Atomic masses in amu.
- **masses** (*iterable, 1d, shape (atoms.size,)*) -- Atomic masses in amu.

Returns

ortho_vecs -- 2d array containing row vectors describing translations and rotations.

Return type

*np.array(6, 3*atoms.size)*

`pysisyphus.Geometry.inertia_tensor(coords3d, masses)`

Inertia tensor.

$x^2 \quad xy \quad xz \mid$

$(x \ y \ z)^T \cdot (x \ y \ z) = \mid xy \ y^2 \ yz \mid$

$xz \ yz \ z^2 \mid$

17.1.4 pysisyphus.TableFormatter module

class `pysisyphus.TableFormatter.TableFormatter`(*header, fmts, min_width=7, space=3*)

Bases: `object`

property `header`

join_format(*fmts, lst*)

Format a given iterable *lst* with formats given in the iterable *lst* and return the joined items of the formatted list.

line(**args*)

min_width_fmts(*raw_fmts=None*)

`pysisyphus.TableFormatter.run()`

17.1.5 pysisyphus.TablePrinter module

class `pysisyphus.TablePrinter.TablePrinter`(*header, col_fmts, width=12, sub_underline=True, shift_left=0, fmts_update=None, mark='*'*)

Bases: `object`

print(**args, **kwargs*)

print_header(*with_sep=True*)

print_row(*args, marks=None*)

print_sep()

17.1.6 pysisyphus.color module

`pysisyphus.color.bool_color(bool_, str_=None)`

`pysisyphus.color.green(str_)`

`pysisyphus.color.red(str_)`

17.1.7 pysisyphus.config module

`pysisyphus.config.detect_paths()`

`pysisyphus.config.get_cmd(section, key='cmd', use_defaults=True)`

`pysisyphus.config.parse_args(args)`

`pysisyphus.config.run_detect_paths()`

17.1.8 pysisyphus.constants module

17.1.9 pysisyphus.elem_data module

`pysisyphus.elem_data.get_tm_indices(atoms)`

`pysisyphus.elem_data.nuc_charges_for_atoms(atoms)`

17.1.10 pysisyphus.exceptions module

exception `pysisyphus.exceptions.HEIIsFirstOrLastException`

Bases: `Exception`

17.1.11 pysisyphus.filtertrj module

`pysisyphus.filtertrj.get_unique_internals(geom)`

`pysisyphus.filtertrj.parse_args(args)`

`pysisyphus.filtertrj.parse_filter(raw_filter)`

`pysisyphus.filtertrj.run()`

17.1.12 pysisyphus.helpers module

class `pysisyphus.helpers.FinalHessianResult(neg_eigvals, eigvals, nus, imag_fns, thermo)`

Bases: `tuple`

eigvals

Alias for field number 1

imag_fns

Alias for field number 3

neg_eigvals

Alias for field number 0

nus

Alias for field number 2

thermo

Alias for field number 4

`pysisyphus.helpers.align_coords(coords_list)``pysisyphus.helpers.align_geoms(geoms)``pysisyphus.helpers.check_for_end_sign(check_user=True, cwd='.', add_signs=None)``pysisyphus.helpers.complete_fragments(atoms, fragments)``pysisyphus.helpers.confirm_input(message)``pysisyphus.helpers.do_final_hessian(geom, save_hessian=True, write_imag_modes=False, is_ts=False, prefix='', T=298.15, p=101325, ev_thresh=-1e-06, print_thermo=False, out_dir=None)``pysisyphus.helpers.fit_rigid(geometry, vectors=None, vector_lists=None, hessian=None, align_factor=1.0)``pysisyphus.helpers.geom_from_xyz_file(xyz_fn, coord_type='cart', **coord_kwargs)``pysisyphus.helpers.geom_loader(fn, coord_type='cart', iterable=False, **coord_kwargs)`

After introducing the pubchem functionality I don't like this function anymore :) Too complicated.

`pysisyphus.helpers.geoms_from_trj(trj_fn, first=None, coord_type='cart', **coord_kwargs)``pysisyphus.helpers.get_coords_diffs(coords, align=False, normalize=True)``pysisyphus.helpers.get_geom_getter(ref_geom, calc_setter)``pysisyphus.helpers.get_tangent_trj_str(atoms, coords, tangent, comment=None, points=10, displ=None)``pysisyphus.helpers.imag_modes_from_geom(geom, freq_thresh=-10, points=10, displ=None)``pysisyphus.helpers.index_array_from_overlaps(overlaps, axis=1)`It is assumed that the overlaps between two points with indices *i* and *j* with (*j* > *i*) are computed and that *i* changes along the first axis (*axis*=0) and *j* changes along the second axis (*axis*=1).So the first row of the overlap matrix (*overlaps*[0]) should contain the overlaps between state 0 at index *i* and all states at index *j*.*argmax* along axis 1 returns the indices of the most overlapping states at index *j* with the states at index *i*, given by the item index in the indices array. E.g.:[0 1 3 2] indicates a root flip in a system with four states when going from index *i* to index *j*. Root 2 at *i* became root 3 at *j* and vice versa.`pysisyphus.helpers.match_geoms(ref_geom, geom_to_match, hydrogen=False)`**See**

[1] 10.1021/ci400534h [2] 10.1021/acs.jcim.6b00516

```
pysisyphus.helpers.norm_max_rms(arr)
pysisyphus.helpers.np_print(func, precision=2, suppress=True, linewidth=120)
pysisyphus.helpers.pick_image_inds(cart_coords, images)
    Pick approx. evenly distributed images from given Cartesian coordinates.
pysisyphus.helpers.print_barrier(ref_energy, comp_energy, ref_str, comp_str)
pysisyphus.helpers.procrustes(geometry, align_factor=1.0)
pysisyphus.helpers.shake_coords(coords, scale=0.1, seed=None)
pysisyphus.helpers.simple_peaks(data)
pysisyphus.helpers.slugify_worker(dask_worker)
```

17.1.13 pysisyphus.helpers_pure module

```
class pysisyphus.helpers_pure.OrderedEnum(value)
    Bases: Enum
    An enumeration.
pysisyphus.helpers_pure.approx_float(num, expected, abs_tol=1e-06, rel_tol=1e-12)

    Return type
    bool
pysisyphus.helpers_pure.cache_arrays(sources, dest)
pysisyphus.helpers_pure.check_mem(mem, pal, avail_frac=0.85, logger=None)
pysisyphus.helpers_pure.chunks(l, n)
    Yield successive n-sized chunks from l. https://stackoverflow.com/a/312464
pysisyphus.helpers_pure.describe(arr)
pysisyphus.helpers_pure.eigval_to_wavenumber(ev)
pysisyphus.helpers_pure.estimate(gen, elems)
pysisyphus.helpers_pure.expand(to_expand)
pysisyphus.helpers_pure.file_or_str(*args, method=False, mode='r', exact=False)
pysisyphus.helpers_pure.filter_fixture_store(test_name)
pysisyphus.helpers_pure.find_closest_sequence(str_, comp_strs)

    Return type
    Tuple[str, float]
pysisyphus.helpers_pure.full_expand(to_expand)
pysisyphus.helpers_pure.get_box(coords3d, offset=0.0)
pysisyphus.helpers_pure.get_clock()
```



```
pysisyphus.helpers_pure.get_cubic_crystal(l, n=2, atom='Na')
pysisyphus.helpers_pure.get_input(data, prompt, lbl_func=None)
pysisyphus.helpers_pure.get_molecular_radius(coords3d, min_offset=0.9452)
pysisyphus.helpers_pure.get_random_path(stem='')
pysisyphus.helpers_pure.get_ratio(str_, comp_str)
See https://stackoverflow.com/a/17388505
```

Return type

str

```
pysisyphus.helpers_pure.hash_args(*args, precision=4)
pysisyphus.helpers_pure.hash_arr(arr, precision=4)
pysisyphus.helpers_pure.highlight_text(text, width=80, level=0)
pysisyphus.helpers_pure.increment_fn(org_fn, suffix=None)
```

Append, or increase a suffixed counter on a given filename. If no counter is present it will be set to zero. Otherwise it is incremented by one.

```
>>> increment_fn("opt", "rebuilt")
'opt_rebuilt_000'
>>> increment_fn("opt")
'opt_000'
>>> increment_fn("opt_rebuilt_000", "rebuilt")
'opt_rebuilt_001'
```

Parameters

- **org_fn** (str) -- The original, unaltered filename.
- **suffix** (Optional[str], default: None) -- Optional suffix to be append.

Returns

Modified filename with optional suffix and incremented counter.

Return type

incr_fn

```
pysisyphus.helpers_pure.interpolate_colors(values, c1, c2, num=32)
```

Expects two RGB colors c1 and c2.

```
pysisyphus.helpers_pure.json_to_results(as_json)
```

```
pysisyphus.helpers_pure.kill_dir(path)
```

Innocent function remove a directory.

It must contain only files and no other directories. So this won't do too much damage hopefully.

```
pysisyphus.helpers_pure.log(logger, msg, level=10)
```

```
pysisyphus.helpers_pure.merge_sets(fragments)
```

Merge a list of iterables.

`pysisyphus.helpers_pure.molecular_volume(coords3d, vdw_radii, n_trial=10000, offset=1.0)`

Monte-Carlo estimate of molecular volume using Van der Waals spheres. Cartesian coordinates and VdW-radii are expected in Bohr!

Return type

Tuple[float, float, float]

`pysisyphus.helpers_pure.recursive_update(d, u)`

Recursive update of d with keys/values from u.

From <https://stackoverflow.com/questions/3232943>

`pysisyphus.helpers_pure.remove_duplicates(seq)`

`pysisyphus.helpers_pure.report_frozen_atoms(geom)`

`pysisyphus.helpers_pure.report_isotopes(geom, affect_str)`

`pysisyphus.helpers_pure.results_to_json(results)`

`pysisyphus.helpers_pure.rms(arr)`

`pysisyphus.helpers_pure.sort_by_central(set1, set2)`

Determines a common index in two sets and returns a length 3 tuple with the central index at the middle position and the two terminal indices as first and last indices.

`pysisyphus.helpers_pure.standard_state_corr(T=298.15, p=101325, n=1)`

dG for change of standard state from gas to solution of 1 mol/l

`pysisyphus.helpers_pure.timed(logger=None)`

`pysisyphus.helpers_pure.to_sets(iterable)`

`pysisyphus.helpers_pure.to_subscript_num(num)`

`pysisyphus.helpers_pure.touch(fn)`

17.1.14 pysisyphus.init_logging module

`pysisyphus.init_logging.get_fh_logger(name, log_fn)`

Initialize a logger with 'name', level DEBUG and a FileHandler.

`pysisyphus.init_logging.init_logging(log_dir='.', scheduler=None)`

Prepare the logger in log_path. When called with scheduler loggers for every worker are prepared.

`pysisyphus.init_logging.init_logging_base(dask_worker, log_path)`

Prepare individual loggers for one dask_worker.

17.1.15 pysisyphus.linalg module

`pysisyphus.linalg.cross3(a, b)`

10x as fast as `np.cross` for two 1d arrays of size 3.

`pysisyphus.linalg.eigvec_grad(w, v, ind, mat_grad)`

Gradient of 'ind'-th eigenvector.

$$dv_i / dx_i = (w_i I - mat)^{-1} dmat/dx_i v_i$$

`pysisyphus.linalg.finite_difference_hessian(coords, grad_func, step_size=0.01, acc=2, callback=None)`

Numerical Hessian from central finite gradient differences.

Return type

`ndarray[Any, dtype[float]]`

See central differences in

https://en.wikipedia.org/wiki/Finite_difference_coefficient

for the different accuracies.

`pysisyphus.linalg.get_rot_mat(abc=None)`

`pysisyphus.linalg.get_rot_mat_for_coords(coords3d_1, coords3d_2)`

`pysisyphus.linalg.gram_schmidt(vecs, thresh=1e-08)`

`pysisyphus.linalg.make_unit_vec(vec1, vec2)`

Return unit vector pointing from `vec2` to `vec1`.

`pysisyphus.linalg.matrix_power(mat, p, thresh=1e-12, strict=True)`

`pysisyphus.linalg.multi_component_sym_mat(arr, dim)`

`pysisyphus.linalg.norm3(a)`

5x as fast as `np.linalg.norm` for a 1d array of size 3.

`pysisyphus.linalg.orthogonalize_against(mat, vecs, max_cycles=5, thresh=1e-10)`

Orthogonalize rows of 'mat' against rows in 'vecs'

Returns a (modified) copy of `mat`.

`pysisyphus.linalg.perp_comp(vec, along)`

Return the perpendicular component of `vec` along `along`.

`pysisyphus.linalg.pivoted_cholesky(A, tol=-1.0)`

Cholesky factorization a real symmetric positive semidefinite matrix. Cholesky factorization is carried out with full pivoting.

Adapted from PySCF.

$$P.T * A * P = L * L.T$$

Parameters

- **A** (`ndarray[Any, dtype[TypeVar(_ScalarType_co, bound= generic, covariant=True)]]`) -- Matrix to be factorized.
- **tol** (`float`, default: `-1.0`) -- User defined tolerance, as outlined in the LAPACK docs.

Returns

- *L* -- Lower or upper triangular matrix.
- *piv* -- Pivot vectors, starting at 0.
- *rank* -- Rank of the factoirzed matrix.

`pysisyphus.linalg.pivoted_cholesky2(A, tol=None, m_max=0)`

<https://doi.org/10.1016/j.apnum.2011.10.001>

`pysisyphus.linalg.quaternion_to_rot_mat(q)`

`pysisyphus.linalg.rmsd_grad(coords3d, ref_coords3d, offset=1e-09)`

RMSD and gradient between two sets of coordinates from quaternions.

The gradient is given w.r.t. the coordinates of 'coords3d'.

Python adaption of

`ls_rmsd.f90`

from the xtb repository of the Grimme group, which in turn implements

[1] <https://doi.org/10.1002/jcc.20110>

.

Parameters

- **coords3d** (ndarray[Any, dtype[float]]) -- Coordinate array of shape (N, 3) with N denoting the number of atoms.
- **ref_coords3d** (ndarray[Any, dtype[float]]) -- Reference coordinates.
- **offset** (float, default: 1e-09) -- Small floating-point number that is added to the RMSD, to avoid division by zero.

Return type

Tuple[float, ndarray[Any, dtype[float]]]

Returns

- *rmsd* -- RMSD value.
- *rmsd_grad* -- Gradient of the RMSD value w.r.t. to 'coords3d'.

`pysisyphus.linalg.rot_quaternion(coords3d, ref_coords3d)`

`pysisyphus.linalg.svd_inv(array, thresh, hermitian=False)`

`pysisyphus.linalg.sym_mat_from_tril(arr, data)`

`pysisyphus.linalg.sym_mat_from_triu(arr, data)`

17.1.16 pysisyphus.pack module

`pysisyphus.pack.as_pdb(fn)`

`pysisyphus.pack.parse_args(args)`

`pysisyphus.pack.print_info(title, geom)`

`pysisyphus.pack.run()`

`pysisyphus.pack.sphere_radius_from_volume(volume)`

`pysisyphus.pack.volume_for_density(molecule_num, mol_mass, density)`

17.1.17 pysisyphus.peakdetect module

`pysisyphus.peakdetect.peakdetect(y_axis, x_axis=None, lookahead=200, delta=0)`

Converted from/based on a MATLAB script at: <http://billauer.co.il/peakdet.html>

function for detecting local maxima and minima in a signal. Discovers peaks by searching for values which are surrounded by lower or larger values for maxima and minima respectively

keyword arguments: **y_axis** -- A list containing the signal over which to find peaks

x_axis -- A x-axis whose values correspond to the y_axis list and is used

in the return to specify the position of the peaks. If omitted an index of the y_axis is used. (default: None)

lookahead -- distance to look ahead from a peak candidate to determine if

it is the actual peak (default: 200) '(samples / period) / f' where '4 >= f >= 1.25' might be a good value

delta -- this specifies a minimum difference between a peak and

the following points, before a peak may be considered a peak. Useful to hinder the function from picking up false peaks towards to end of the signal. To work well delta should be set to $\text{delta} \geq \text{RMSnoise} * 5$. (default: 0)

When omitted delta function causes a 20% decrease in speed. When used Correctly it can double the speed of the function

return: two lists [max_peaks, min_peaks] containing the positive and

negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: `np.mean(max_peaks, 0)[1]` on the results to unpack one of the lists into x, y coordinates do: `x, y = zip(*max_peaks)`

`pysisyphus.peakdetect.peakdetect_fft(y_axis, x_axis, pad_len=20)`

Performs a FFT calculation on the data and zero-pads the results to increase the time domain resolution after performing the inverse fft and send the data to the 'peakdetect' function for peak detection.

Omitting the x_axis is forbidden as it would make the resulting x_axis value silly if it was returned as the index 50.234 or similar.

Will find at least 1 less peak then the 'peakdetect_zero_crossing' function, but should result in a more precise value of the peak as resolution has been increased. Some peaks are lost in an attempt to minimize spectral leakage by calculating the fft between two zero crossings for n amount of signal periods.

The biggest time eater in this function is the ifft and thereafter it's the 'peakdetect' function which takes only half the time of the ifft. Speed improvements could include to check if $2*n$ points could be used for fft and ifft or change the 'peakdetect' to the 'peakdetect_zero_crossing', which is maybe 10 times faster than 'peakdetect'. The pro of 'peakdetect' is that it results in one less lost peak. It should also be noted that the time used by the ifft function can change greatly depending on the input.

keyword arguments: `y_axis` -- A list containing the signal over which to find peaks

`x_axis` -- A x-axis whose values correspond to the `y_axis` list and is used
in the return to specify the position of the peaks.

`pad_len` -- By how many times the time resolution should be
increased by, e.g. 1 doubles the resolution. The amount is rounded up to the nearest 2^n amount (default: 20)

return: two lists [`max_peaks`, `min_peaks`] containing the positive and
negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: `np.mean(max_peaks, 0)[1]` on the results to unpack one of the lists into x, y coordinates do: `x, y = zip(*max_peaks)`

`pysisyphus.peakdetect.peakdetect_parabola(y_axis, x_axis, points=31)`

Function for detecting local maxima and minima in a signal. Discovers peaks by fitting the model function: $y = k(x - \tau)^2 + m$ to the peaks. The amount of points used in the fitting is set by the points argument.

Omitting the `x_axis` is forbidden as it would make the resulting `x_axis` value silly, if it was returned as index 50.234 or similar.

will find the same amount of peaks as the 'peakdetect_zero_crossing' function, but might result in a more precise value of the peak.

keyword arguments: `y_axis` -- A list containing the signal over which to find peaks

`x_axis` -- A x-axis whose values correspond to the `y_axis` list and is used
in the return to specify the position of the peaks.

`points` -- How many points around the peak should be used during curve
fitting (default: 31)

return: two lists [`max_peaks`, `min_peaks`] containing the positive and
negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: `np.mean(max_peaks, 0)[1]` on the results to unpack one of the lists into x, y coordinates do: `x, y = zip(*max_peaks)`

`pysisyphus.peakdetect.peakdetect_sine(y_axis, x_axis, points=31, lock_frequency=False)`

Function for detecting local maxima and minima in a signal. Discovers peaks by fitting the model function: $y = A \sin(2 \pi f (x - \tau))$ to the peaks. The amount of points used in the fitting is set by the points argument.

Omitting the `x_axis` is forbidden as it would make the resulting `x_axis` value silly if it was returned as index 50.234 or similar.

will find the same amount of peaks as the 'peakdetect_zero_crossing' function, but might result in a more precise value of the peak.

The function might have some problems if the sine wave has a non-negligible total angle i.e. a kx component, as this messes with the internal offset calculation of the peaks, might be fixed by fitting a $y = kx + m$ function to the peaks for offset calculation.

keyword arguments: `y_axis` -- A list containing the signal over which to find peaks

`x_axis` -- A x-axis whose values correspond to the `y_axis` list and is used
in the return to specify the position of the peaks.

`points` -- How many points around the peak should be used during curve
fitting (default: 31)

`lock_frequency` -- Specifies if the frequency argument of the model
function should be locked to the value calculated from the raw peaks or if optimization process may tinker with it. (default: False)

return: two lists [max_peaks, min_peaks] containing the positive and

negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*max_peaks)

`pysisyphus.peakdetect.peakdetect_sine_locked(y_axis, x_axis, points=31)`

Convenience function for calling the 'peakdetect_sine' function with the lock_frequency argument as True.

keyword arguments: y_axis -- A list containing the signal over which to find peaks x_axis -- A x-axis whose values correspond to the y_axis list and is used

in the return to specify the position of the peaks.

points -- How many points around the peak should be used during curve

fitting (default: 31)

return: see the function 'peakdetect_sine'

`pysisyphus.peakdetect.peakdetect_spline(y_axis, x_axis, pad_len=20)`

Performs a b-spline interpolation on the data to increase resolution and send the data to the 'peakdetect_zero_crossing' function for peak detection.

Omitting the x_axis is forbidden as it would make the resulting x_axis value silly if it was returned as the index 50.234 or similar.

will find the same amount of peaks as the 'peakdetect_zero_crossing' function, but might result in a more precise value of the peak.

keyword arguments: y_axis -- A list containing the signal over which to find peaks

x_axis -- A x-axis whose values correspond to the y_axis list and is used

in the return to specify the position of the peaks. x-axis must be equally spaced.

pad_len -- By how many times the time resolution should be increased by,

e.g. 1 doubles the resolution. (default: 20)

return: two lists [max_peaks, min_peaks] containing the positive and

negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: np.mean(max_peaks, 0)[1] on the results to unpack one of the lists into x, y coordinates do: x, y = zip(*max_peaks)

`pysisyphus.peakdetect.peakdetect_zero_crossing(y_axis, x_axis=None, window=11)`

Function for detecting local maxima and minima in a signal. Discovers peaks by dividing the signal into bins and retrieving the maximum and minimum value of each the even and odd bins respectively. Division into bins is performed by smoothing the curve and finding the zero crossings.

Suitable for repeatable signals, where some noise is tolerated. Executes faster than 'peakdetect', although this function will break if the offset of the signal is too large. It should also be noted that the first and last peak will probably not be found, as this function only can find peaks between the first and last zero crossing.

keyword arguments: y_axis -- A list containing the signal over which to find peaks

x_axis -- A x-axis whose values correspond to the y_axis list

and is used in the return to specify the position of the peaks. If omitted an index of the y_axis is used. (default: None)

window -- the dimension of the smoothing window; should be an odd integer

(default: 11)

return: two lists [max_peaks, min_peaks] containing the positive and negative peaks respectively. Each cell of the lists contains a tuple of: (position, peak_value) to get the average peak value do: `np.mean(max_peaks, 0)[1]` on the results to unpack one of the lists into x, y coordinates do: `x, y = zip(*max_peaks)`

`pysisyphus.peakdetect.zero_crossings(y_axis, window_len=11, window_f='hanning', offset_corrected=False)`

Algorithm to find zero crossings. Smooths the curve and finds the zero-crossings by looking for a sign change.

keyword arguments: `y_axis` -- A list containing the signal over which to find zero-crossings

window_len -- the dimension of the smoothing window; should be an odd integer (default: 11)

window_f -- the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman' (default: 'hanning')

`offset_corrected` -- Used for recursive calling to remove offset when needed

return: the index for each zero-crossing

`pysisyphus.peakdetect.zero_crossings_sine_fit(y_axis, x_axis, fit_window=None, smooth_window=11)`

Detects the zero crossings of a signal by fitting a sine model function around the zero crossings: $y = A * \sin(2 * \pi * \text{Hz} * (x - \text{tau})) + k * x + m$ Only tau (the zero crossing) is varied during fitting.

Offset and a linear drift of offset is accounted for by fitting a linear function the negative respective positive raw peaks of the wave-shape and the amplitude is calculated using data from the offset calculation i.e. the 'm' constant from the negative peaks is subtracted from the positive one to obtain amplitude.

Frequency is calculated using the mean time between raw peaks.

Algorithm seems to be sensitive to first guess e.g. a large `smooth_window` will give an error in the results.

keyword arguments: `y_axis` -- A list containing the signal over which to find peaks

x_axis -- A x-axis whose values correspond to the `y_axis` list and is used in the return to specify the position of the peaks. If omitted an index of the `y_axis` is used. (default: None)

fit_window -- Number of points around the approximate zero crossing that should be used when fitting the sine wave. Must be small enough that no other zero crossing will be seen. If set to none then the mean distance between zero crossings will be used (default: None)

smooth_window -- the dimension of the smoothing window; should be an odd integer (default: 11)

return: A list containing the positions of all the zero crossings.

17.1.18 pysisyphus.plot module

`pysisyphus.plot.CDD_PNG_FNS = 'cdd_png_fns'`

Default to kJ mol^{-1} . `DE_LABEL` and `get_en_conv` will be overwritten when `run()` is executed. Both are still defined here, to make the functions from `plot.py` importable.

`pysisyphus.plot.anim_cos(cart_coords, energies)`

`pysisyphus.plot.get_en_conv()`

`pysisyphus.plot.get_force_unit(coord_type)`


```
pysisyphus.plot.list_h5_groups(h5_fn)
pysisyphus.plot.load_h5(h5_fn, h5_group, datasets=None, attrs=None)
pysisyphus.plot.parse_args(args)
pysisyphus.plot.plot_afir(h5_fn='afir.h5', h5_group='afir')
pysisyphus.plot.plot_all_energies(h5)
pysisyphus.plot.plot_cos_energies(h5_fn='optimization.h5', h5_group='opt')
pysisyphus.plot.plot_cos_forces(h5_fn='optimization.h5', h5_group='opt', last=15)
pysisyphus.plot.plot_cycle(cart_coords, energies)
pysisyphus.plot.plot_gau(gau_fns, num=50)
pysisyphus.plot.plot_irc()
pysisyphus.plot.plot_irc_h5(h5, title=None)
pysisyphus.plot.plot_md(h5_group='run')
pysisyphus.plot.plot_opt(h5_fn='optimization.h5', h5_group='opt')
pysisyphus.plot.plot_overlaps(h5, thresh=0.1)
pysisyphus.plot.plot_scan(h5_fn='scan.h5')
pysisyphus.plot.plot_trj_energies(trj)
    Parse comments of .xyz/.trj as energies and plot.
pysisyphus.plot.render_cdds(h5)
pysisyphus.plot.run()
pysisyphus.plot.spline_plot_cycles(cart_coords, energies)
```

17.1.19 pysisyphus.run module

```
class pysisyphus.run.RunResult(preopt_first_geom, preopt_last_geom, cos, cos_opt, ts_geom, ts_opt,
                                end_geoms, irc, irc_geom, mdp_result, opt_geom, opt, calced_geoms,
                                calced_results, stochastic, calc_getter, scan_geoms, scan_vals,
                                scan_energies, perf_results)
```

Bases: tuple

calc_getter

Alias for field number 15

calced_geoms

Alias for field number 12

calced_results

Alias for field number 13

cos

Alias for field number 2

cos_opt

Alias for field number 3

end_geoms

Alias for field number 6

irc

Alias for field number 7

irc_geom

Alias for field number 8

mdp_result

Alias for field number 9

opt

Alias for field number 11

opt_geom

Alias for field number 10

perf_results

Alias for field number 19

preopt_first_geom

Alias for field number 0

preopt_last_geom

Alias for field number 1

scan_energies

Alias for field number 18

scan_geoms

Alias for field number 16

scan_vals

Alias for field number 17

stochastic

Alias for field number 14

ts_geom

Alias for field number 4

ts_opt

Alias for field number 5

`pysisyphus.run.check_asserts(results, run_dict)`

`pysisyphus.run.copy_yaml_and_geometries(run_dict, yaml_fn, dest_and_add_cp, new_yaml_fn=None)`

`pysisyphus.run.do_clean(force=False)`

Deletes files from previous runs in the cwd. A similar function could be used to store everything ...

`pysisyphus.run.do_rmsds(xyz, geoms, end_fns, end_geoms, preopt_map=None, similar_thresh=0.25)`

`pysisyphus.run.get_calc_closure(base_name, calc_key, calc_kwargs, iter_dict=None, index=None)`

```
pysisyphus.run.get_defaults(conf_dict, T_default=298.15, p_default=101325)
pysisyphus.run.get_last_calc_cycle()
pysisyphus.run.load_run_dict(yaml_fn)
pysisyphus.run.main(run_dict, restart=False, yaml_dir='.', scheduler=None)
pysisyphus.run.parse_args(args)
pysisyphus.run.print_bibtex()
pysisyphus.run.print_header()
    Generated from https://asciitgen.now.sh/?s=pysisyphus&style=colossal
pysisyphus.run.run()
pysisyphus.run.run_calculations(geoms, calc_getter, scheduler=None, assert_track=False, run_func=None)
pysisyphus.run.run_endopt(irc, endopt_key, endopt_kwargs, calc_getter)
pysisyphus.run.run_from_dict(run_dict, cwd=None, set_defaults=True, yaml_fn=None, cp=None,
                             scheduler=None, clean=False, fclean=False, version=False, restart=False)
pysisyphus.run.run_irc(geom, irc_key, irc_kwargs, calc_getter)
pysisyphus.run.run_md(geom, calc_getter, md_kwargs)
pysisyphus.run.run_mdp(geom, calc_getter, mdp_kwargs)
pysisyphus.run.run_preopt(first_geom, last_geom, calc_getter, preopt_key, preopt_kwargs)
pysisyphus.run.run_scan(geom, calc_getter, scan_kwargs, callback=None)
pysisyphus.run.run_stochastic(stoc)
pysisyphus.run.run_tsopt_from_cos(cos, tsopt_key, tsopt_kwargs, calc_getter=None, ovlp_thresh=0.4,
                                  coordinate_union='bonds')
pysisyphus.run.setup_run_dict(run_dict)
```

17.1.20 pysisyphus.socket_helper module

```
pysisyphus.socket_helper.get_fmts(cartesians)
pysisyphus.socket_helper.recv_closure(sock, hdrln, fmts, verbose=False)
pysisyphus.socket_helper.send_closure(sock, hdrln, fmts, verbose=False)
```

17.1.21 pysisyphus.testing module

class pysisyphus.testing.DummyMark(*available*)

Bases: object

pysisyphus.testing.available(*calculator*, ***kwargs*)

pysisyphus.testing.module_available(*calculator*)

pysisyphus.testing.using(*calculator*, *set_pytest_mark=True*)

Calling disabling set_pytest_mark avoids a runtime dependency on pytest.

17.1.22 pysisyphus.thermo module

pysisyphus.thermo.get_thermoanalysis_from_hess_h5(*h5_fn*, *T=298.15*, *p=101325*, *point_group='c1'*,
return_geom=False)

pysisyphus.thermo.print_thermoanalysis(*thermo*, *geom=None*, *is_ts=False*, *unit='joule'*, *level=0*,
title=None)

Print thermochemical analysis.

17.1.23 pysisyphus.trj module

exception pysisyphus.trj.GotNoGeometryException

Bases: Exception

pysisyphus.trj.align(*geoms*)

Align all geometries onto the first using partical procrustes.

pysisyphus.trj.append(*geoms*)

pysisyphus.trj.center(*geoms*)

pysisyphus.trj.centerm(*geoms*)

pysisyphus.trj.dump_geoms(*geoms*, *fn_base*, *trj_infix=""*, *dump_trj=True*, *dump_xyz=True*, *dump_pdb=False*,
ang=False)

pysisyphus.trj.every(*geoms*, *every_nth*)

pysisyphus.trj.frag_sort(*geoms*)

pysisyphus.trj.get(*geoms*, *index*)

pysisyphus.trj.get_geoms(*xyz_fns*, *coord_type='cart'*, *geom_kwargs=None*, *union=False*, *same_prims=True*,
quiet=False, *interpol_kwargs=None*)

Returns a list of Geometry objects in the given coordinate system and interpolates if necessary.

pysisyphus.trj.get_interactively(*geoms*)

pysisyphus.trj.hardsphere_merge(*geoms*)

pysisyphus.trj.match(*ref_geom*, *geom_to_match*)

pysisyphus.trj.origin(*geoms*)

```
pysisyphus.trj.parse_args(args)
pysisyphus.trj.print_internal_vals(geoms, typed_prim)
pysisyphus.trj.print_internals(geoms, filter_atoms=None, add_prims='')
pysisyphus.trj.read_geoms(xyz_fns, coord_type='cart', geom_kwargs=None)
pysisyphus.trj.run()
pysisyphus.trj.shake(geoms, scale=0.1, seed=None)
pysisyphus.trj.spline_redistribute(geoms)
pysisyphus.trj.standard_orientation(geoms)
pysisyphus.trj.standardize_geoms(geoms, coord_type, geom_kwargs, same_prims=True, union=False)
pysisyphus.trj.translate(geoms, trans)
```

17.1.24 pysisyphus.version module

17.1.25 pysisyphus.xyzloader module

```
pysisyphus.xyzloader.coords_to_trj(trj_fn, atoms, coords_list, comments=None)
pysisyphus.xyzloader.make_trj_str(atoms, coords_list, comments=None)
pysisyphus.xyzloader.make_trj_str_from_geoms(geoms, comments=None, energy_comments=False)
pysisyphus.xyzloader.make_xyz_str(atoms, coords, comment='')
pysisyphus.xyzloader.parse_trj_file(trj_fn, with_comments=False)
pysisyphus.xyzloader.parse_trj_str(trj_str, with_comments=False)
pysisyphus.xyzloader.parse_xyz_file(xyz_fn, with_comment=False)
pysisyphus.xyzloader.parse_xyz_str(xyz_str, with_comment)
```

Parse a xyz string.

17.1.25.1 Paramters

xyz_str

[str] The contents of a .xyz file.

with_comment

[bool] Return comment line if True.

returns

- **atoms** (*list*) -- List of length N (N = number of atoms) holding the element symbols.
- **coords** (*np.array*) -- An array of shape (N, 3) holding the xyz coordinates.
- **comment_line** (*str, optional*) -- Comment line if with_comment argument was True.

`pysisyphus.xyzloader.split_xyz_str(xyz_str)`

Example:

xyz:

1

X -1.2 1.4 0.0 1

X 2.0 4.0 0.0

`pysisyphus.xyzloader.write_geoms_to_trj(geoms, fn, comments=None)`

17.1.26 pysisyphus.yaml_mods module

`pysisyphus.yaml_mods.get_constructor(unit)`

`pysisyphus.yaml_mods.get_loader(units=['Eh', 'kJpermol', 'kcalpermol', 'eV', 'fs', 'ps', 'ns', 'a0', 'angstrom', 'nm', 'deg'])`

17.1.27 Module contents

INDICES AND TABLES

- search

PYTHON MODULE INDEX

p

pysisyphus, 334
pysisyphus.benchmarks, 152
pysisyphus.benchmarks.Benchmark, 151
pysisyphus.benchmarks.data, 151
pysisyphus.calculators, 193
pysisyphus.calculators.AFIR, 72
pysisyphus.calculators.AnaPot, 154
pysisyphus.calculators.AnaPot2, 154
pysisyphus.calculators.AnaPot3, 155
pysisyphus.calculators.AnaPot4, 155
pysisyphus.calculators.AnaPotBase, 78
pysisyphus.calculators.AnaPotCBM, 156
pysisyphus.calculators.AtomAtomTransTorque, 156
pysisyphus.calculators.Calculator, 48
pysisyphus.calculators.CerjanMiller, 162
pysisyphus.calculators.CFOUR, 68
pysisyphus.calculators.Composite, 162
pysisyphus.calculators.ConicalIntersection, 162
pysisyphus.calculators.Dalton, 68
pysisyphus.calculators.DFTBp, 63
pysisyphus.calculators.DFTD3, 71
pysisyphus.calculators.Dimer, 76
pysisyphus.calculators.Dummy, 166
pysisyphus.calculators.EGO, 166
pysisyphus.calculators.EnergyMin, 167
pysisyphus.calculators.ExternalPotential, 168
pysisyphus.calculators.ExternalPotential.ExternalPotential, 70
pysisyphus.calculators.ExternalPotential.HarmonicSphere, 71
pysisyphus.calculators.ExternalPotential.LogForm, 71
pysisyphus.calculators.ExternalPotential.Restrain, 71
pysisyphus.calculators.FakeASE, 79
pysisyphus.calculators.FourWellAnaPot, 169
pysisyphus.calculators.FreeEndNEBPot, 170
pysisyphus.calculators.Gaussian09, 55
pysisyphus.calculators.Gaussian16, 56
pysisyphus.calculators.HardSphere, 171
pysisyphus.calculators.IDPPCalculator, 172
pysisyphus.calculators.IPIClient, 172
pysisyphus.calculators.IPIServer, 172
pysisyphus.calculators.LennardJones, 79
pysisyphus.calculators.LEPSBase, 173
pysisyphus.calculators.LEPSExpr, 173
pysisyphus.calculators.MOPAC, 64
pysisyphus.calculators.MullerBrownSymPyPot, 175
pysisyphus.calculators.MultiCalc, 175
pysisyphus.calculators.Obabel, 68
pysisyphus.calculators.ONIOMv2, 74
pysisyphus.calculators.OpenMM, 180
pysisyphus.calculators.OpenMolcas, 58
pysisyphus.calculators.ORCA, 59
pysisyphus.calculators.ORCA5, 180
pysisyphus.calculators.OverlapCalculator, 53
pysisyphus.calculators.parser, 192
pysisyphus.calculators.Psi4, 65
pysisyphus.calculators.PyPsi4, 184
pysisyphus.calculators.PyXTB, 184
pysisyphus.calculators.Rastrigin, 185
pysisyphus.calculators.Remote, 185
pysisyphus.calculators.Rosenbrock, 185
pysisyphus.calculators.SocketCalc, 185
pysisyphus.calculators.TIP3P, 79
pysisyphus.calculators.TransTorque, 186
pysisyphus.calculators.Turbomole, 61
pysisyphus.calculators.WFOWrapper, 189
pysisyphus.calculators.WFOWrapper2, 189
pysisyphus.calculators.XTB, 66
pysisyphus.color, 318
pysisyphus.config, 318
pysisyphus.constants, 318
pysisyphus.cos, 221
pysisyphus.cos.AdaptiveNEB, 111
pysisyphus.cos.ChainOfStates, 108
pysisyphus.cos.FreeEndNEB, 112
pysisyphus.cos.FreezingString, 217
pysisyphus.cos.GrowingChainOfStates, 112
pysisyphus.cos.GrowingNT, 218

pysisyphus.cos.GrowingString, 113
pysisyphus.cos.NEB, 110
pysisyphus.cos.SimpleZTS, 112
pysisyphus.db, 222
pysisyphus.db.generate_db, 221
pysisyphus.db.helpers, 221
pysisyphus.db.level, 221
pysisyphus.db.molecules, 221
pysisyphus.drivers, 231
pysisyphus.drivers.afir, 222
pysisyphus.drivers.barriers, 224
pysisyphus.drivers.birkholz, 224
pysisyphus.drivers.merge, 224
pysisyphus.drivers.opt, 225
pysisyphus.drivers.perf, 226
pysisyphus.drivers.pka, 226
pysisyphus.drivers.precon_pos_rot, 226
pysisyphus.drivers.rates, 227
pysisyphus.drivers.replace, 230
pysisyphus.drivers.scan, 231
pysisyphus.drivers.thermo, 231
pysisyphus.dynamics, 237
pysisyphus.dynamics.colvars, 232
pysisyphus.dynamics.driver, 232
pysisyphus.dynamics.Gaussian, 231
pysisyphus.dynamics.helpers, 233
pysisyphus.dynamics.lincs, 236
pysisyphus.dynamics.mdp, 236
pysisyphus.dynamics.rattle, 236
pysisyphus.dynamics.thermostats, 237
pysisyphus.elem_data, 318
pysisyphus.exceptions, 318
pysisyphus.filtertrj, 318
pysisyphus.Geometry, 35
pysisyphus.helpers, 318
pysisyphus.helpers_pure, 320
pysisyphus.init_logging, 322
pysisyphus.intcoords, 257
pysisyphus.intcoords.augment_bonds, 248
pysisyphus.intcoords.Bend, 237
pysisyphus.intcoords.Bend2, 237
pysisyphus.intcoords.BondedFragment, 238
pysisyphus.intcoords.Cartesian, 238
pysisyphus.intcoords.CartesianCoords, 238
pysisyphus.intcoords.Coords, 239
pysisyphus.intcoords.derivatives, 248
pysisyphus.intcoords.DistanceFunction, 241
pysisyphus.intcoords.DLC, 240
pysisyphus.intcoords.DummyTorsion, 241
pysisyphus.intcoords.eval, 249
pysisyphus.intcoords.exceptions, 250
pysisyphus.intcoords.findiffs, 250
pysisyphus.intcoords.generate_derivatives, 250
pysisyphus.intcoords.helpers, 251
pysisyphus.intcoords.LinearBend, 241
pysisyphus.intcoords.LinearDisplacement, 241
pysisyphus.intcoords.mp_derivatives, 252
pysisyphus.intcoords.OutOfPlane, 242
pysisyphus.intcoords.Primitive, 244
pysisyphus.intcoords.PrimTypes, 242
pysisyphus.intcoords.RedundantCoords, 42
pysisyphus.intcoords.Rotation, 246
pysisyphus.intcoords.setup, 253
pysisyphus.intcoords.setup_fast, 256
pysisyphus.intcoords.Stretch, 247
pysisyphus.intcoords.Torsion, 247
pysisyphus.intcoords.Torsion2, 247
pysisyphus.intcoords.Translation, 247
pysisyphus.intcoords.update, 256
pysisyphus.intcoords.valid, 257
pysisyphus.interpolate, 264
pysisyphus.interpolate.helpers, 264
pysisyphus.interpolate.IDPP, 263
pysisyphus.interpolate.Interpolator, 263
pysisyphus.interpolate.LST, 263
pysisyphus.interpolate.Redund, 263
pysisyphus.io, 267
pysisyphus.io.cjson, 264
pysisyphus.io.crd, 264
pysisyphus.io.hdf5, 264
pysisyphus.io.hessian, 265
pysisyphus.io.mol2, 265
pysisyphus.io.molden, 265
pysisyphus.io.pdb, 265
pysisyphus.io.pubchem, 265
pysisyphus.io.sdf, 266
pysisyphus.io.xyz, 266
pysisyphus.io.zmat, 266
pysisyphus.irc, 276
pysisyphus.irc.DampedVelocityVerlet, 126
pysisyphus.irc.DWI, 267
pysisyphus.irc.Euler, 127
pysisyphus.irc.EulerPC, 127
pysisyphus.irc.GonzalezSchlegel, 127
pysisyphus.irc.IMKMod, 128
pysisyphus.irc.initial_displ, 275
pysisyphus.irc.Instanton, 272
pysisyphus.irc.IRC, 124
pysisyphus.irc.IRCDummy, 272
pysisyphus.irc.LQA, 128
pysisyphus.irc.ModeKill, 274
pysisyphus.irc.ParamPlot, 274
pysisyphus.irc.RK4, 128
pysisyphus.linalg, 323
pysisyphus.line_searches, 280
pysisyphus.line_searches.Backtracking, 278
pysisyphus.line_searches.HagerZhang, 278

pysisyphus.line_searches.interpol, 280
pysisyphus.line_searches.LineSearch, 279
pysisyphus.line_searches.StrongWolfe, 280
pysisyphus.modelfollow, 283
pysisyphus.modelfollow.davidson, 282
pysisyphus.modelfollow.lanczos, 283
pysisyphus.modelfollow.NormalMode, 282
pysisyphus.optimizers, 301
pysisyphus.optimizers.BacktrackingOptimizer, 284
pysisyphus.optimizers.BFGS, 284
pysisyphus.optimizers.closures, 296
pysisyphus.optimizers.cls_map, 297
pysisyphus.optimizers.ConjugateGradient, 285
pysisyphus.optimizers.CubicNewton, 285
pysisyphus.optimizers.exceptions, 297
pysisyphus.optimizers.FIRE, 285
pysisyphus.optimizers.gdiis, 297
pysisyphus.optimizers.guess_hessians, 298
pysisyphus.optimizers.hessian_updates, 299
pysisyphus.optimizers.HessianOptimizer, 87
pysisyphus.optimizers.LayerOpt, 96
pysisyphus.optimizers.LBFGS, 96
pysisyphus.optimizers.MicroOptimizer, 289
pysisyphus.optimizers.NCOptimizer, 290
pysisyphus.optimizers.Optimizer, 84
pysisyphus.optimizers.poly_fit, 300
pysisyphus.optimizers.precon, 301
pysisyphus.optimizers.PreconLBFGS, 97
pysisyphus.optimizers.PreconSteepestDescent, 294
pysisyphus.optimizers.QuickMin, 294
pysisyphus.optimizers.restrict_step, 301
pysisyphus.optimizers.RFOptimizer, 89
pysisyphus.optimizers.RSA, 295
pysisyphus.optimizers.StabilizedQNMethod, 295
pysisyphus.optimizers.SteepestDescent, 296
pysisyphus.optimizers.StringOptimizer, 296
pysisyphus.pack, 325
pysisyphus.peakdetect, 325
pysisyphus.plot, 328
pysisyphus.plotters, 302
pysisyphus.plotters.AnimPlot, 302
pysisyphus.run, 329
pysisyphus.socket_helper, 331
pysisyphus.stochastic, 304
pysisyphus.stochastic.align, 304
pysisyphus.stochastic.FragmentKick, 302
pysisyphus.stochastic.Kick, 303
pysisyphus.stochastic.Pipeline, 303
pysisyphus.TableFormatter, 317
pysisyphus.TablePrinter, 317
pysisyphus.testing, 332
pysisyphus.tests, 305
pysisyphus.thermo, 332
pysisyphus.trj, 332
pysisyphus.tsoptimizers, 308
pysisyphus.tsoptimizers.RSIRFOptimizer, 120
pysisyphus.tsoptimizers.RSPRFOptimizer, 120
pysisyphus.tsoptimizers.TRIM, 120
pysisyphus.tsoptimizers.TSHessianOptimizer, 118
pysisyphus.version, 333
pysisyphus.wrapper, 309
pysisyphus.wrapper.jmol, 308
pysisyphus.wrapper.mwfn, 309
pysisyphus.wrapper.packmol, 309
pysisyphus.xyzloader, 333
pysisyphus.yaml_mods, 334

Symbols

- `__init__()` (*pysisyphus.Geometry.Geometry* method), 35, 309
- `__init__()` (*pysisyphus.calculators.AFIR* method), 193
- `__init__()` (*pysisyphus.calculators.AFIR.AFIR* method), 72, 152
- `__init__()` (*pysisyphus.calculators.AtomAtomTransTorque* method), 194
- `__init__()` (*pysisyphus.calculators.AtomAtomTransTorque.AtomAtomTransTorque* method), 156
- `__init__()` (*pysisyphus.calculators.CFOUR* method), 194
- `__init__()` (*pysisyphus.calculators.CFOUR.CFOUR* method), 68
- `__init__()` (*pysisyphus.calculators.Calculator.Calculator* method), 48, 156
- `__init__()` (*pysisyphus.calculators.EnergyMin* method), 199
- `__init__()` (*pysisyphus.calculators.EnergyMin.EnergyMin* method), 167
- `__init__()` (*pysisyphus.calculators.ExternalPotential.LogFermi* method), 168
- `__init__()` (*pysisyphus.calculators.ExternalPotential.RMSD* method), 168
- `__init__()` (*pysisyphus.calculators.FreeEndNEBPot.FreeEndNEBPot* method), 170
- `__init__()` (*pysisyphus.calculators.HardSphere* method), 202
- `__init__()` (*pysisyphus.calculators.HardSphere.HardSphere* method), 171
- `__init__()` (*pysisyphus.calculators.HardSphere.PWHardSphere* method), 171
- `__init__()` (*pysisyphus.calculators.LEPSEExpr.LEPSEExpr* method), 173
- `__init__()` (*pysisyphus.calculators.ONIOM* method), 205
- `__init__()` (*pysisyphus.calculators.ONIOMv2.ONIOM* method), 75, 177
- `__init__()` (*pysisyphus.calculators.ORCA* method), 206
- `__init__()` (*pysisyphus.calculators.ORCA.ORCA* method), 59, 178
- `__init__()` (*pysisyphus.calculators.TransTorque* method), 210
- `__init__()` (*pysisyphus.calculators.TransTorque.TransTorque* method), 186
- `__init__()` (*pysisyphus.calculators.XTB* method), 212
- `__init__()` (*pysisyphus.calculators.XTB.XTB* method), 66, 190
- `__init__()` (*pysisyphus.cos.AdaptiveNEB.AdaptiveNEB* method), 111, 214
- `__init__()` (*pysisyphus.cos.FreeEndNEB.FreeEndNEB* method), 112, 217
- `__init__()` (*pysisyphus.dynamics.Gaussian.Gaussian* method), 231
- `__init__()` (*pysisyphus.irc.DWI.DWI* method), 267
- `__init__()` (*pysisyphus.irc.IRC.IRC* method), 124, 269
- `__init__()` (*pysisyphus.line_searches.Backtracking* method), 280
- `__init__()` (*pysisyphus.line_searches.Backtracking.Backtracking* method), 278
- `__init__()` (*pysisyphus.line_searches.StrongWolfe* method), 281
- `__init__()` (*pysisyphus.line_searches.StrongWolfe.StrongWolfe* method), 280
- `__init__()` (*pysisyphus.modefollow.NormalMode* method), 283
- `__init__()` (*pysisyphus.modefollow.NormalMode.NormalMode* method), 282
- `__init__()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 87, 285
- `__init__()` (*pysisyphus.optimizers.LBFGS.LBFGS* method), 96, 287
- `__init__()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 84, 290
- `__init__()` (*pysisyphus.optimizers.PreconLBFGS.PreconLBFGS* method), 97, 293
- `__init__()` (*pysisyphus.optimizers.RFOptimizer.RFOptimizer* method), 89, 294
- `__init__()` (*pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer* method), 118, 306

A

- `a` (*pysisyphus.io.zmat.ZLine* attribute), 266

- `a8()` (in module `pysisyphus.io.crd`), 264
- `action()` (`pysisyphus.irc.Instanton.Instanton` method), 272
- `action_gradient()` (`pysisyphus.irc.Instanton.Instanton` method), 272
- `action_hessian()` (`pysisyphus.irc.Instanton.Instanton` method), 273
- `adapt_this_cycle()` (`pysisyphus.cos.AdaptiveNEB.AdaptiveNEB` method), 111, 214
- `AdaptiveNEB` (class in `pysisyphus.cos.AdaptiveNEB`), 111, 214
- `add_gaussian()` (`pysisyphus.calculators.Dimer` method), 198
- `add_gaussian()` (`pysisyphus.calculators.Dimer.Dimer` method), 76, 121, 165
- `adjust_alpha()` (`pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod` method), 295
- `adjust_alpha_stretch()` (`pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod` method), 295
- `AFIR` (class in `pysisyphus.calculators`), 193
- `AFIR` (class in `pysisyphus.calculators.AFIR`), 72, 152
- `afir_closure()` (in module `pysisyphus.calculators.AFIR`), 73, 154
- `afir_fd_hessian_wrapper()` (`pysisyphus.calculators.AFIR` method), 194
- `afir_fd_hessian_wrapper()` (`pysisyphus.calculators.AFIR.AFIR` method), 73, 153
- `AFIRPath` (class in `pysisyphus.drivers.afir`), 222
- `aHOH` (`pysisyphus.calculators.TIP3P` attribute), 210
- `aHOH` (`pysisyphus.calculators.TIP3P.TIP3P` attribute), 79, 186
- `aind` (`pysisyphus.io.zmat.ZLine` attribute), 266
- `align()` (in module `pysisyphus.trj`), 332
- `align_coords()` (in module `pysisyphus.helpers`), 319
- `align_geoms()` (in module `pysisyphus.helpers`), 319
- `align_on_subset()` (in module `pysisyphus.drivers.merge`), 224
- `align_principal_axes()` (`pysisyphus.Geometry.Geometry` method), 35, 310
- `ALL` (`pysisyphus.calculators.Calculator.KeepKind` attribute), 53, 161
- `all_coords` (`pysisyphus.irc.IRCDummy.IRCDummy` attribute), 272
- `all_energies` (`pysisyphus.Geometry.Geometry` property), 35, 310
- `all_geoms_to_trj()` (`pysisyphus.interpolate.Interpolator.Interpolator` method), 263
- `all_overlaps()` (`pysisyphus.calculators.WFOWrapper2.WFOWrapper2` method), 189
- `allcoords` (`pysisyphus.cos.FreezingString.FreezingString` property), 217
- `alpha` (`pysisyphus.line_searches.LineSearch.LineSearchResult` attribute), 279
- `alpha_new_from_phi()` (`pysisyphus.line_searches.Backtracking` method), 280
- `alpha_new_from_phi()` (`pysisyphus.line_searches.Backtracking.Backtracking` method), 278
- `alpha_new_from_phi_dphi()` (`pysisyphus.line_searches.Backtracking` method), 281
- `alpha_new_from_phi_dphi()` (`pysisyphus.line_searches.Backtracking.Backtracking` method), 278
- `analyze_afir_path()` (in module `pysisyphus.drivers.afir`), 222
- `AnaPot` (class in `pysisyphus.calculators.Anapot`), 154
- `AnaPot2` (class in `pysisyphus.calculators.Anapot2`), 154
- `AnaPot2_` (class in `pysisyphus.calculators.Anapot2`), 154
- `AnaPot3` (class in `pysisyphus.calculators.Anapot3`), 155
- `AnaPot4` (class in `pysisyphus.calculators.Anapot4`), 155
- `AnaPotBase` (class in `pysisyphus.calculators.AnapotBase`), 78, 155
- `AnaPotCBM` (class in `pysisyphus.calculators.AnapotCBM`), 156
- `anim_coords()` (`pysisyphus.calculators.AnapotBase.AnapotBase` method), 78, 155
- `anim_cos()` (in module `pysisyphus.plot`), 328
- `anim_opt()` (`pysisyphus.calculators.AnapotBase.AnapotBase` method), 78, 155
- `animate()` (`pysisyphus.plotters.AnimPlot.AnimPlot` method), 302
- `AnimPlot` (class in `pysisyphus.plotters.AnimPlot`), 302
- `append()` (in module `pysisyphus.trj`), 332
- `append_control()` (`pysisyphus.calculators.Turbomole` method), 211
- `append_control()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 61, 187
- `apply_keep_kind()` (`pysisyphus.calculators.Calculator.Calculator` method), 48, 157
- `apply_set_plans()` (`pysisyphus.calculators.Calculator.Calculator` method), 49, 157
- `apply_transform()` (in module `pysisyphus.stochastic.align`), 304
- `approx_float()` (in module `pysisyphus.helpers_pure`), 320
- `approximate_radius()` (`pysisyphus` method), 189

- phus.Geometry.Geometry method*), 36, 310
- arc_dims* (*pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates* property), 112, 218
- are_collinear()* (in module *pysisyphus.intcoords.valid*), 257
- as_ase_atoms()* (*pysisyphus.Geometry.Geometry method*), 36, 310
- as_calculator()* (*pysisyphus.calculators.ONIOMv2.Model method*), 74, 176
- as_g98_list()* (*pysisyphus.Geometry.Geometry method*), 36, 310
- as_geom()* (*pysisyphus.calculators.ONIOMv2.Model method*), 74, 176
- as_html5()* (*pysisyphus.plotters.AnimPlot.AnimPlot method*), 302
- as_pdb()* (in module *pysisyphus.pack*), 325
- as_xyz()* (*pysisyphus.cos.ChainOfStates.ChainOfStates method*), 108, 215
- as_xyz()* (*pysisyphus.cos.FreezingString.FreezingString method*), 217
- as_xyz()* (*pysisyphus.cos.GrowingNT.GrowingNT method*), 218
- as_xyz()* (*pysisyphus.Geometry.Geometry method*), 36, 310
- as_xyz()* (*pysisyphus.irc.Instanton.Instanton method*), 273
- assert_cart_coords()* (*pysisyphus.Geometry.Geometry method*), 36, 311
- assert_compatibility()* (*pysisyphus.Geometry.Geometry method*), 36, 311
- atom* (*pysisyphus.calculators.ONIOMv2.Link attribute*), 74, 176
- atom* (*pysisyphus.io.zmat.ZLine attribute*), 266
- atom_indices()* (*pysisyphus.Geometry.Geometry method*), 36, 311
- atom_inds_in_layer()* (*pysisyphus.calculators.ONIOM method*), 205
- atom_inds_in_layer()* (*pysisyphus.calculators.ONIOMv2.ONIOM method*), 75, 177
- atom_inds_to_cart_inds()* (in module *pysisyphus.calculators.ONIOMv2*), 76, 178
- atom_types* (*pysisyphus.Geometry.Geometry property*), 36, 311
- atom_xyz_iter()* (*pysisyphus.Geometry.Geometry method*), 36, 311
- AtomAtomTransTorque* (class in *pysisyphus.calculators*), 194
- AtomAtomTransTorque* (class in *pysisyphus.calculators.AtomAtomTransTorque*), 156
- atomic_numbers* (*pysisyphus.Geometry.Geometry property*), 36, 311
- atoms* (*pysisyphus.cos.ChainOfStates.ChainOfStates property*), 108, 215
- atoms* (*pysisyphus.cos.GrowingNT.GrowingNT property*), 218
- atoms* (*pysisyphus.drivers.afir.AFIRPath attribute*), 222
- atoms* (*pysisyphus.irc.IRCDummy.IRCDummy attribute*), 272
- atoms_are_too_close()* (*pysisyphus.stochastic.Pipeline.Pipeline method*), 303
- atoms_coords_to_crd_str()* (in module *pysisyphus.io.crd*), 264
- atoms_coords_to_pdb_str()* (in module *pysisyphus.io.pdb*), 265
- augment_bonds()* (in module *pysisyphus.intcoords.augment_bonds*), 248
- augment_primitives()* (in module *pysisyphus.intcoords.eval*), 249
- automatic_fragmentation()* (in module *pysisyphus.drivers.afir*), 222
- AUX_BOND* (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- AUX_INTERFRAG_BOND* (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- aux_interfrag_bonds* (*pysisyphus.intcoords.setup.CoordInfo attribute*), 253
- available()* (in module *pysisyphus.testing*), 332
- available_potentials* (*pysisyphus.calculators.ExternalPotential attribute*), 200
- available_potentials* (*pysisyphus.calculators.ExternalPotential.ExternalPotential attribute*), 168
- ## B
- B* (*pysisyphus.intcoords.DLC property*), 258
- B* (*pysisyphus.intcoords.DLC.DLC property*), 240
- B* (*pysisyphus.intcoords.RedundantCoords property*), 260
- B* (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 43, 244
- B_inv* (*pysisyphus.intcoords.RedundantCoords property*), 260
- B_inv* (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 43, 244
- B_inv_prim* (*pysisyphus.intcoords.RedundantCoords property*), 260
- B_inv_prim* (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 43, 244
- B_prim* (*pysisyphus.intcoords.RedundantCoords property*), 260
- B_prim* (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 43, 244

- `backtrack()` (`pysisyphus.optimizers.BacktrackingOptimizer` method), 284
- `Backtracking` (class in `pysisyphus.line_searches`), 280
- `Backtracking` (class in `pysisyphus.line_searches.Backtracking`), 278
- `BacktrackingOptimizer` (class in `pysisyphus.optimizers.BacktrackingOptimizer`), 284
- `backtransform_hessian()` (`pysisyphus.intcoords.DLC` method), 258
- `backtransform_hessian()` (`pysisyphus.intcoords.DLC.DLC` method), 240
- `backtransform_hessian()` (`pysisyphus.intcoords.RedundantCoords` method), 260
- `backtransform_hessian()` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` method), 43, 245
- `backward` (`pysisyphus.irc.IRCDummy.IRCDummy` attribute), 272
- `barrier` (`pysisyphus.drivers.rates.ReactionRates` attribute), 227
- `barrier_si` (`pysisyphus.drivers.rates.ReactionRates` attribute), 227
- `base_cmd` (`pysisyphus.calculators.MOPAC` attribute), 203
- `base_cmd` (`pysisyphus.calculators.MOPAC.MOPAC` attribute), 64, 174
- `bell_corr()` (in module `pysisyphus.drivers.rates`), 227
- `Benchmark` (class in `pysisyphus.benchmarks.Benchmark`), 151
- `Bend` (class in `pysisyphus.intcoords`), 257
- `Bend` (class in `pysisyphus.intcoords.Bend`), 237
- `BEND` (`pysisyphus.intcoords.PrimTypes.PrimTypes` attribute), 242
- `Bend2` (class in `pysisyphus.intcoords`), 257
- `Bend2` (class in `pysisyphus.intcoords.Bend2`), 237
- `BEND2` (`pysisyphus.intcoords.PrimTypes.PrimTypes` attribute), 242
- `bend_atom_indices` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `bend_atom_indices` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `bend_indices` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `bend_indices` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `bend_valid()` (in module `pysisyphus.intcoords.valid`), 257
- `bends` (`pysisyphus.intcoords.setup.CoordInfo` attribute), 253
- `bendsearch()` (in module `pysisyphus.dynamics.thermostats`), 237
- `BFGS` (class in `pysisyphus.optimizers.BFGS`), 284
- `bfgs_multiply()` (in module `pysisyphus.optimizers.closures`), 296
- `bfgs_update()` (in module `pysisyphus.optimizers.hessian_updates`), 299
- `bfgs_update()` (`pysisyphus.optimizers.BFGS.BFGS` method), 284
- `bio_mode()` (`pysisyphus.optimizers.StabilizedQNMMethod.StabilizedQNMMethod` method), 295
- `birkholz_interpolation()` (in module `pysisyphus.drivers.birkholz`), 224
- `bisect()` (`pysisyphus.line_searches.HagerZhang` method), 281
- `bisect()` (`pysisyphus.line_searches.HagerZhang.HagerZhang` method), 278
- `block_davidson()` (in module `pysisyphus.modelfollow.davidson`), 282
- `bofill_update()` (in module `pysisyphus.optimizers.hessian_updates`), 299
- `BOND` (`pysisyphus.intcoords.PrimTypes.PrimTypes` attribute), 242
- `bond_atom_indices` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `bond_atom_indices` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `bond_indices` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `bond_indices` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `bond_order()` (in module `pysisyphus.drivers.birkholz`), 224
- `bond_orders()` (in module `pysisyphus.drivers.birkholz`), 224
- `bond_orders_for_geom()` (in module `pysisyphus.drivers.birkholz`), 224
- `bond_sets` (`pysisyphus.Geometry.Geometry` property), 36, 311
- `bond_typed_prims` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `bond_typed_prims` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `BONDED_FRAGMENT` (`pysisyphus.intcoords.PrimTypes.PrimTypes` attribute), 242
- `BondedFragment` (class in `pysisyphus.intcoords.BondedFragment`), 238
- `bonds` (`pysisyphus.intcoords.setup.CoordInfo` attribute),

- 254
- `bool_color()` (in module `pysisyphus.color`), 318
- `bracket()` (`pysisyphus.line_searches.HagerZhang` method), 281
- `bracket()` (`pysisyphus.line_searches.HagerZhang.HagerZhang` method), 278
- `Bt_inv` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `Bt_inv` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `Bt_inv_prim` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `Bt_inv_prim` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `build_gateway_str()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `build_gateway_str()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `build_mcpdft_str()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `build_mcpdft_str()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `build_rasscf_str()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `build_rasscf_str()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `build_rassi_str()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `build_rassi_str()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `build_set_plans()` (`pysisyphus.calculators.Calculator.Calculator` method), 49, 157
- `build_str_from_dict()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `build_str_from_dict()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- ## C
- `C` (`pysisyphus.calculators.Dimer` property), 198
- `C` (`pysisyphus.calculators.Dimer.Dimer` property), 76, 121, 164
- `C` (`pysisyphus.intcoords.RedundantCoords` property), 260
- `C` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 43, 245
- `cache_arrays()` (in module `pysisyphus.helpers_pure`), 320
- `calc()` (`pysisyphus.calculators.DFTD3.DFTD3` method), 71
- `calc()` (`pysisyphus.calculators.ExternalPotential.HarmonicSphere` method), 168
- `calc()` (`pysisyphus.calculators.ExternalPotential.LogFermi` method), 168
- `calc()` (`pysisyphus.calculators.ExternalPotential.Restraint` method), 169
- `calc()` (`pysisyphus.calculators.ExternalPotential.RMSD` method), 169
- `calc_angle()` (`pysisyphus.irc.ParamPlot.ParamPlot` method), 274
- `calc_bond()` (`pysisyphus.irc.ParamPlot.ParamPlot` method), 274
- `calc_centroid()` (`pysisyphus.calculators.CFOUR` method), 54, 182
- `calc_double_ao_overlap()` (`pysisyphus.calculators.CFOUR` method), 70
- `calc_double_ao_overlap()` (`pysisyphus.Geometry.Geometry` method), 37, 311
- `calc_energy_and_forces()` (`pysisyphus.Geometry.Geometry` method), 37, 311
- `calc_getter` (`pysisyphus.run.RunResult` attribute), 329
- `calc_hessian_for()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 218
- `calc_ipi_client()` (in module `pysisyphus.calculators.IPIClient`), 172
- `calc_layer()` (`pysisyphus.calculators.ONIOM` method), 205
- `calc_layer()` (`pysisyphus.calculators.ONIOMv2.ONIOM` method), 75, 177
- `calc_prim_restraint()` (`pysisyphus.calculators.ExternalPotential.Restraint` static method), 169
- `calc_rot_matrix()` (in module `pysisyphus.calculators.CFOUR`), 70
- `CALC_TYPES` (`pysisyphus.calculators.MOPAC` attribute), 203
- `CALC_TYPES` (`pysisyphus.calculators.MOPAC.MOPAC` attribute), 64, 174
- `calced_geoms` (`pysisyphus.run.RunResult` attribute), 329
- `calced_results` (`pysisyphus.run.RunResult` attribute), 329
- `calcs_from_dict()` (in module `pysisyphus.calculators.MultiCalc`), 175
- `calculate()` (`pysisyphus.calculators.LennardJones` method), 203
- `calculate()` (`pysisyphus.calculators.LennardJones.LennardJones` method), 79, 174
- `calculate()` (`pysisyphus.calculators.TIP3P` method), 210
- `calculate()` (`pysisyphus.calculators.TIP3P.TIP3P` method), 79, 186

- `calculate()` (*pysisyphus.dynamics.Gaussian.Gaussian method*), 231
- `calculate()` (*pysisyphus.intcoords.LinearBend method*), 259
- `calculate()` (*pysisyphus.intcoords.LinearBend.LinearBend method*), 241
- `calculate()` (*pysisyphus.intcoords.LinearDisplacement method*), 259
- `calculate()` (*pysisyphus.intcoords.LinearDisplacement.LinearDisplacement method*), 241
- `calculate()` (*pysisyphus.intcoords.Primitive.Primitive method*), 244
- `calculate_forces()` (*pysisyphus.cos.ChainOfStates.ChainOfStates method*), 108, 215
- `Calculator` (class in *pysisyphus.calculators.Calculator*), 48, 156
- `calculator` (*pysisyphus.cos.ChainOfStates.ChainOfStates property*), 108, 215
- `call_jmol()` (in module *pysisyphus.wrapper.jmol*), 308
- `call_mwfn()` (in module *pysisyphus.wrapper.mwfn*), 309
- `call_packmol()` (in module *pysisyphus.wrapper.packmol*), 309
- `can_bias_f0` (*pysisyphus.calculators.Dimer property*), 198
- `can_bias_f0` (*pysisyphus.calculators.Dimer.Dimer property*), 76, 121, 165
- `can_bias_f1` (*pysisyphus.calculators.Dimer property*), 198
- `can_bias_f1` (*pysisyphus.calculators.Dimer.Dimer property*), 76, 121, 165
- `cap_fragment()` (in module *pysisyphus.calculators.ONIOMv2*), 76, 178
- `capped_atoms_coords()` (*pysisyphus.calculators.ONIOMv2.Model method*), 74, 176
- `cart_axis` (*pysisyphus.intcoords.Cartesian.CartesianX attribute*), 238
- `cart_axis` (*pysisyphus.intcoords.Cartesian.CartesianY attribute*), 238
- `cart_axis` (*pysisyphus.intcoords.Cartesian.CartesianZ attribute*), 238
- `cart_axis` (*pysisyphus.intcoords.CartesianX attribute*), 258
- `cart_axis` (*pysisyphus.intcoords.CartesianY attribute*), 258
- `cart_axis` (*pysisyphus.intcoords.CartesianZ attribute*), 258
- `cart_axis` (*pysisyphus.intcoords.Translation.TranslationX attribute*), 247
- `cart_axis` (*pysisyphus.intcoords.Translation.TranslationY attribute*), 247
- `cart_axis` (*pysisyphus.intcoords.Translation.TranslationZ attribute*), 247
- `cart_axis` (*pysisyphus.intcoords.TranslationX attribute*), 262
- `cart_axis` (*pysisyphus.intcoords.TranslationY attribute*), 262
- `cart_axis` (*pysisyphus.intcoords.TranslationZ attribute*), 262
- `cart_coords` (*pysisyphus.cos.ChainOfStates.ChainOfStates property*), 108, 215
- `cart_coords` (*pysisyphus.cos.GrowingNT.GrowingNT property*), 219
- `cart_coords` (*pysisyphus.drivers.afir.AFIRPath attribute*), 222
- `cart_coords` (*pysisyphus.Geometry.Geometry property*), 37, 311
- `cart_coords` (*pysisyphus.irc.Instanton.Instanton property*), 273
- `cart_forces` (*pysisyphus.cos.GrowingNT.GrowingNT property*), 219
- `cart_forces` (*pysisyphus.Geometry.Geometry property*), 37, 311
- `cart_forces` (*pysisyphus.irc.Instanton.Instanton property*), 273
- `cart_gradient` (*pysisyphus.Geometry.Geometry property*), 37, 311
- `cart_hessian` (*pysisyphus.Geometry.Geometry property*), 37, 311
- `cart_hessian` (*pysisyphus.irc.Instanton.Instanton property*), 273
- `Cartesian` (class in *pysisyphus.intcoords.Cartesian*), 238
- `CARTESIAN` (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- `cartesian_indices` (*pysisyphus.intcoords.RedundantCoords property*), 260
- `cartesian_indices` (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 43, 245
- `cartesian_inds` (*pysisyphus.intcoords.setup.CoordInfo attribute*), 254
- `CARTESIAN_X` (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- `CARTESIAN_Y` (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- `CARTESIAN_Z` (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- `CartesianCoords` (class in *pysisyphus.intcoords*), 257
- `CartesianCoords` (class in *pysisyphus.intcoords.CartesianCoords*), 238
- `CartesianX` (class in *pysisyphus.intcoords*), 258
- `CartesianX` (class in *pysisyphus.intcoords.Cartesian*), 238
- `CartesianY` (class in *pysisyphus.intcoords*), 258
- `CartesianY` (class in *pysisyphus.intcoords.Cartesian*), 238

- 238
- CartesianZ (class in *pysisyphus.intcoords*), 258
- CartesianZ (class in *pysisyphus.intcoords.Cartesian*), 238
- CDD_PNG_FNS (in module *pysisyphus.plot*), 328
- center() (in module *pysisyphus.trj*), 332
- center() (*pysisyphus.Geometry.Geometry* method), 37, 311
- center_fragments() (in module *pysisyphus.drivers.precon_pos_rot*), 226
- center_of_mass (*pysisyphus.Geometry.Geometry* property), 37, 311
- center_of_mass_at() (*pysisyphus.Geometry.Geometry* method), 37, 311
- centerm() (in module *pysisyphus.trj*), 332
- centroid (*pysisyphus.Geometry.Geometry* property), 37, 312
- CerjanMiller (class in *pysisyphus.calculators.CerjanMiller*), 162
- CFOUR (class in *pysisyphus.calculators*), 194
- CFOUR (class in *pysisyphus.calculators.CFOUR*), 68
- cg_step() (*pysisyphus.optimizers.MicroOptimizer* method), 301
- cg_step() (*pysisyphus.optimizers.MicroOptimizer.MicroOptimizer* method), 289
- ChainOfStates (class in *pysisyphus.cos.ChainOfStates*), 108, 215
- charge (*pysisyphus.calculators.AFIR* property), 194
- charge (*pysisyphus.calculators.AFIR.AFIR* property), 73, 153
- charge (*pysisyphus.calculators.ONIOM* property), 205
- charge (*pysisyphus.calculators.ONIOMv2.LayerCalc* property), 74, 176
- charge (*pysisyphus.calculators.ONIOMv2.ONIOM* property), 75, 177
- charge (*pysisyphus.db.molecules.Molecule* attribute), 221
- charge (*pysisyphus.drivers.afir.AFIRPath* attribute), 222
- charges (*pysisyphus.calculators.TIP3P* attribute), 210
- charges (*pysisyphus.calculators.TIP3P.TIP3P* attribute), 79, 186
- check_alpha() (*pysisyphus.line_searches.LineSearch.LineSearch* method), 279
- check_asserts() (in module *pysisyphus.run*), 330
- check_convergence() (*pysisyphus.cos.GrowingNT.GrowingNT* method), 219
- check_convergence() (*pysisyphus.optimizers.LayerOpt.LayerOpt* method), 96, 288
- check_convergence() (*pysisyphus.optimizers.Optimizer.Optimizer* method), 86, 292
- check_convergence() (*pysisyphus.optimizers.StringOptimizer.StringOptimizer* method), 296
- check_for_climbing_start() (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 108, 215
- check_for_end_sign() (in module *pysisyphus.helpers*), 319
- check_mem() (in module *pysisyphus.helpers_pure*), 320
- check_primitives() (in module *pysisyphus.intcoords.eval*), 249
- check_termination() (*pysisyphus.calculators.ORCA* method), 206
- check_termination() (*pysisyphus.calculators.ORCA.ORCA* method), 59, 178
- check_termination() (*pysisyphus.calculators.XTB* static method), 213
- check_termination() (*pysisyphus.calculators.XTB.XTB* static method), 66, 191
- check_typed_prims() (in module *pysisyphus.intcoords.valid*), 257
- ci_coeffs() (in module *pysisyphus.helpers_pure*), 320
- ci_coeffs_above_thresh() (*pysisyphus.calculators.WFOWrapper.WFOWrapper* method), 189
- ci_coeffs_above_thresh() (*pysisyphus.calculators.WFOWrapper2.WFOWrapper2* method), 189
- cid_from_name() (in module *pysisyphus.io.pubchem*), 265
- CIQuantities (class in *pysisyphus.calculators.ConicalIntersection*), 162
- clean() (in module *pysisyphus.db.generate_db*), 221
- clean() (*pysisyphus.calculators.Calculator.Calculator* method), 49, 157
- clean_tmp() (*pysisyphus.calculators.ORCA* method), 206
- clean_tmp() (*pysisyphus.calculators.ORCA.ORCA* method), 59, 178
- clear() (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 108, 215
- clear() (*pysisyphus.Geometry.Geometry* method), 37, 312
- clear() (*pysisyphus.intcoords.RedundantCoords* method), 260
- clear() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 245
- clear_passed() (*pysisyphus.cos.GrowingNT.GrowingNT* method), 219
- coeffs (*pysisyphus.optimizers.gdiis.DIISResult* attribute), 297

- Colvar (*class in pysisyphus.dynamics.colvars*), 232
- comment (*pysisyphus.Geometry.Geometry property*), 37, 312
- compare_to_geometric() (*in module pysisyphus.intcoords.Rotation*), 247
- compatible() (*pysisyphus.drivers.afir.AFIRPath method*), 222
- complete_fragments() (*in module pysisyphus.helpers*), 319
- Composite (*class in pysisyphus.calculators*), 196
- Composite (*class in pysisyphus.calculators.Composite*), 162
- compute() (*pysisyphus.calculators.Dalton.Dalton method*), 68, 164
- concurrent_force_calcs() (*pysisyphus.cos.ChainOfStates.ChainOfStates method*), 108, 215
- condition() (*pysisyphus.calculators.Calculator.SetPlan method*), 53, 161
- conf_key (*pysisyphus.calculators.Calculator.Calculator attribute*), 49, 157
- conf_key (*pysisyphus.calculators.CFOUR attribute*), 195
- conf_key (*pysisyphus.calculators.CFOUR.CFOUR attribute*), 69
- conf_key (*pysisyphus.calculators.Dalton.Dalton attribute*), 68, 164
- conf_key (*pysisyphus.calculators.DFTBp attribute*), 197
- conf_key (*pysisyphus.calculators.DFTBp.DFTBp attribute*), 63, 163
- conf_key (*pysisyphus.calculators.DFTD3.DFTD3 attribute*), 71
- conf_key (*pysisyphus.calculators.Gaussian09 attribute*), 201
- conf_key (*pysisyphus.calculators.Gaussian09.Gaussian09 attribute*), 55, 170
- conf_key (*pysisyphus.calculators.Gaussian16 attribute*), 201
- conf_key (*pysisyphus.calculators.Gaussian16.Gaussian16 attribute*), 56, 170
- conf_key (*pysisyphus.calculators.MOPAC attribute*), 204
- conf_key (*pysisyphus.calculators.MOPAC.MOPAC attribute*), 64, 174
- conf_key (*pysisyphus.calculators.OpenMolcas attribute*), 208
- conf_key (*pysisyphus.calculators.OpenMolcas.OpenMolcas attribute*), 58, 181
- conf_key (*pysisyphus.calculators.ORCA attribute*), 206
- conf_key (*pysisyphus.calculators.ORCA.ORCA attribute*), 59, 178
- conf_key (*pysisyphus.calculators.ORCA5 attribute*), 207
- conf_key (*pysisyphus.calculators.ORCA5.ORCA5 attribute*), 180
- conf_key (*pysisyphus.calculators.Psi4 attribute*), 209
- conf_key (*pysisyphus.calculators.Psi4.Psi4 attribute*), 65, 184
- conf_key (*pysisyphus.calculators.Turbomole attribute*), 211
- conf_key (*pysisyphus.calculators.Turbomole.Turbomole attribute*), 61, 187
- conf_key (*pysisyphus.calculators.XTB attribute*), 213
- conf_key (*pysisyphus.calculators.XTB.XTB attribute*), 67, 191
- conf_thresh (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator attribute*), 54, 182
- conf_thresh (*pysisyphus.calculators.WFOWrapper.WFOWrapper property*), 189
- confirm_input() (*in module pysisyphus.helpers*), 319
- ConicalIntersection (*class in pysisyphus.calculators*), 196
- ConicalIntersection (*class in pysisyphus.calculators.ConicalIntersection*), 163
- ConjugateGradient (*class in pysisyphus.optimizers.ConjugateGradient*), 285
- connect_fragments() (*in module pysisyphus.intcoords.setup*), 254
- connect_fragments_ahlrichs() (*in module pysisyphus.intcoords.setup*), 254
- connect_fragments_kmeans() (*in module pysisyphus.intcoords.setup*), 254
- constrained_indices (*pysisyphus.intcoords.RedundantCoords property*), 261
- constrained_indices (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 44, 245
- constraints (*pysisyphus.intcoords.DLC property*), 258
- constraints (*pysisyphus.intcoords.DLC.DLC property*), 240
- control_from_simple_input() (*in module pysisyphus.calculators.Turbomole*), 63, 188
- conv_dict (*pysisyphus.calculators.OBabel attribute*), 204
- conv_dict (*pysisyphus.calculators.OBabel.OBabel attribute*), 68, 175
- converged (*pysisyphus.line_searches.LineSearch.LineSearchResult attribute*), 279
- converged (*pysisyphus.modelfollow.davidson.DavidsonResult attribute*), 282
- ConvInfo (*class in pysisyphus.optimizers.Optimizer*), 84, 290
- coord_types (*pysisyphus.Geometry.Geometry attribute*), 37, 312
- coordinates_similar() (*in module pysisyphus.drivers.afir*), 222
- CoordInfo (*class in pysisyphus.intcoords.setup*), 253
- coords (*pysisyphus.cos.ChainOfStates.ChainOfStates*

- property), 108, 215
- coords (pysisyphus.cos.FreezingString.FreezingString property), 217
- coords (pysisyphus.cos.GrowingNT.GrowingNT property), 219
- coords (pysisyphus.dynamics.driver.MDResult attribute), 232
- coords (pysisyphus.Geometry.Geometry property), 37, 312
- coords (pysisyphus.intcoords.CartesianCoords property), 257
- coords (pysisyphus.intcoords.CartesianCoords.CartesianCoords property), 238
- coords (pysisyphus.intcoords.Coords.CoordSys property), 239
- coords (pysisyphus.intcoords.DLC property), 258
- coords (pysisyphus.intcoords.DLC.DLC property), 240
- coords (pysisyphus.intcoords.RedundantCoords property), 261
- coords (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 245
- coords (pysisyphus.irc.Instanton.Instanton property), 273
- coords (pysisyphus.irc.IRC.IRC property), 125, 270
- coords (pysisyphus.optimizers.gdiis.DIISResult attribute), 297
- coords0 (pysisyphus.calculators.Dimer property), 198
- coords0 (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 165
- coords1 (pysisyphus.calculators.Dimer property), 198
- coords1 (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 165
- coords3d (pysisyphus.cos.ChainOfStates.ChainOfStates property), 109, 215
- coords3d (pysisyphus.Geometry.Geometry property), 38, 312
- coords3d (pysisyphus.intcoords.CartesianCoords property), 257
- coords3d (pysisyphus.intcoords.CartesianCoords.CartesianCoords property), 238
- coords3d (pysisyphus.intcoords.Coords.CoordSys property), 239
- coords3d (pysisyphus.intcoords.RedundantCoords property), 261
- coords3d (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 245
- coords_by_type (pysisyphus.Geometry.Geometry property), 38, 312
- coords_minus (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275
- coords_plus (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275
- coords_to_trj() (in module pysisyphus.xyzloader), 333
- CoordSys (class in pysisyphus.intcoords.Coords), 239
- copy() (pysisyphus.Geometry.Geometry method), 38, 312
- copy_all() (pysisyphus.Geometry.Geometry method), 38, 313
- copy_yaml_and_geometries() (in module pysisyphus.run), 330
- correct_dihedrals() (in module pysisyphus.intcoords.update), 256
- corrector_step() (pysisyphus.irc.EulerPC method), 276
- corrector_step() (pysisyphus.irc.EulerPC.EulerPC method), 127, 268
- cos (pysisyphus.run.RunResult attribute), 329
- cos_opt (pysisyphus.run.RunResult attribute), 329
- cost_function() (pysisyphus.interpolate.LST.LST method), 263
- coulomb() (pysisyphus.calculators.TIP3P method), 210
- coulomb() (pysisyphus.calculators.TIP3P.TIP3P method), 79, 186
- covalent_radii (pysisyphus.Geometry.Geometry property), 38, 313
- CovRadiiSumZero, 73, 154
- create_bond_vec_getters() (pysisyphus.calculators.ONIOMv2.Model method), 74, 176
- create_links() (pysisyphus.calculators.ONIOMv2.Model method), 74, 176
- cross3() (in module pysisyphus.linalg), 323
- csvr_closure() (in module pysisyphus.dynamics.thermostats), 237
- csvr_closure_2() (in module pysisyphus.dynamics.thermostats), 237
- cubic_displ() (in module pysisyphus.irc.initial_displ), 275
- cubic_displ_for_geom() (in module pysisyphus.irc.initial_displ), 275
- cubic_displ_for_h5() (in module pysisyphus.irc.initial_displ), 275
- cubic_fit() (in module pysisyphus.optimizers.poly_fit), 300
- CubicNewton (class in pysisyphus.optimizers), 301
- CubicNewton (class in pysisyphus.optimizers.CubicNewton), 285
- cur_cycle (pysisyphus.modelfollow.davidson.DavidsonResult attribute), 282
- cur_cycle (pysisyphus.optimizers.Optimizer.ConvInfo attribute), 84, 290
- curvature() (pysisyphus.calculators.Dimer method), 198
- curvature() (pysisyphus.calculators.Dimer.Dimer method), 76, 121, 165

- curvature_condition() (pysisyphus.line_searches.LineSearch.LineSearch method), 279
- CVBend (class in pysisyphus.dynamics.colvars), 232
- CVDistance (class in pysisyphus.dynamics.colvars), 232
- CVTorsion (class in pysisyphus.dynamics.colvars), 232
- ## D
- d (pysisyphus.io.zmat.ZLine attribute), 266
- d0 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 250
- d1 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 250
- d2 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 250
- d2q_a() (in module pysisyphus.intcoords.derivatives), 248
- d2q_a() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_a2() (in module pysisyphus.intcoords.derivatives), 248
- d2q_a2() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_b() (in module pysisyphus.intcoords.derivatives), 248
- d2q_b() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_d() (in module pysisyphus.intcoords.derivatives), 248
- d2q_d() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_d2() (in module pysisyphus.intcoords.derivatives), 248
- d2q_d2() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_lb() (in module pysisyphus.intcoords.derivatives), 248
- d2q_lb() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_ld() (in module pysisyphus.intcoords.derivatives), 248
- d2q_ld() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_oop() (in module pysisyphus.intcoords.derivatives), 248
- d2q_oop() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_rd1() (in module pysisyphus.intcoords.derivatives), 248
- d2q_rd1() (in module pysisyphus.intcoords.mp_derivatives), 252
- d2q_rd2() (in module pysisyphus.intcoords.derivatives), 248
- d2q_rd2() (in module pysisyphus.intcoords.mp_derivatives), 252
- Dalton (class in pysisyphus.calculators.Dalton), 68, 164
- damp_velocity() (pysisyphus.irc.DampedVelocityVerlet method), 276
- damp_velocity() (pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet method), 126, 268
- damped_bfgs_update() (in module pysisyphus.optimizers.hessian_updates), 299
- damped_bfgs_update() (pysisyphus.optimizers.BFGS.BFGS method), 284
- DampedVelocityVerlet (class in pysisyphus.irc), 276
- DampedVelocityVerlet (class in pysisyphus.irc.DampedVelocityVerlet), 126, 268
- data_funcs (pysisyphus.benchmarks.Benchmark.Benchmark attribute), 151
- data_model (pysisyphus.calculators.OverlapCalculator.OverlapCalculator property), 54, 182
- DavidsonResult (class in pysisyphus.modelfollow.davidson), 282
- decrease_distance() (in module pysisyphus.drivers.afir), 222
- del_atoms() (pysisyphus.Geometry.Geometry method), 38, 313
- density (pysisyphus.db.molecules.Molecule attribute), 221
- describe() (in module pysisyphus.helpers_pure), 320
- describe() (pysisyphus.cos.ChainOfStates.ChainOfStates method), 109, 215
- describe() (pysisyphus.Geometry.Geometry method), 38, 313
- desired_eigval_structure (pysisyphus.optimizers.Optimizer.ConvInfo attribute), 84, 290
- detect_paths() (in module pysisyphus.config), 318
- determine_target_pairs() (in module pysisyphus.drivers.afir), 222
- determine_target_pairs_for_geom() (in module pysisyphus.drivers.afir), 223
- df_evals (pysisyphus.line_searches.LineSearch.LineSearchResult attribute), 280
- DFTBp (class in pysisyphus.calculators), 196
- DFTBp (class in pysisyphus.calculators.DFTBp), 63, 163
- DFTD3 (class in pysisyphus.calculators.DFTD3), 71
- DFTD4 (class in pysisyphus.calculators), 197
- dict_to_mol2_string() (in module pysisyphus.io.mol2), 265
- DifferentCoordLengthsException, 250
- DifferentPrimitivesException, 250
- dihedral_atom_indices (pysisyphus.intcoords.RedundantCoords property), 261

dihedral_atom_indices (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 245
 dihedral_indices (pysisyphus.intcoords.RedundantCoords property), 261
 dihedral_indices (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 245
 dihedral_valid() (in module pysisyphus.intcoords.valid), 257
 dihedrals_are_valid() (in module pysisyphus.intcoords.valid), 257
 diis_result() (in module pysisyphus.optimizers.gdiis), 297
 DIISResult (class in pysisyphus.optimizers.gdiis), 297
 Dimer (class in pysisyphus.calculators), 197
 Dimer (class in pysisyphus.calculators.Dimer), 76, 121, 164
 dind (pysisyphus.io.zmat.ZLine attribute), 266
 direct_cycle() (in module pysisyphus.drivers.pka), 226
 direct_rotation() (pysisyphus.calculators.Dimer method), 198
 direct_rotation() (pysisyphus.calculators.Dimer.Dimer method), 76, 121, 165
 DISTANCE_FUNCTION (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 242
 DistanceFunction (class in pysisyphus.intcoords), 259
 DistanceFunction (class in pysisyphus.intcoords.DistanceFunction), 241
 DLC (class in pysisyphus.intcoords), 258
 DLC (class in pysisyphus.intcoords.DLC), 240
 do_calculations() (pysisyphus.calculators.EnergyMin method), 200
 do_calculations() (pysisyphus.calculators.EnergyMin.EnergyMin method), 167
 do_clean() (in module pysisyphus.run), 330
 do_dimer_rotations() (pysisyphus.calculators.Dimer method), 198
 do_dimer_rotations() (pysisyphus.calculators.Dimer.Dimer method), 76, 121, 165
 do_endopt_ts_barriers() (in module pysisyphus.drivers.barriers), 224
 do_final_hessian() (in module pysisyphus.helpers), 319
 do_line_search() (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer static method), 119, 307
 do_parent() (pysisyphus.calculators.ONIOMv2.LayerCalculator method), 74, 176
 do_rmsds() (in module pysisyphus.run), 330
 double_damp() (in module pysisyphus.optimizers.hessian_updates), 299
 double_damped_bfgs_update() (pysisyphus.optimizers.BFGS.BFGS method), 284
 double_secant() (pysisyphus.line_searches.HagerZhang method), 281
 double_secant() (pysisyphus.line_searches.HagerZhang.HagerZhang method), 278
 downhill (pysisyphus.irc.IRCDummy.IRCDummy attribute), 272
 dphi0 (pysisyphus.line_searches.LineSearch.LineSearchResult attribute), 280
 dq_a() (in module pysisyphus.intcoords.derivatives), 248
 dq_a() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_a2() (in module pysisyphus.intcoords.derivatives), 248
 dq_a2() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_b() (in module pysisyphus.intcoords.derivatives), 248
 dq_b() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_d() (in module pysisyphus.intcoords.derivatives), 248
 dq_d() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_d2() (in module pysisyphus.intcoords.derivatives), 248
 dq_d2() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_lb() (in module pysisyphus.intcoords.derivatives), 249
 dq_lb() (in module pysisyphus.intcoords.mp_derivatives), 252
 dq_ld() (in module pysisyphus.intcoords.derivatives), 249
 dq_ld() (in module pysisyphus.intcoords.mp_derivatives), 253
 dq_oop() (in module pysisyphus.intcoords.derivatives), 249
 dq_oop() (in module pysisyphus.intcoords.mp_derivatives), 253
 dq_rd1() (in module pysisyphus.intcoords.derivatives), 249
 dq_rd1() (in module pysisyphus.intcoords.mp_derivatives), 253
 dq_rd2() (in module pysisyphus.intcoords.derivatives), 249
 dq_rd2() (in module pysisyphus.intcoords.mp_derivatives), 253
 dq_rd2() (in module pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 249

- tribute*), 275
- Dummy (class in *pysisyphus.calculators*), 199
- Dummy (class in *pysisyphus.calculators.Dummy*), 166
- dummy_hessian_update() (in module *pysisyphus.optimizers.HessianOptimizer*), 89, 287
- DUMMY_IMPROPER (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 242
- DUMMY_TORSION (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 242
- DummyColvar (class in *pysisyphus.dynamics.Gaussian*), 231
- DummyImproper (class in *pysisyphus.intcoords*), 259
- DummyMark (class in *pysisyphus.testing*), 332
- DummyTorsion (class in *pysisyphus.intcoords*), 259
- DummyTorsion (class in *pysisyphus.intcoords.DummyTorsion*), 241
- dump() (pysisyphus.dynamics.Gaussian.Gaussian method), 231
- dump() (pysisyphus.irc.DWI.DWI method), 267
- dump_coords() (in module *pysisyphus.dynamics.helpers*), 233
- dump_data() (pysisyphus.irc.IRC.IRC method), 125, 270
- dump_ends() (pysisyphus.irc.IRC.IRC method), 125, 270
- dump_geoms() (in module *pysisyphus.trj*), 332
- dump_h5() (pysisyphus.calculators.AFIR method), 194
- dump_h5() (pysisyphus.calculators.AFIR.AFIR method), 73, 153
- dump_overlap_data() (pysisyphus.calculators.OverlapCalculator.OverlapCalculator method), 54, 182
- dump_progress() (pysisyphus.interpolate.Redund.Redund method), 263
- dump_restart_info() (pysisyphus.optimizers.Optimizer.Optimizer method), 86, 292
- dump_trj() (pysisyphus.drivers.afir.AFIRPath method), 222
- dump_xyz() (pysisyphus.Geometry.Geometry method), 38, 313
- DWI (class in *pysisyphus.irc.DWI*), 267
- E**
- E_tot (pysisyphus.dynamics.driver.MDResult attribute), 232
- eckart_corr() (in module *pysisyphus.drivers.rates*), 228
- eckart_corr_brown() (in module *pysisyphus.drivers.rates*), 228
- eckart_projection() (pysisyphus.Geometry.Geometry method), 38, 313
- EGO (class in *pysisyphus.calculators*), 199
- EGO (class in *pysisyphus.calculators.EGO*), 166
- eigval_to_wavenumber() (in module *pysisyphus.helpers_pure*), 320
- eigvals (pysisyphus.helpers.FinalHessianResult attribute), 318
- eigvec_grad() (in module *pysisyphus.linalg*), 323
- embeddings (pysisyphus.calculators.ONIOM attribute), 205
- embeddings (pysisyphus.calculators.ONIOMv2.ONIOM attribute), 75, 177
- end_geoms (pysisyphus.run.RunResult attribute), 330
- energies (pysisyphus.drivers.afir.AFIRPath attribute), 222
- energy (pysisyphus.calculators.ConicalIntersection.CIQuantities attribute), 162
- energy (pysisyphus.cos.ChainOfStates.ChainOfStates property), 109, 215
- energy (pysisyphus.cos.FreezingString.FreezingString property), 217
- energy (pysisyphus.cos.GrowingNT.GrowingNT property), 219
- energy (pysisyphus.Geometry.Geometry property), 38, 313
- energy (pysisyphus.irc.Instanton.Instanton property), 273
- energy (pysisyphus.irc.IRC.IRC property), 125, 270
- energy (pysisyphus.optimizers.gdiis.DIISResult attribute), 297
- energy() (pysisyphus.calculators.Dimer.Gaussian method), 77, 122, 166
- energy0 (pysisyphus.calculators.Dimer property), 198
- energy0 (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 165
- energy1 (pysisyphus.calculators.ConicalIntersection.CIQuantities attribute), 162
- energy2 (pysisyphus.calculators.ConicalIntersection.CIQuantities attribute), 162
- energy_converged (pysisyphus.optimizers.Optimizer.ConvInfo attribute), 84, 290
- energy_diff (pysisyphus.calculators.ConicalIntersection.CIQuantities attribute), 162
- energy_forces_getter_closure() (in module *pysisyphus.dynamics.helpers*), 233
- energy_from_results() (pysisyphus.calculators.ONIOMv2.LayerCalc static method), 74, 176
- energy_minus (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275
- energy_plus (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275
- EnergyMin (class in *pysisyphus.calculators*), 199

- EnergyMin (class in pysisyphus.calculators.EnergyMin), 167
- epsilon (pysisyphus.calculators.TIP3P attribute), 210
- epsilon (pysisyphus.calculators.TIP3P.TIP3P attribute), 79, 186
- estimate() (in module pysisyphus.helpers_pure), 320
- estimate_error() (pysisyphus.irc.DampedVelocityVerlet method), 276
- estimate_error() (pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet method), 126, 268
- Euler (class in pysisyphus.irc), 276
- Euler (class in pysisyphus.irc.Euler), 127, 268
- EulerPC (class in pysisyphus.irc), 276
- EulerPC (class in pysisyphus.irc.EulerPC), 127, 268
- eval() (pysisyphus.dynamics.colvars.Colvar method), 232
- eval() (pysisyphus.dynamics.Gaussian.DummyColvar method), 231
- eval() (pysisyphus.dynamics.Gaussian.Gaussian method), 231
- eval() (pysisyphus.intcoords.RedundantCoords method), 261
- eval() (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 245
- eval_B() (in module pysisyphus.intcoords.eval), 249
- eval_primitives() (in module pysisyphus.intcoords.eval), 250
- every() (in module pysisyphus.trj), 332
- expand() (in module pysisyphus.helpers_pure), 320
- ExternalPotential (class in pysisyphus.calculators), 200
- ExternalPotential (class in pysisyphus.calculators.ExternalPotential), 168
- eye (pysisyphus.optimizers.BFGS.BFGS property), 284
- eyring_rate() (in module pysisyphus.drivers.rates), 228
- F**
- f0 (pysisyphus.calculators.Dimer property), 198
- f0 (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 165
- f0 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 250
- f1 (pysisyphus.calculators.Dimer property), 198
- f1 (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 165
- f1 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 250
- f1_bias (pysisyphus.calculators.Dimer property), 198
- f1_bias (pysisyphus.calculators.Dimer.Dimer property), 77, 121, 165
- f2 (pysisyphus.calculators.Dimer property), 198
- f2 (pysisyphus.calculators.Dimer.Dimer property), 77, 121, 165
- f2 (pysisyphus.intcoords.generate_derivatives.FuncResult attribute), 251
- f20() (in module pysisyphus.io.crd), 264
- f_evals (pysisyphus.line_searches.LineSearch.LineSearchResult attribute), 280
- f_new (pysisyphus.line_searches.LineSearch.LineSearchResult attribute), 280
- fail (pysisyphus.calculators.Calculator.SetPlan attribute), 53, 161
- fake_turbo_mos() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189
- fake_turbo_mos() (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 static method), 189
- FakeASE (class in pysisyphus.calculators), 201
- FakeASE (class in pysisyphus.calculators.FakeASE), 79, 169
- fd_coords3d_gen() (pysisyphus.Geometry.Geometry method), 39, 313
- file_or_str() (in module pysisyphus.helpers_pure), 320
- filter_fixture_store() (in module pysisyphus.helpers_pure), 320
- filter_small_eigvals() (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer method), 88, 286
- fin_diff_B() (in module pysisyphus.intcoords.findiffs), 250
- fin_diff_prim() (in module pysisyphus.intcoords.findiffs), 250
- final_modes (pysisyphus.modelfollow.davidson.DavidsonResult attribute), 282
- final_summary() (pysisyphus.optimizers.Optimizer.Optimizer method), 86, 292
- FinalHessianResult (class in pysisyphus.helpers), 318
- find_bends() (in module pysisyphus.intcoords.setup_fast), 256
- find_bonds() (in module pysisyphus.intcoords.setup_fast), 256
- find_bonds_bends() (in module pysisyphus.intcoords.setup_fast), 256
- find_bonds_for_geom() (in module pysisyphus.intcoords.setup_fast), 256
- find_candidates() (in module pysisyphus.drivers.afir), 223
- find_closest_sequence() (in module pysisyphus.helpers_pure), 320
- find_dihedrals() (in module pysisyphus.intcoords.setup_fast), 256
- find_missing_bonds_by_projection() (in module

- pysisyphus.intcoords.augment_bonds*), 248
- find_missing_strong_bonds()* (in module *pysisyphus.intcoords.augment_bonds*), 248
- finite_difference_hessian()* (in module *pysisyphus.linalg*), 323
- FIRE* (class in *pysisyphus.optimizers.FIRE*), 285
- fischer_guess()* (in module *pysisyphus.optimizers.guess_hessians*), 298
- fit_grad_and_energies()* (*pysisyphus.irc.IMKMod* method), 277
- fit_grad_and_energies()* (*pysisyphus.irc.IMKMod.IMKMod* method), 128, 269
- fit_parabola()* (*pysisyphus.irc.IMKMod* method), 277
- fit_parabola()* (*pysisyphus.irc.IMKMod.IMKMod* method), 128, 269
- fit_rigid()* (in module *pysisyphus.helpers*), 319
- fit_rigid()* (*pysisyphus.optimizers.Optimizer.Optimizer* method), 86, 292
- FitResult* (class in *pysisyphus.optimizers.poly_fit*), 300
- flowchart_update()* (in module *pysisyphus.optimizers.hessian_updates*), 299
- fmt_k()* (*pysisyphus.cos.NEB.NEB* method), 110, 220
- fn* (*pysisyphus.db.molecules.Molecule* attribute), 221
- fn* (*pysisyphus.drivers.opt.OptResult* attribute), 225
- force_num_hessian()* (*pysisyphus.calculators.Calculator.Calculator* method), 49, 157
- forces* (*pysisyphus.calculators.ConicalIntersection.CIQuantities* attribute), 162
- forces* (*pysisyphus.cos.AdaptiveNEB.AdaptiveNEB* property), 111, 215
- forces* (*pysisyphus.cos.ChainOfStates.ChainOfStates* property), 109, 216
- forces* (*pysisyphus.cos.FreezingString.FreezingString* property), 218
- forces* (*pysisyphus.cos.GrowingNT.GrowingNT* property), 219
- forces* (*pysisyphus.cos.GrowingString.GrowingString* property), 113, 219
- forces* (*pysisyphus.cos.NEB.NEB* property), 110, 220
- forces* (*pysisyphus.drivers.afir.AFIRPath* attribute), 222
- forces* (*pysisyphus.Geometry.Geometry* property), 39, 313
- forces* (*pysisyphus.irc.Instanton.Instanton* property), 273
- forces* (*pysisyphus.optimizers.gdiis.DIISResult* attribute), 297
- forces()* (*pysisyphus.calculators.Dimer.Gaussian* method), 77, 122, 166
- forces_fin_diff()* (in module *pysisyphus.modelfollow.davidson*), 283
- form_A()* (in module *pysisyphus.drivers.precon_pos_rot*), 226
- form_coordinate_union()* (in module *pysisyphus.intcoords.helpers*), 251
- forward* (*pysisyphus.irc.IRCDummy.IRCDummy* attribute), 272
- fourier_rotation()* (*pysisyphus.calculators.Dimer* method), 198
- fourier_rotation()* (*pysisyphus.calculators.Dimer.Dimer* method), 77, 121, 165
- FourWellAnaPot* (class in *pysisyphus.calculators.FourWellAnaPot*), 169
- frag_sort()* (in module *pysisyphus.trj*), 332
- FragmentKick* (class in *pysisyphus.stochastic*), 304
- FragmentKick* (class in *pysisyphus.stochastic.FragmentKick*), 302
- fragments* (*pysisyphus.intcoords.setup.CoordInfo* attribute), 254
- FreeEndNEB* (class in *pysisyphus.cos.FreeEndNEB*), 112, 217
- FreeEndNEBPot* (class in *pysisyphus.calculators.FreeEndNEBPot*), 170
- freeze_primitives()* (*pysisyphus.intcoords.DLC* method), 258
- freeze_primitives()* (*pysisyphus.intcoords.DLC.DLC* method), 240
- FreezingString* (class in *pysisyphus.cos.FreezingString*), 217
- from_* (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
- from_coeffs()* (in module *pysisyphus.optimizers.gdiis*), 297
- from_h5()* (*pysisyphus.irc.DWI.DWI* static method), 267
- from_instanton()* (*pysisyphus.irc.Instanton.Instanton* class method), 273
- from_oniom_calculator()* (*pysisyphus.optimizers.LayerOpt.Layers* class method), 96, 289
- from_overlap_data()* (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* static method), 54, 182
- from_ts()* (*pysisyphus.irc.Instanton.Instanton* class method), 273
- full_expand()* (in module *pysisyphus.helpers_pure*), 320
- full_name()* (in module *pysisyphus.db.helpers*), 221
- full_string_image_inds* (*pysisyphus.cos.GrowingString.GrowingString* property), 113, 219
- fully_grown* (*pysisyphus.cos.FreezingString.FreezingString* property), 218
- fully_grown* (*pysisyphus.cos.GrowingString.GrowingString* property), 113, 219
- func()* (*pysisyphus.plotters.AnimPlot.AnimPlot* method), 302

FuncResult (class in pysisyphus.intcoords.generate_derivatives), 250

G

g (pysisyphus.calculators.ONIOMv2.Link attribute), 74, 176

G() (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173

G_aq_from_h5_hessian() (in module pysisyphus.drivers.pka), 226

g_new (pysisyphus.line_searches.LineSearch.LineSearchResult attribute), 280

G_vec (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275

gamma (pysisyphus.drivers.afir.AFIRPath attribute), 222

Gaussian (class in pysisyphus.calculators.Dimer), 77, 122, 166

Gaussian (class in pysisyphus.dynamics.Gaussian), 231

Gaussian09 (class in pysisyphus.calculators), 201

Gaussian09 (class in pysisyphus.calculators.Gaussian09), 55, 170

Gaussian16 (class in pysisyphus.calculators), 201

Gaussian16 (class in pysisyphus.calculators.Gaussian16), 56, 170

gdiis() (in module pysisyphus.optimizers.gdiis), 298

Gdimer() (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173

gediis() (in module pysisyphus.optimizers.gdiis), 298

gen_solutions() (in module pysisyphus.optimizers.poly_fit), 300

generate_all_dets() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189

generate_all_dets() (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 189

generate_db() (in module pysisyphus.db.generate_db), 221

generate_random_union() (in module pysisyphus.drivers.afir), 223

generate_random_union_ref() (in module pysisyphus.drivers.afir), 223

generate_wilson() (in module pysisyphus.intcoords.generate_derivatives), 251

geom (pysisyphus.drivers.opt.OptResult attribute), 225

geom_davidson() (in module pysisyphus.modefollow), 283

geom_davidson() (in module pysisyphus.modefollow.davidson), 283

geom_from_cjson() (in module pysisyphus.io), 267

geom_from_cjson() (in module pysisyphus.io.cjson), 264

geom_from_crd() (in module pysisyphus.io), 267

geom_from_crd() (in module pysisyphus.io.crd), 264

geom_from_cube() (in module pysisyphus.io), 267

geom_from_fchk() (in module pysisyphus.io), 267

geom_from_fn() (pysisyphus.calculators.Calculator.Calculator class method), 49, 157

geom_from_hessian() (in module pysisyphus.io), 267

geom_from_hessian() (in module pysisyphus.io.hessian), 265

geom_from_mol2() (in module pysisyphus.io), 267

geom_from_mol2() (in module pysisyphus.io.mol2), 265

geom_from_pdb() (in module pysisyphus.io), 267

geom_from_pdb() (in module pysisyphus.io.pdb), 265

geom_from_pubchem_name() (in module pysisyphus.io), 267

geom_from_pubchem_name() (in module pysisyphus.io.pubchem), 265

geom_from_qcschema() (in module pysisyphus.io), 267

geom_from_sdf() (in module pysisyphus.io.sdf), 266

geom_from_xyz() (in module pysisyphus.io.xyz), 266

geom_from_xyz_file() (in module pysisyphus.helpers), 319

geom_from_zmat() (in module pysisyphus.io), 267

geom_from_zmat() (in module pysisyphus.io.zmat), 266

geom_from_zmat_fn() (in module pysisyphus.io.zmat), 266

geom_from_zmat_str() (in module pysisyphus.io.zmat), 266

geom_is_close_in_energy() (pysisyphus.stochastic.Pipeline.Pipeline method), 303

geom_is_new() (pysisyphus.stochastic.Pipeline.Pipeline method), 303

geom_is_valid() (pysisyphus.stochastic.Pipeline.Pipeline method), 303

geom_iter (pysisyphus.benchmarks.Benchmark.Benchmark property), 151

geom_lanczos() (in module pysisyphus.modefollow), 283

geom_lanczos() (in module pysisyphus.modefollow.lanczos), 283

geom_loader() (in module pysisyphus.helpers), 319

geom_similar() (in module pysisyphus.drivers.afir), 223

geom_to_crd_str() (in module pysisyphus.io.crd), 264

geom_to_pdb_str() (in module pysisyphus.io.pdb), 265

Geometry (class in pysisyphus.Geometry), 35, 309

geoms (pysisyphus.benchmarks.Benchmark.Benchmark property), 151

geoms_from_inline_xyz() (in module pysisyphus.io.xyz), 266

geoms_from_molden() (in module pysisyphus.io), 267

geoms_from_molden() (in module pysisyphus.io.molden), 265

- [geoms_from_trj\(\)](#) (in module `pysisyphus.helpers`), 319
[geoms_from_xyz\(\)](#) (in module `pysisyphus.io`), 267
[geoms_from_xyz\(\)](#) (in module `pysisyphus.io.xyz`), 266
[get\(\)](#) (in module `pysisyphus.trj`), 332
[get_active_set\(\)](#) (`pysisyphus.intcoords.DLC` method), 258
[get_active_set\(\)](#) (`pysisyphus.intcoords.DLC.DLC` method), 240
[get_additional_print\(\)](#) (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
[get_additional_print\(\)](#) (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 219
[get_additional_print\(\)](#) (`pysisyphus.irc.Instanton.Instanton` method), 273
[get_additional_print\(\)](#) (`pysisyphus.irc.ModeKill.ModeKill` method), 277
[get_additional_print\(\)](#) (`pysisyphus.irc.ModeKill.ModeKill` method), 274
[get_alpha_step\(\)](#) (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 286
[get_atoms_coords\(\)](#) (`pysisyphus.calculators.FakeASE` method), 201
[get_atoms_coords\(\)](#) (`pysisyphus.calculators.FakeASE.FakeASE` method), 79, 169
[get_augmented_hessian\(\)](#) (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 286
[get_baker_data\(\)](#) (in module `pysisyphus.benchmarks.data`), 151
[get_baker_ts_data\(\)](#) (in module `pysisyphus.benchmarks.data`), 151
[get_bend_candidates\(\)](#) (in module `pysisyphus.intcoords.setup_fast`), 256
[get_bend_inds\(\)](#) (in module `pysisyphus.intcoords.setup`), 255
[get_beta\(\)](#) (`pysisyphus.optimizers.ConjugateGradient.ConjugateGradient` method), 54, 182
[get_birkholz_rx_data\(\)](#) (in module `pysisyphus.benchmarks.data`), 151
[get_block_str\(\)](#) (`pysisyphus.calculators.ORCA` method), 206
[get_block_str\(\)](#) (`pysisyphus.calculators.ORCA.ORCA` method), 59, 178
[get_bond_difference\(\)](#) (in module `pysisyphus.intcoords.helpers`), 251
[get_bond_differences_verbose\(\)](#) (in module `pysisyphus.intcoords.helpers`), 251
[get_bond_mat\(\)](#) (in module `pysisyphus.intcoords.setup`), 255
[get_bond_sets\(\)](#) (in module `pysisyphus.intcoords.setup`), 255
[get_bond_subgeom\(\)](#) (in module `pysisyphus.drivers.replace`), 230
[get_bond_vec_getter\(\)](#) (in module `pysisyphus.intcoords.setup_fast`), 256
[get_bonded_frag_coord\(\)](#) (in module `pysisyphus.intcoords.PrimTypes`), 243
[get_box\(\)](#) (in module `pysisyphus.helpers_pure`), 320
[get_calc_closure\(\)](#) (in module `pysisyphus.run`), 330
[get_calculator\(\)](#) (`pysisyphus.calculators.PyXTB` method), 209
[get_calculator\(\)](#) (`pysisyphus.calculators.PyXTB.PyXTB` method), 184
[get_charges\(\)](#) (`pysisyphus.calculators.OpenMM.OpenMM` method), 180
[get_chkfiles\(\)](#) (`pysisyphus.calculators.EnergyMin` method), 200
[get_chkfiles\(\)](#) (`pysisyphus.calculators.EnergyMin.EnergyMin` method), 167
[get_chkfiles\(\)](#) (`pysisyphus.calculators.Gaussian16` method), 201
[get_chkfiles\(\)](#) (`pysisyphus.calculators.Gaussian16.Gaussian16` method), 56, 170
[get_chkfiles\(\)](#) (`pysisyphus.calculators.ORCA` method), 206
[get_chkfiles\(\)](#) (`pysisyphus.calculators.ORCA.ORCA` method), 59, 178
[get_chkfiles\(\)](#) (`pysisyphus.calculators.Turbomole` method), 211
[get_chkfiles\(\)](#) (`pysisyphus.calculators.Turbomole.Turbomole` method), 61, 187
[get_ci_coeffs_for\(\)](#) (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` method), 196
[get_ci_quantities\(\)](#) (`pysisyphus.calculators.ConicalIntersection` method), 163
[get_ci_quantities\(\)](#) (`pysisyphus.calculators.ConicalIntersection.ConicalIntersection` method), 163
[get_climbing_forces\(\)](#) (`pysisyphus.cos.ChainOfStates.ChainOfStates` method), 109, 216
[get_climbing_indices\(\)](#) (`pysisyphus.cos.ChainOfStates.ChainOfStates` method), 109, 216
[get_clock\(\)](#) (in module `pysisyphus.helpers_pure`), 320
[get_cmd\(\)](#) (in module `pysisyphus.config`), 318

- `get_cmd()` (*pysisyphus.calculators.Calculator.Calculator method*), 49, 157
- `get_colvar()` (in module *pysisyphus.dynamics.colvars*), 232
- `get_conect_lines()` (in module *pysisyphus.io.pdb*), 265
- `get_constrained_U()` (*pysisyphus.intcoords.DLC method*), 258
- `get_constrained_U()` (*pysisyphus.intcoords.DLC.DLC method*), 240
- `get_constructor()` (in module *pysisyphus.yaml_mods*), 334
- `get_conv_fact()` (*pysisyphus.irc.IRC.IRC method*), 125, 270
- `get_convergence()` (*pysisyphus.optimizers.Optimizer.ConvInfo method*), 84, 290
- `get_coords3d_and_indices_ext()` (*pysisyphus.intcoords.DummyImproper static method*), 259
- `get_coords3d_and_indices_ext()` (*pysisyphus.intcoords.DummyTorsion static method*), 259
- `get_coords3d_and_indices_ext()` (*pysisyphus.intcoords.DummyTorsion.DummyTorsion static method*), 241
- `get_coords_diffs()` (in module *pysisyphus.helpers*), 319
- `get_cosmo_data_groups()` (in module *pysisyphus.calculators.Turbomole*), 63, 188
- `get_cubic_crystal()` (in module *pysisyphus.helpers_pure*), 320
- `get_cur_param_density()` (*pysisyphus.cos.GrowingString.GrowingString method*), 113, 219
- `get_curv_vec()` (in module *pysisyphus.irc.initial_displ*), 275
- `get_dask_client()` (*pysisyphus.cos.ChainOfStates.ChainOfStates method*), 109, 216
- `get_data_model()` (in module *pysisyphus.calculators.AFIR*), 73, 154
- `get_data_model()` (in module *pysisyphus.calculators.OverlapCalculator*), 55, 183
- `get_data_model()` (in module *pysisyphus.dynamics.driver*), 232
- `get_data_model()` (in module *pysisyphus.optimizers.Optimizer*), 87, 293
- `get_defaults()` (in module *pysisyphus.run*), 330
- `get_dihedral_inds()` (in module *pysisyphus.intcoords.setup*), 255
- `get_dimer()` (*pysisyphus.calculators.LEPSEExpr.LEPSEExpr method*), 173
- `get_dipole_moment()` (*pysisyphus.calculators.OpenMM.OpenMM method*), 180
- `get_dispersion()` (*pysisyphus.calculators.DFTD4 method*), 197
- `get_dist_func()` (in module *pysisyphus.intcoords.PrimTypes*), 243
- `get_embedding_charges()` (in module *pysisyphus.calculators.ONIOMv2*), 76, 178
- `get_en_conv()` (in module *pysisyphus.plot*), 328
- `get_endpoint_and_ts_geoms()` (*pysisyphus.irc.IRC.IRC method*), 125, 270
- `get_energy()` (*pysisyphus.calculators.AFIR method*), 194
- `get_energy()` (*pysisyphus.calculators.AFIR.AFIR method*), 73, 153
- `get_energy()` (*pysisyphus.calculators.AnaPot2.AnaPot2_ method*), 154
- `get_energy()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase method*), 78, 155
- `get_energy()` (*pysisyphus.calculators.Calculator.Calculator method*), 49, 157
- `get_energy()` (*pysisyphus.calculators.CFOUR method*), 195
- `get_energy()` (*pysisyphus.calculators.CFOUR.CFOUR method*), 69
- `get_energy()` (*pysisyphus.calculators.Composite method*), 196
- `get_energy()` (*pysisyphus.calculators.Composite.Composite method*), 162
- `get_energy()` (*pysisyphus.calculators.ConicalIntersection method*), 196
- `get_energy()` (*pysisyphus.calculators.ConicalIntersection.ConicalIntersection method*), 163
- `get_energy()` (*pysisyphus.calculators.Dalton.Dalton method*), 68, 164
- `get_energy()` (*pysisyphus.calculators.DFTBp method*), 197
- `get_energy()` (*pysisyphus.calculators.DFTBp.DFTBp method*), 63, 163
- `get_energy()` (*pysisyphus.calculators.DFTD4 method*), 197
- `get_energy()` (*pysisyphus.calculators.Dummy method*), 199
- `get_energy()` (*pysisyphus.calculators.Dummy.Dummy method*), 166
- `get_energy()` (*pysisyphus.calculators.EGO method*), 199

`get_energy()` (`pysisyphus.calculators.EGO.EGO` method), 166
`get_energy()` (`pysisyphus.calculators.EnergyMin` method), 200
`get_energy()` (`pysisyphus.calculators.EnergyMin.EnergyMin` method), 167
`get_energy()` (`pysisyphus.calculators.ExternalPotential` method), 201
`get_energy()` (`pysisyphus.calculators.ExternalPotential.ExternalPotential` method), 168
`get_energy()` (`pysisyphus.calculators.Gaussian16` method), 201
`get_energy()` (`pysisyphus.calculators.Gaussian16.Gaussian16` method), 56, 170
`get_energy()` (`pysisyphus.calculators.IPIServer` method), 203
`get_energy()` (`pysisyphus.calculators.IPIServer.IPIServer` method), 172
`get_energy()` (`pysisyphus.calculators.LennardJones` method), 203
`get_energy()` (`pysisyphus.calculators.LennardJones.LennardJones` method), 79, 174
`get_energy()` (`pysisyphus.calculators.MOPAC` method), 204
`get_energy()` (`pysisyphus.calculators.MOPAC.MOPAC` method), 64, 174
`get_energy()` (`pysisyphus.calculators.OBabel` method), 204
`get_energy()` (`pysisyphus.calculators.OBabel.OBabel` method), 68, 175
`get_energy()` (`pysisyphus.calculators.ONIOM` method), 205
`get_energy()` (`pysisyphus.calculators.ONIOMv2.LayerCalc` method), 74, 176
`get_energy()` (`pysisyphus.calculators.ONIOMv2.Model` method), 74, 176
`get_energy()` (`pysisyphus.calculators.ONIOMv2.ModelDummyCalc` method), 75, 177
`get_energy()` (`pysisyphus.calculators.ONIOMv2.ONIOM` method), 75, 177
`get_energy()` (`pysisyphus.calculators.OpenMM.OpenMM` method), 180
`get_energy()` (`pysisyphus.calculators.OpenMolcas` method), 208
`get_energy()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
`get_energy()` (`pysisyphus.calculators.ORCA` method), 206
`get_energy()` (`pysisyphus.calculators.ORCA.ORCA` method), 59, 179
`get_energy()` (`pysisyphus.calculators.Psi4` method), 209
`get_energy()` (`pysisyphus.calculators.Psi4.Psi4` method), 65, 184
`get_energy()` (`pysisyphus.calculators.PyXTB` method), 209
`get_energy()` (`pysisyphus.calculators.PyXTB.PyXTB` method), 184
`get_energy()` (`pysisyphus.calculators.Remote` method), 209
`get_energy()` (`pysisyphus.calculators.Remote.Remote` method), 185
`get_energy()` (`pysisyphus.calculators.SocketCalc.SocketCalc` method), 185
`get_energy()` (`pysisyphus.calculators.TIP3P` method), 210
`get_energy()` (`pysisyphus.calculators.TIP3P.TIP3P` method), 79, 186
`get_energy()` (`pysisyphus.calculators.Turbomole` method), 211
`get_energy()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 61, 187
`get_energy()` (`pysisyphus.calculators.XTB` method), 213
`get_energy()` (`pysisyphus.calculators.XTB.XTB` method), 67, 191
`get_energy_and_cart_forces_at()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_energy_and_cart_hessian_at()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_energy_and_forces_at()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
`get_energy_and_forces_at()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_energy_at()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
`get_energy_at()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_energy_at_cart_coords()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_exc_ens_fosc()` (in module `pysisyphus.calculators.ORCA`), 60, 180

`get_excited_state_str()` (pysisyphus.calculators.DFTBp static method), 197
`get_excited_state_str()` (pysisyphus.calculators.DFTBp.DFTBp static method), 63, 163
`get_expr()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
`get_fchk_str()` (pysisyphus.calculators.Psi4 method), 209
`get_fchk_str()` (pysisyphus.calculators.Psi4.Psi4 method), 65, 184
`get_fg()` (pysisyphus.line_searches.LineSearch.LineSearch method), 279
`get_fh_logger()` (in module pysisyphus.init_logging), 322
`get_final_energy()` (pysisyphus.calculators.Composite method), 196
`get_final_energy()` (pysisyphus.calculators.Composite.Composite method), 162
`get_fixed_indices()` (pysisyphus.cos.ChainOfStates.ChainOfStates method), 109, 216
`get_fmfs()` (in module pysisyphus.socket_helper), 331
`get_force_unit()` (in module pysisyphus.plot), 328
`get_forces()` (pysisyphus.calculators.AFIR method), 194
`get_forces()` (pysisyphus.calculators.AFIR.AFIR method), 73, 153
`get_forces()` (pysisyphus.calculators.AnaPotBase.AnaPotBase method), 78, 155
`get_forces()` (pysisyphus.calculators.AtomAtomTransTorque method), 194
`get_forces()` (pysisyphus.calculators.AtomAtomTransTorque.AtomAtomTransTorque method), 156
`get_forces()` (pysisyphus.calculators.Calculator.Calculator method), 49, 157
`get_forces()` (pysisyphus.calculators.CFOUR method), 195
`get_forces()` (pysisyphus.calculators.CFOUR.CFOUR method), 69
`get_forces()` (pysisyphus.calculators.Composite method), 196
`get_forces()` (pysisyphus.calculators.Composite.Composite method), 162
`get_forces()` (pysisyphus.calculators.ConicalIntersection method), 196
`get_forces()` (pysisyphus.calculators.ConicalIntersection.ConicalIntersection method), 163
`get_forces()` (pysisyphus.calculators.Dalton.Dalton method), 68, 164
`get_forces()` (pysisyphus.calculators.DFTBp method), 197
`get_forces()` (pysisyphus.calculators.DFTBp.DFTBp method), 63, 163
`get_forces()` (pysisyphus.calculators.DFTD4 method), 197
`get_forces()` (pysisyphus.calculators.Dimer method), 198
`get_forces()` (pysisyphus.calculators.Dimer.Dimer method), 77, 121, 165
`get_forces()` (pysisyphus.calculators.Dummy method), 199
`get_forces()` (pysisyphus.calculators.Dummy.Dummy method), 166
`get_forces()` (pysisyphus.calculators.EGO method), 199
`get_forces()` (pysisyphus.calculators.EGO.EGO method), 166
`get_forces()` (pysisyphus.calculators.EnergyMin method), 200
`get_forces()` (pysisyphus.calculators.EnergyMin.EnergyMin method), 167
`get_forces()` (pysisyphus.calculators.ExternalPotential method), 201
`get_forces()` (pysisyphus.calculators.ExternalPotential.ExternalPotential method), 168
`get_forces()` (pysisyphus.calculators.FakeASE method), 201
`get_forces()` (pysisyphus.calculators.FakeASE.FakeASE method), 79, 169
`get_forces()` (pysisyphus.calculators.Gaussian16 method), 201
`get_forces()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170
`get_forces()` (pysisyphus.calculators.HardSphere method), 202
`get_forces()` (pysisyphus.calculators.HardSphere.HardSphere method), 171
`get_forces()` (pysisyphus.calculators.HardSphere.PWHardSphere method), 171
`get_forces()` (pysisyphus.calculators.IDPPCalculator.IDPPCalculator method), 196

- method), 172
- get_forces() (pysisyphus.calculators.IPIServer method), 203
- get_forces() (pysisyphus.calculators.IPIServer.IPIServer method), 172
- get_forces() (pysisyphus.calculators.LennardJones method), 203
- get_forces() (pysisyphus.calculators.LennardJones.LennardJones method), 79, 174
- get_forces() (pysisyphus.calculators.MOPAC method), 204
- get_forces() (pysisyphus.calculators.MOPAC.MOPAC method), 64, 174
- get_forces() (pysisyphus.calculators.OBabel method), 205
- get_forces() (pysisyphus.calculators.OBabel.OBabel method), 68, 175
- get_forces() (pysisyphus.calculators.ONIOM method), 205
- get_forces() (pysisyphus.calculators.ONIOMv2.LayerCalc method), 74, 176
- get_forces() (pysisyphus.calculators.ONIOMv2.Model method), 74, 176
- get_forces() (pysisyphus.calculators.ONIOMv2.ModelDummyCalc method), 75, 177
- get_forces() (pysisyphus.calculators.ONIOMv2.ONIOM method), 75, 177
- get_forces() (pysisyphus.calculators.OpenMM.OpenMM method), 180
- get_forces() (pysisyphus.calculators.OpenMolcas method), 208
- get_forces() (pysisyphus.calculators.OpenMolcas.OpenMolcas method), 58, 181
- get_forces() (pysisyphus.calculators.ORCA method), 206
- get_forces() (pysisyphus.calculators.ORCA.ORCA method), 59, 179
- get_forces() (pysisyphus.calculators.Psi4 method), 209
- get_forces() (pysisyphus.calculators.Psi4.Psi4 method), 65, 184
- get_forces() (pysisyphus.calculators.PyPsi4 method), 209
- get_forces() (pysisyphus.calculators.PyPsi4.PyPsi4 method), 184
- get_forces() (pysisyphus.calculators.PyXTB method), 209
- get_forces() (pysisyphus.calculators.PyXTB.PyXTB method), 184
- get_forces() (pysisyphus.calculators.Remote method), 209
- get_forces() (pysisyphus.calculators.Remote.Remote method), 185
- get_forces() (pysisyphus.calculators.SocketCalc.SocketCalc method), 185
- get_forces() (pysisyphus.calculators.TIP3P method), 210
- get_forces() (pysisyphus.calculators.TIP3P.TIP3P method), 79, 186
- get_forces() (pysisyphus.calculators.TransTorque method), 210
- get_forces() (pysisyphus.calculators.TransTorque.TransTorque method), 186
- get_forces() (pysisyphus.calculators.Turbomole method), 211
- get_forces() (pysisyphus.calculators.Turbomole.Turbomole method), 61, 187
- get_forces() (pysisyphus.calculators.XTB method), 213
- get_forces() (pysisyphus.calculators.XTB.XTB method), 67, 191
- get_forces_naive() (pysisyphus.calculators.TransTorque method), 210
- get_forces_naive() (pysisyphus.calculators.TransTorque.TransTorque method), 186
- get_fourth_coords() (pysisyphus.intcoords.DummyTorsion static method), 259
- get_fourth_coords() (pysisyphus.intcoords.DummyTorsion.DummyTorsion static method), 241
- get_fragments() (in module pysisyphus.intcoords.setup), 255
- get_fragments() (pysisyphus.Geometry.Geometry method), 39, 313
- get_fragments() (pysisyphus.stochastic.FragmentKick method), 304
- get_fragments() (pysisyphus.stochastic.FragmentKick.FragmentKick method), 302
- get_fragments_and_bonds() (in module pysisyphus.drivers.precon_pos_rot), 226
- get_from_to_sets() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189
- get_full_irc_data() (pysisyphus.irc.IRC.IRC

method), 125, 270

get_g_value() (in module *pysisyphus.calculators.ONIOMv2*), 76, 178

get_gaussian_energies() (*pysisyphus.calculators.Dimer* method), 198

get_gaussian_energies() (*pysisyphus.calculators.Dimer.Dimer* method), 77, 121, 165

get_gaussian_forces() (*pysisyphus.calculators.Dimer* method), 198

get_gaussian_forces() (*pysisyphus.calculators.Dimer.Dimer* method), 77, 121, 165

get_gen_str() (*pysisyphus.calculators.DFTBp* static method), 197

get_gen_str() (*pysisyphus.calculators.DFTBp.DFTBp* static method), 63, 163

get_geom() (*pysisyphus.calculators.AnaPotBase.AnaPotBase* class method), 78, 155

get_geom_getter() (in module *pysisyphus.helpers*), 319

get_geom_kwargs() (in module *pysisyphus.optimizers.LayerOpt*), 96, 289

get_geoms() (in module *pysisyphus.trj*), 332

get_geoms() (*pysisyphus.benchmarks.Benchmark.Benchmark* method), 151

get_geoms_from_stored_coords() (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155

get_gs_line() (*pysisyphus.calculators.WFOWrapper.WFOWrapper* method), 189

get_guess_hessian() (in module *pysisyphus.optimizers.guess_hessians*), 298

get_h5_group() (in module *pysisyphus.io.hdf5*), 264

get_h5_group() (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* method), 54, 182

get_harmonic() (*pysisyphus.calculators.LEPSEExpr.LEPSEExpr* method), 173

get_hei_index() (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 109, 216

get_hessian() (*pysisyphus.calculators.AFIR* method), 194

get_hessian() (*pysisyphus.calculators.AFIR.AFIR* method), 73, 153

get_hessian() (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155

get_hessian() (*pysisyphus.calculators.Calculator.Calculator* method), 49, 157

get_hessian() (*pysisyphus.calculators.Composite* method), 196

get_hessian() (*pysisyphus.calculators.Composite.Composite* method), 162

get_hessian() (*pysisyphus.calculators.ConicalIntersection* method), 196

get_hessian() (*pysisyphus.calculators.ConicalIntersection.ConicalIntersection* method), 163

get_hessian() (*pysisyphus.calculators.Dimer* method), 198

get_hessian() (*pysisyphus.calculators.Dimer.Dimer* method), 77, 121, 165

get_hessian() (*pysisyphus.calculators.Dummy* method), 199

get_hessian() (*pysisyphus.calculators.Dummy.Dummy* method), 166

get_hessian() (*pysisyphus.calculators.EnergyMin* method), 200

get_hessian() (*pysisyphus.calculators.EnergyMin.EnergyMin* method), 167

get_hessian() (*pysisyphus.calculators.ExternalPotential* method), 201

get_hessian() (*pysisyphus.calculators.ExternalPotential.ExternalPotential* method), 168

get_hessian() (*pysisyphus.calculators.Gaussian16* method), 201

get_hessian() (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 56, 170

get_hessian() (*pysisyphus.calculators.IPIServer* method), 203

get_hessian() (*pysisyphus.calculators.IPIServer.IPIServer* method), 172

get_hessian() (*pysisyphus.calculators.MOPAC* method), 204

get_hessian() (*pysisyphus.calculators.MOPAC.MOPAC* method), 64, 174

get_hessian() (*pysisyphus.calculators.ONIOM* method), 205

get_hessian() (*pysisyphus.calculators.ONIOMv2.LayerCalc* method), 74, 176

get_hessian() (*pysisyphus.calculators.ONIOMv2.Model* method), 74, 176

- `get_hessian()` (pysisyphus.calculators.ONIOMv2.ONIOM method), 75, 177
- `get_hessian()` (pysisyphus.calculators.ORCA method), 206
- `get_hessian()` (pysisyphus.calculators.ORCA.ORCA method), 59, 179
- `get_hessian()` (pysisyphus.calculators.Psi4 method), 209
- `get_hessian()` (pysisyphus.calculators.Psi4.Psi4 method), 65, 184
- `get_hessian()` (pysisyphus.calculators.Remote method), 209
- `get_hessian()` (pysisyphus.calculators.Remote.Remote method), 185
- `get_hessian()` (pysisyphus.calculators.SocketCalc.SocketCalc method), 185
- `get_hessian()` (pysisyphus.calculators.Turbomole method), 211
- `get_hessian()` (pysisyphus.calculators.Turbomole.Turbomole method), 61, 187
- `get_hessian()` (pysisyphus.calculators.XTB method), 213
- `get_hessian()` (pysisyphus.calculators.XTB.XTB method), 67, 191
- `get_hydrogen_bond_inds()` (in module pysisyphus.intcoords.setup), 255
- `get_hydrogen_bond_inds_v2()` (in module pysisyphus.intcoords.setup), 255
- `get_imag_frequencies()` (pysisyphus.Geometry.Geometry method), 39, 313
- `get_image_calc_counter_sum()` (pysisyphus.cos.ChainOfStates.ChainOfStates method), 109, 216
- `get_index_of_typed_prim()` (pysisyphus.intcoords.RedundantCoords method), 261
- `get_index_of_typed_prim()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 245
- `get_indices()` (pysisyphus.calculators.OverlapCalculator.OverlapCalculator method), 54, 182
- `get_input()` (in module pysisyphus.helpers_pure), 321
- `get_input_geom()` (pysisyphus.stochastic.FragmentKick method), 304
- `get_input_geom()` (pysisyphus.stochastic.FragmentKick.FragmentKick method), 302
- `get_input_geom()` (pysisyphus.stochastic.Kick method), 304
- `get_input_geom()` (pysisyphus.stochastic.Kick.Kick method), 303
- `get_input_geom()` (pysisyphus.stochastic.Pipeline.Pipeline method), 303
- `get_integration_length_func()` (pysisyphus.irc.EulerPC method), 276
- `get_integration_length_func()` (pysisyphus.irc.EulerPC.EulerPC method), 127, 268
- `get_interactively()` (in module pysisyphus.trj), 332
- `get_irc_data()` (pysisyphus.irc.IRC.IRC method), 125, 271
- `get_iteration()` (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190
- `get_jacobian()` (pysisyphus.calculators.ONIOMv2.Model method), 74, 176
- `get_k()` (pysisyphus.irc.RK4 method), 277
- `get_k()` (pysisyphus.irc.RK4.RK4 method), 128, 274
- `get_K_matrix()` (pysisyphus.intcoords.RedundantCoords method), 261
- `get_K_matrix()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 245
- `get_kick()` (pysisyphus.stochastic.Kick method), 304
- `get_kick()` (pysisyphus.stochastic.Kick.Kick method), 303
- `get_last_calc_cycle()` (in module pysisyphus.run), 331
- `get_layer_calc()` (pysisyphus.calculators.ONIOM method), 205
- `get_layer_calc()` (pysisyphus.calculators.ONIOMv2.ONIOM method), 75, 177
- `get_lbfgs_step()` (pysisyphus.optimizers.LBFGS.LBFGS method), 97, 288
- `get_leps()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
- `get_lindh_alpha()` (in module pysisyphus.optimizers.guess_hessians), 298
- `get_lindh_k()` (in module pysisyphus.optimizers.precon), 301
- `get_lindh_precon()` (in module pysisyphus.optimizers.precon), 301
- `get_linear_bond_inds()` (in module pysisyphus.intcoords.setup), 255
- `get_loader()` (in module pysisyphus.yaml_mods), 334
- `get_max_bond_dists()` (in module pysisyphus.intcoords.setup_fast), 256
- `get_maximum()` (in module pysisyphus.optimizers.poly_fit), 300

`get_mb_velocities()` (in module `pysisyphus.dynamics.helpers`), 233
`get_mb_velocities_for_geom()` (in module `pysisyphus.dynamics.helpers`), 234
`get_mdrestart_str()` (`pysisyphus.calculators.XTB` method), 213
`get_mdrestart_str()` (`pysisyphus.calculators.XTB.XTB` method), 67, 191
`get_minima()` (`pysisyphus.calculators.AnaPotBase.AnaPotBase` method), 78, 155
`get_minimum()` (in module `pysisyphus.optimizers.poly_fit`), 300
`get_mo_norms()` (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` static method), 54, 183
`get_model()` (`pysisyphus.calculators.DFTD4` method), 197
`get_mods()` (`pysisyphus.calculators.EGO` method), 199
`get_mods()` (`pysisyphus.calculators.EGO.EGO` method), 166
`get_moinp_str()` (`pysisyphus.calculators.ORCA` method), 206
`get_moinp_str()` (`pysisyphus.calculators.ORCA.ORCA` method), 59, 179
`get_molecular_radius()` (in module `pysisyphus.helpers_pure`), 321
`get_molecules_levels()` (in module `pysisyphus.db.helpers`), 221
`get_mwfn_exc_str()` (in module `pysisyphus.wrapper.mwfn`), 309
`get_N_raw_from_hessian()` (`pysisyphus.calculators.Dimer` method), 198
`get_N_raw_from_hessian()` (`pysisyphus.calculators.Dimer.Dimer` method), 77, 121, 165
`get_name()` (in module `pysisyphus.calculators.ORCA`), 60, 180
`get_new_image()` (`pysisyphus.cos.FreezingString.FreezingString` method), 218
`get_new_image()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`get_new_image_from_coords()` (`pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates` method), 112, 218
`get_newton_step()` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` static method), 88, 286
`get_newton_step_on_trust()` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 287
`get_normal_modes()` (`pysisyphus.Geometry.Geometry` method), 39, 313
`get_num_hessian()` (`pysisyphus.calculators.Calculator.Calculator` method), 49, 157
`get_opt_cls()` (in module `pysisyphus.optimizers.cls_map`), 297
`get_opt_kwargs()` (in module `pysisyphus.optimizers.LayerOpt`), 96, 289
`get_optimal_bias()` (in module `pysisyphus.drivers.opt`), 225
`get_orbital_matrices()` (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` method), 54, 183
`get_origin()` (`pysisyphus.stochastic.FragmentKick` method), 304
`get_origin()` (`pysisyphus.stochastic.FragmentKick.FragmentKick` method), 302
`get_P()` (in module `pysisyphus.calculators.ConicalIntersection`), 163
`get_pair_covalent_radii()` (in module `pysisyphus.intcoords.setup`), 255
`get_pal_env()` (`pysisyphus.calculators.OpenMolcas` method), 208
`get_pal_env()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
`get_pal_env()` (`pysisyphus.calculators.Turbomole` method), 211
`get_pal_env()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 62, 187
`get_pal_env()` (`pysisyphus.calculators.XTB` method), 213
`get_pal_env()` (`pysisyphus.calculators.XTB.XTB` method), 67, 191
`get_parallel_forces()` (`pysisyphus.cos.NEB.NEB` method), 110, 220
`get_param()` (`pysisyphus.irc.ParamPlot.ParamPlot` method), 274
`get_parser()` (in module `pysisyphus.io.pdb`), 265
`get_path()` (in module `pysisyphus.db.helpers`), 221
`get_path()` (`pysisyphus.calculators.AnaPotBase.AnaPotBase` method), 78, 155
`get_path()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
`get_path_for_fn()` (`pysisyphus.irc.IRC.IRC` method), 125, 271
`get_path_for_fn()` (`pysisyphus.optimizers.Optimizer.Optimizer` method), 86, 292
`get_perpendicular_forces()` (`pysisyphus`

- phus.cos.ChainOfStates.ChainOfStates* method), 109, 216
- `get_phi_dphi()` (*pysisyphus.line_searches.LineSearch.LineSearch* method), 279
- `get_potential_energy()` (*pysisyphus.calculators.ExternalPotential* method), 201
- `get_potential_energy()` (*pysisyphus.calculators.ExternalPotential.ExternalPotential* method), 168
- `get_potential_energy()` (*pysisyphus.calculators.FakeASE* method), 201
- `get_potential_energy()` (*pysisyphus.calculators.FakeASE.FakeASE* method), 79, 169
- `get_potential_forces()` (*pysisyphus.calculators.ExternalPotential* method), 201
- `get_potential_forces()` (*pysisyphus.calculators.ExternalPotential.ExternalPotential* method), 168
- `get_precon_getter()` (*pysisyphus.optimizers.PreconLBFGS.PreconLBFGS* method), 98, 294
- `get_precon_pos_rot_data()` (in module *pysisyphus.benchmarks.data*), 152
- `get_prim_internals_by_indices()` (*pysisyphus.intcoords.RedundantCoords* method), 261
- `get_prim_internals_by_indices()` (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 245
- `get_primitives()` (in module *pysisyphus.intcoords.setup*), 255
- `get_quenched_dneb_forces()` (*pysisyphus.cos.NEB.NEB* method), 110, 220
- `get_r()` (*pysisyphus.cos.GrowingNT.GrowingNT* static method), 219
- `get_r0s()` (in module *pysisyphus.drivers.birkholz*), 224
- `get_random_path()` (in module *pysisyphus.helpers_pure*), 321
- `get_rates()` (in module *pysisyphus.drivers.rates*), 229
- `get_rates_for_geoms()` (in module *pysisyphus.drivers.rates*), 229
- `get_rates_for_hessians()` (in module *pysisyphus.drivers.rates*), 229
- `get_ratio()` (in module *pysisyphus.helpers_pure*), 321
- `get_ref_mos()` (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* method), 55, 183
- `get_restart_info()` (*pysisyphus.calculators.Calculator.Calculator* method), 49, 157
- `get_restart_info()` (*pysisyphus.Geometry.Geometry* method), 39, 314
- `get_restart_info()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 86, 292
- `get_retry_args()` (*pysisyphus.calculators.XTB* method), 213
- `get_retry_args()` (*pysisyphus.calculators.XTB.XTB* method), 67, 191
- `get_ricc2_root()` (*pysisyphus.calculators.Turbomole* method), 211
- `get_ricc2_root()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
- `get_root()` (*pysisyphus.calculators.OpenMolcas* method), 208
- `get_root()` (*pysisyphus.calculators.OpenMolcas.OpenMolcas* method), 58, 181
- `get_rot_coord()` (in module *pysisyphus.intcoords.PrimTypes*), 243
- `get_rot_mat()` (in module *pysisyphus.drivers.precon_pos_rot*), 226
- `get_rot_mat()` (in module *pysisyphus.linalg*), 323
- `get_rot_mat_for_coords()` (in module *pysisyphus.linalg*), 323
- `get_rs_step()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 88, 287
- `get_s22_data()` (in module *pysisyphus.benchmarks.data*), 152
- `get_saddles()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `get_sao_from_mo_coeffs()` (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* static method), 55, 183
- `get_scale_max()` (in module *pysisyphus.optimizers.restrict_step*), 301
- `get_shifted_step_trans()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* static method), 88, 287
- `get_sparse_jacobian()` (*pysisyphus.calculators.ONIOMv2.Model* method), 74, 176
- `get_sphere_radius()` (*pysisyphus.Geometry.Geometry* method), 39, 314
- `get_splined_hei()` (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 109, 216
- `get_spring_forces()` (*pysisyphus.cos.NEB.NEB* method), 110, 220
- `get_stable_wavefunction()` (*pysisyphus.calculators.ORCA* method), 206
- `get_stable_wavefunction()` (*pysisyphus* method), 206

`phus.calculators.ORCA.ORCA` (method), 59, 179
`get_step()` (in module `pysisyphus.intcoords.helpers`), 251
`get_step_func()` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 287
`get_steps_to_active_atom_mean()` (in module `pysisyphus.drivers.precon_pos_rot`), 226
`get_subgeom()` (`pysisyphus.Geometry.Geometry` method), 39, 314
`get_subgeom_without()` (`pysisyphus.Geometry.Geometry` method), 39, 314
`get_tangent()` (in module `pysisyphus.intcoords.helpers`), 251
`get_tangent()` (`pysisyphus.cos.ChainOfStates.ChainOfStates` method), 109, 216
`get_tangent()` (`pysisyphus.cos.FreezingString.FreezingString` method), 218
`get_tangent()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`get_tangent_trj_str()` (in module `pysisyphus.helpers`), 319
`get_tangents()` (`pysisyphus.cos.ChainOfStates.ChainOfStates` method), 109, 216
`get_tden_overlaps()` (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` method), 55, 183
`get_temporary_coords()` (`pysisyphus.Geometry.Geometry` method), 39, 314
`get_thermoanalysis()` (`pysisyphus.Geometry.Geometry` method), 39, 314
`get_thermoanalysis_from_hess_h5()` (in module `pysisyphus.thermo`), 332
`get_tm_indices()` (in module `pysisyphus.elem_data`), 318
`get_top_differences()` (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` method), 55, 183
`get_tot()` (`pysisyphus.calculators.LEPSEExpr.LEPSEExpr` method), 173
`get_trans_rot_projector()` (in module `pysisyphus.Geometry`), 42, 316
`get_trans_rot_projector()` (`pysisyphus.Geometry.Geometry` method), 39, 314
`get_trans_rot_vectors()` (in module `pysisyphus.Geometry`), 42, 316
`get_trans_torque_forces()` (in module `pysisyphus.calculators.TransTorque`), 186
`get_unique_geometries()` (`pysisyphus.calculators.ORCA.ORCA` method), 59, 179
`get_unique_internals()` (in module `pysisyphus.filtertrj`), 318
`get_update_mu_reg()` (in module `pysisyphus.optimizers.closures`), 296
`get_valid_index_set()` (`pysisyphus.stochastic.Pipeline.Pipeline` method), 303
`get_weighted_bond_mode()` (in module `pysisyphus.intcoords.helpers`), 251
`get_weighted_bond_mode_getter()` (in module `pysisyphus.intcoords.helpers`), 251
`get_which_frag()` (in module `pysisyphus.drivers.precon_pos_rot`), 227
`get_xtb_rx_data()` (in module `pysisyphus.benchmarks.data`), 152
`get_zimmerman_data()` (in module `pysisyphus.benchmarks.data`), 152
`get_zimmerman_xtb_data()` (in module `pysisyphus.benchmarks.data`), 152
`GonzalezSchlegel` (class in `pysisyphus.irc`), 276
`GonzalezSchlegel` (class in `pysisyphus.irc.GonzalezSchlegel`), 127, 269
`got_alpha_phi_dphi()` (`pysisyphus.line_searches.LineSearch.LineSearch` method), 279
`GotNoGeometryException`, 332
`gradient` (`pysisyphus.cos.ChainOfStates.ChainOfStates` property), 109, 216
`gradient` (`pysisyphus.Geometry.Geometry` property), 40, 314
`gradient` (`pysisyphus.irc.Instanton.Instanton` property), 273
`gradient` (`pysisyphus.irc.IRC.IRC` property), 125, 271
`gradient()` (`pysisyphus.dynamics.colvars.Colvar` method), 232
`gradient()` (`pysisyphus.dynamics.Gaussian.Gaussian` method), 231
`gradient1` (`pysisyphus.calculators.ConicalIntersection.CIQuantities` attribute), 162
`gradient2` (`pysisyphus.calculators.ConicalIntersection.CIQuantities` attribute), 162
`gradient_diff` (`pysisyphus.calculators.ConicalIntersection.CIQuantities` attribute), 162
`gradient_mean` (`pysisyphus.calculators.ConicalIntersection.CIQuantities` attribute), 162
`gram_schmidt()` (in module `pysisyphus.linalg`), 323
`green()` (in module `pysisyphus.color`), 318
`grow_image()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
`GrowingChainOfStates` (class in `pysisyphus`), 219

phus.cos.GrowingChainOfStates), 112, 218
 GrowingNT (class in *pysisyphus.cos.GrowingNT*), 218
 GrowingString (class in *pysisyphus.cos.GrowingString*), 113, 219

H

H5_MAP (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* attribute), 54, 182
 H_minus (*pysisyphus.irc.initial_displ.ThirdDerivResult* attribute), 275
 H_plus (*pysisyphus.irc.initial_displ.ThirdDerivResult* attribute), 275
 HagerZhang (class in *pysisyphus.line_searches*), 281
 HagerZhang (class in *pysisyphus.line_searches.HagerZhang*), 278
 HardSphere (class in *pysisyphus.calculators*), 202
 HardSphere (class in *pysisyphus.calculators.HardSphere*), 171
 hardsphere_merge() (in module *pysisyphus.drivers.merge*), 224
 hardsphere_merge() (in module *pysisyphus.trj*), 332
 harmonic_tst_rate() (in module *pysisyphus.drivers.rates*), 229
 HarmonicSphere (class in *pysisyphus.calculators.ExternalPotential*), 168
 hash_args() (in module *pysisyphus.helpers_pure*), 321
 hash_arr() (in module *pysisyphus.helpers_pure*), 321
 HDLC (class in *pysisyphus.intcoords*), 259
 HDLC (class in *pysisyphus.intcoords.DLC*), 240
 header (*pysisyphus.TableFormatter.TableFormatter* property), 317
 HEIIsFirstOrLastException, 318
 hessian (*pysisyphus.Geometry.Geometry* property), 40, 314
 hessian (*pysisyphus.irc.Instanton.Instanton* property), 273
 HessianOptimizer (class in *pysisyphus.optimizers.HessianOptimizer*), 87, 285
 HessKind (class in *pysisyphus.calculators.Calculator*), 53, 161
 highlight_text() (in module *pysisyphus.helpers_pure*), 321
 housekeeping() (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 88, 287
 hubbard_derivs (*pysisyphus.calculators.DFTBp* attribute), 197
 hubbard_derivs (*pysisyphus.calculators.DFTBp.DFTBp* attribute), 63, 163
 HybridRedundantCoords (class in *pysisyphus.intcoords*), 259

HybridRedundantCoords (class in *pysisyphus.intcoords.RedundantCoords*), 42, 244
 HYDROGEN_BOND (*pysisyphus.intcoords.PrimTypes.PrimTypes* attribute), 242
 hydrogen_bonds (*pysisyphus.intcoords.setup.CoordInfo* attribute), 254

I

i10() (in module *pysisyphus.io.crd*), 264
 IDPP (class in *pysisyphus.interpolate.IDPP*), 263
 IDPPCalculator (class in *pysisyphus.calculators.IDPPCalculator*), 172
 imag_fns (*pysisyphus.helpers.FinalHessianResult* attribute), 318
 imag_frequency (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 imag_modes_from_geom() (in module *pysisyphus.helpers*), 319
 imag_wavenumber (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 image_coords (*pysisyphus.cos.ChainOfStates.ChainOfStates* property), 109, 216
 image_inds (*pysisyphus.cos.ChainOfStates.ChainOfStates* property), 109, 216
 image_inds (*pysisyphus.cos.GrowingString.GrowingString* property), 113, 220
 IMKMod (class in *pysisyphus.irc*), 277
 IMKMod (class in *pysisyphus.irc.IMKMod*), 128, 269
 IMPROPER_DIHEDRAL (*pysisyphus.intcoords.PrimTypes.PrimTypes* attribute), 242
 improper_dihedrals (*pysisyphus.intcoords.setup.CoordInfo* attribute), 254
 improved_guess() (in module *pysisyphus.optimizers.guess_hessians*), 298
 increment_fn() (in module *pysisyphus.helpers_pure*), 321
 ind (*pysisyphus.calculators.ONIOMv2.Link* attribute), 74, 176
 index (*pysisyphus.intcoords.Rotation.Rotation* attribute), 246
 index (*pysisyphus.intcoords.Rotation.RotationA* attribute), 246
 index (*pysisyphus.intcoords.Rotation.RotationB* attribute), 247
 index (*pysisyphus.intcoords.Rotation.RotationC* attribute), 247
 index (*pysisyphus.intcoords.RotationA* attribute), 262
 index (*pysisyphus.intcoords.RotationB* attribute), 262

- index (*pysisyphus.intcoords.RotationC* attribute), 262
 index_array_from_overlaps() (in module *pysisyphus.helpers*), 319
 index_strs_for_atoms() (in module *pysisyphus.calculators.Turbomole*), 63, 188
 inds_from_prim_types() (in module *pysisyphus.intcoords.update*), 256
 inertia_tensor (*pysisyphus.Geometry.Geometry* property), 40, 314
 inertia_tensor() (in module *pysisyphus.Geometry*), 42, 317
 init_h5_group() (in module *pysisyphus.io.hdf5*), 264
 init_h5_group() (*pysisyphus.calculators.AFIR* method), 194
 init_h5_group() (*pysisyphus.calculators.AFIR.AFIR* method), 73, 153
 init_logging() (in module *pysisyphus.init_logging*), 322
 init_logging_base() (in module *pysisyphus.init_logging*), 322
 initial() (*pysisyphus.line_searches.HagerZhang* method), 281
 initial() (*pysisyphus.line_searches.HagerZhang.HagerZhang* method), 278
 initial_displacement() (*pysisyphus.irc.IRC.IRC* method), 125, 271
 initialize() (*pysisyphus.cos.GrowingNT.GrowingNT* method), 219
 instant_pressure() (*pysisyphus.calculators.ExternalPotential.HarmonicSphere* method), 168
 Instanton (class in *pysisyphus.irc.Instanton*), 272
 INTERFRAG_BOND (*pysisyphus.intcoords.PrimTypes.PrimTypes* attribute), 242
 interfrag_bonds (*pysisyphus.intcoords.setup.CoordInfo* attribute), 254
 interfragment_distance() (in module *pysisyphus.intcoords.helpers*), 251
 interpol_alpha_cubic() (in module *pysisyphus.line_searches.interpol*), 280
 interpol_alpha_quad() (in module *pysisyphus.line_searches.interpol*), 280
 interpolate() (in module *pysisyphus.interpolate.helpers*), 264
 interpolate() (*pysisyphus.interpolate.IDPP.IDPP* method), 263
 interpolate() (*pysisyphus.interpolate.Interpolator.Interpolator* method), 263
 interpolate() (*pysisyphus.interpolate.LST.LST* method), 263
 interpolate() (*pysisyphus.interpolate.Redund.Redund* method), 263
 interpolate() (*pysisyphus.irc.DWI.DWI* method), 267
 interpolate_all() (in module *pysisyphus.interpolate.helpers*), 264
 interpolate_all() (*pysisyphus.interpolate.Interpolator.Interpolator* method), 263
 interpolate_colors() (in module *pysisyphus.helpers_pure*), 321
 Interpolator (class in *pysisyphus.interpolate.Interpolator*), 263
 interval_update() (*pysisyphus.line_searches.HagerZhang* method), 281
 interval_update() (*pysisyphus.line_searches.HagerZhang.HagerZhang* method), 278
 inv_B() (*pysisyphus.intcoords.RedundantCoords* method), 261
 inv_B() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 245
 inv_Bt() (*pysisyphus.intcoords.RedundantCoords* method), 261
 inv_Bt() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 245
 inv_masses_rep_sqrt (*pysisyphus.intcoords.CartesianCoords* property), 257
 inv_masses_rep_sqrt (*pysisyphus.intcoords.CartesianCoords.CartesianCoords* property), 238
 ipi_client() (in module *pysisyphus.calculators.IPIClient*), 172
 IPIServer (class in *pysisyphus.calculators*), 203
 IPIServer (class in *pysisyphus.calculators.IPIServer*), 172
 IRC (class in *pysisyphus.irc.IRC*), 124, 269
 irc (*pysisyphus.run.RunResult* attribute), 330
 irc() (*pysisyphus.irc.IRC.IRC* method), 126, 271
 irc_geom (*pysisyphus.run.RunResult* attribute), 330
 IRCDummy (class in *pysisyphus.irc.IRCDummy*), 272
 is_analytical_2d (*pysisyphus.Geometry.Geometry* property), 40, 314
 is_analytical_2d() (*pysisyphus.irc.Instanton.Instanton* method), 273
 is_converged() (*pysisyphus.optimizers.Optimizer.ConvInfo* method), 84, 290
- ## J
- J() (*pysisyphus.calculators.LEPSEXP.LEPSEXP* method), 173
 jacobian() (*pysisyphus.intcoords.LinearBend* method), 259

jacobian() (*pysisyphus.intcoords.LinearBend.LinearBend* attribute), 241
 jacobian() (*pysisyphus.intcoords.LinearDisplacement* attribute), 259
 jacobian() (*pysisyphus.intcoords.LinearDisplacement.LinearDisplacement* attribute), 241
 jacobian() (*pysisyphus.intcoords.Primitive.Primitive* attribute), 244
 jmol() (*pysisyphus.Geometry.Geometry* attribute), 40, 314
 join_format() (*pysisyphus.TableFormatter.TableFormatter* attribute), 317
 json_to_results() (in module *pysisyphus.helpers_pure*), 321

K

kappa_bell (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 kappa_eckart (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 kappa_eyring (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 kappa_wigner (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
 keep() (*pysisyphus.calculators.Calculator.Calculator* attribute), 49, 157
 keep() (*pysisyphus.calculators.CFOUR* attribute), 195
 keep() (*pysisyphus.calculators.CFOUR.CFOUR* attribute), 69
 KeepKind (class in *pysisyphus.calculators.Calculator*), 53, 161
 key (*pysisyphus.calculators.Calculator.SetPlan* attribute), 53, 161
 key_is_tsopt() (in module *pysisyphus.optimizers.cls_map*), 297
 Kick (class in *pysisyphus.stochastic*), 304
 Kick (class in *pysisyphus.stochastic.Kick*), 303
 kick_fragment() (*pysisyphus.stochastic.FragmentKick* attribute), 304
 kick_fragment() (*pysisyphus.stochastic.FragmentKick.FragmentKick* attribute), 302
 kill_dir() (in module *pysisyphus.helpers_pure*), 321
 kinetic_energy_for_temperature() (in module *pysisyphus.dynamics.helpers*), 234
 kinetic_energy_from_velocities() (in module *pysisyphus.dynamics.helpers*), 234

L

l_mw (*pysisyphus.modelfollow.NormalMode* attribute), 283
 l_mw (*pysisyphus.modelfollow.NormalMode.NormalMode* attribute), 282
 lambdas (*pysisyphus.calculators.OverlapCalculator.NTOs* attribute), 53, 182
 lanczos() (in module *pysisyphus.modelfollow.lanczos*), 283
 last_index (*pysisyphus.cos.ChainOfStates.ChainOfStates* attribute), 109, 216
 last_two_coords (*pysisyphus.calculators.WFOWrapper2.WFOWrapper2* attribute), 190
 LATEST (*pysisyphus.calculators.Calculator.KeepKind* attribute), 53, 161
 layer_num (*pysisyphus.optimizers.LayerOpt.LayerOpt* attribute), 96, 288
 LayerCalc (class in *pysisyphus.calculators.ONIOMv2*), 74, 176
 LayerOpt (class in *pysisyphus.optimizers.LayerOpt*), 96, 288
 Layers (class in *pysisyphus.optimizers.LayerOpt*), 96, 289
 layers (*pysisyphus.Geometry.Geometry* attribute), 40, 314
 LBFGS (class in *pysisyphus.optimizers.LBFGS*), 96, 287
 lbfgs_closure() (in module *pysisyphus.optimizers.closures*), 296
 lbfgs_step() (*pysisyphus.optimizers.MicroOptimizer* attribute), 301
 lbfgs_step() (*pysisyphus.optimizers.MicroOptimizer.MicroOptimizer* attribute), 289
 left_frontier (*pysisyphus.cos.FreezingString.FreezingString* attribute), 218
 left_size (*pysisyphus.cos.GrowingString.GrowingString* attribute), 113, 220
 length_for_bond_order() (in module *pysisyphus.drivers.birkholz*), 224
 LennardJones (class in *pysisyphus.calculators*), 203
 LennardJones (class in *pysisyphus.calculators.LennardJones*), 79, 174
 LEPSBase (class in *pysisyphus.calculators.LEPSBase*), 173
 LEPSExpr (class in *pysisyphus.calculators.LEPSExpr*), 173
 lf_ind (*pysisyphus.cos.GrowingString.GrowingString* attribute), 113, 220
 lincs_closure() (in module *pysisyphus.dynamics.lincs*), 236
 lindh_guess() (in module *pysisyphus.optimizers.guess_hessians*), 298
 lindh_style_guess() (in module *pysisyphus.optimizers.guess_hessians*), 298
 line() (*pysisyphus.TableFormatter.TableFormatter* attribute), 317
 LINEAR_BEND (*pysisyphus.intcoords.PrimTypes.PrimTypes* attribute), 283

- [attribute](#)), 242
- [LINEAR_BEND_COMPLEMENT](#) (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- [linear_bend_complements](#) (*pysisyphus.intcoords.setup.CoordInfo attribute*), 254
- [linear_bend_indices](#) (*pysisyphus.intcoords.RedundantCoords property*), 261
- [linear_bend_indices](#) (*pysisyphus.intcoords.RedundantCoords.RedundantCoords property*), 44, 245
- [linear_bends](#) (*pysisyphus.intcoords.setup.CoordInfo attribute*), 254
- [LINEAR_DISPLACEMENT](#) (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- [LINEAR_DISPLACEMENT_COMPLEMENT](#) (*pysisyphus.intcoords.PrimTypes.PrimTypes attribute*), 242
- [LinearBend](#) (class in *pysisyphus.intcoords*), 259
- [LinearBend](#) (class in *pysisyphus.intcoords.LinearBend*), 241
- [LinearDisplacement](#) (class in *pysisyphus.intcoords*), 259
- [LinearDisplacement](#) (class in *pysisyphus.intcoords.LinearDisplacement*), 241
- [LineSearch](#) (class in *pysisyphus.line_searches.LineSearch*), 279
- [LineSearchConverged](#), 279
- [LineSearchNotConverged](#), 279
- [LineSearchResult](#) (class in *pysisyphus.line_searches.LineSearch*), 279
- [Link](#) (class in *pysisyphus.calculators.ONIOMv2*), 74, 176
- [list_h5_groups\(\)](#) (in module *pysisyphus.plot*), 328
- [listen_for\(\)](#) (*pysisyphus.calculators.IPIServer method*), 203
- [listen_for\(\)](#) (*pysisyphus.calculators.IPIServer.IPIServer method*), 172
- [listen_for\(\)](#) (*pysisyphus.calculators.SocketCalc.SocketCalc method*), 185
- [listen_for_client_atom_num\(\)](#) (*pysisyphus.calculators.IPIServer method*), 203
- [listen_for_client_atom_num\(\)](#) (*pysisyphus.calculators.IPIServer.IPIServer method*), 172
- [listen_for_energy\(\)](#) (*pysisyphus.calculators.IPIServer method*), 203
- [listen_for_energy\(\)](#) (*pysisyphus.calculators.IPIServer.IPIServer method*), 172
- [listen_for_forces\(\)](#) (*pysisyphus.calculators.IPIServer method*), 203
- [listen_for_forces\(\)](#) (*pysisyphus.calculators.IPIServer.IPIServer method*), 172
- [listen_for_hessian\(\)](#) (*pysisyphus.calculators.IPIServer method*), 203
- [listen_for_hessian\(\)](#) (*pysisyphus.calculators.IPIServer.IPIServer method*), 172
- [listen_kinds](#) (*pysisyphus.calculators.IPIServer attribute*), 203
- [listen_kinds](#) (*pysisyphus.calculators.IPIServer.IPIServer attribute*), 172
- [load_h5\(\)](#) (in module *pysisyphus.plot*), 329
- [load_run_dict\(\)](#) (in module *pysisyphus.run*), 331
- [log\(\)](#) (in module *pysisyphus.helpers_pure*), 321
- [log\(\)](#) (in module *pysisyphus.optimizers.gdiis*), 298
- [log\(\)](#) (in module *pysisyphus.wrapper.mwfn*), 309
- [log\(\)](#) (*pysisyphus.calculators.Calculator.Calculator method*), 49, 158
- [log\(\)](#) (*pysisyphus.calculators.ONIOMv2.Model method*), 75, 177
- [log\(\)](#) (*pysisyphus.calculators.WFOWrapper.WFOWrapper method*), 189
- [log\(\)](#) (*pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method*), 190
- [log\(\)](#) (*pysisyphus.cos.ChainOfStates.ChainOfStates method*), 109, 216
- [log\(\)](#) (*pysisyphus.cos.GrowingNT.GrowingNT method*), 219
- [log\(\)](#) (*pysisyphus.intcoords.Primitive.Primitive method*), 244
- [log\(\)](#) (*pysisyphus.intcoords.RedundantCoords method*), 261
- [log\(\)](#) (*pysisyphus.intcoords.RedundantCoords.RedundantCoords method*), 44, 245
- [log\(\)](#) (*pysisyphus.irc.IRC.IRC method*), 126, 271
- [log\(\)](#) (*pysisyphus.line_searches.LineSearch.LineSearch method*), 279
- [log\(\)](#) (*pysisyphus.optimizers.MicroOptimizer method*), 301
- [log\(\)](#) (*pysisyphus.optimizers.MicroOptimizer.MicroOptimizer method*), 289
- [log\(\)](#) (*pysisyphus.optimizers.Optimizer.Optimizer method*), 86, 292
- [log\(\)](#) (*pysisyphus.stochastic.Pipeline.Pipeline method*), 303
- [log_dbg\(\)](#) (*pysisyphus.intcoords.Primitive.Primitive method*), 244
- [log_fragments\(\)](#) (*pysisyphus.calculators.AFIR method*), 194
- [log_fragments\(\)](#) (*pysisyphus.calculators.AFIR.AFIR*

method), 73, 154

log_int_grad_msg() (pysisyphus.intcoords.RedundantCoords method), 261

log_int_grad_msg() (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 245

log_negative_eigenvalues() (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer method), 88, 287

log_progress() (in module pysisyphus.irc.Instanton), 273

LogFermi (class in pysisyphus.calculators.ExternalPotential), 168

logger (pysisyphus.calculators.WFOWrapper.WFOWrapper attribute), 189

logger (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 attribute), 190

logger (pysisyphus.cos.ChainOfStates.ChainOfStates attribute), 109, 216

logger (pysisyphus.cos.GrowingNT.GrowingNT attribute), 219

LQA (class in pysisyphus.irc), 277

LQA (class in pysisyphus.irc.LQA), 128, 273

LST (class in pysisyphus.interpolate.LST), 263

lstsq_with_reference() (in module pysisyphus.drivers.afir), 223

M

m_sqrt (pysisyphus.irc.IRC.IRC property), 126, 271

main() (in module pysisyphus.run), 331

make_cdd() (in module pysisyphus.wrapper.mwfn), 309

make_conv_dict() (pysisyphus.optimizers.Optimizer.Optimizer method), 86, 292

make_deriv_funcs() (in module pysisyphus.intcoords.generate_derivatives), 251

make_det_string() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189

make_det_string() (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190

make_dets_header() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189

make_dets_header() (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190

make_exc_str() (pysisyphus.calculators.Gaussian16 method), 201

make_exc_str() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170

make_fchk() (pysisyphus.calculators.Gaussian16 method), 201

make_fchk() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170

make_float_class() (in module pysisyphus.calculators.parser), 192

make_fn() (pysisyphus.calculators.Calculator.Calculator method), 50, 158

make_full_dets_list() (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189

make_full_dets_list() (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190

make_gbs_str() (pysisyphus.calculators.Gaussian16 method), 201

make_gbs_str() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170

make_input() (in module pysisyphus.wrapper.packmol), 309

make_molden() (pysisyphus.calculators.Turbomole static method), 211

make_molden() (pysisyphus.calculators.Turbomole.Turbomole static method), 62, 187

make_py_func() (in module pysisyphus.intcoords.generate_derivatives), 251

make_sym_mat() (in module pysisyphus.calculators.ORCA), 60, 180

make_trj_str() (in module pysisyphus.xyzloader), 333

make_trj_str_from_geoms() (in module pysisyphus.xyzloader), 333

make_unit_vec() (in module pysisyphus.linalg), 323

make_xyz_str() (in module pysisyphus.xyzloader), 333

mass_weigh_hessian() (pysisyphus.Geometry.Geometry method), 40, 314

mass_weigh_hessian() (pysisyphus.irc.IRC.IRC method), 126, 271

masses (pysisyphus.Geometry.Geometry property), 40, 314

masses (pysisyphus.intcoords.CartesianCoords property), 257

masses (pysisyphus.intcoords.CartesianCoords.CartesianCoords property), 238

masses_rep (pysisyphus.cos.ChainOfStates.ChainOfStates property), 109, 216

masses_rep (pysisyphus.Geometry.Geometry property), 40, 315

masses_sqrt (pysisyphus.intcoords.CartesianCoords property), 257

masses_sqrt (pysisyphus.intcoords.CartesianCoords.CartesianCoords property), 238

match() (in module pysisyphus.trj), 332

`match_geom_atoms()` (in module `pysisyphus.stochastic.align`), 304
`match_geoms()` (in module `pysisyphus.helpers`), 319
`matched_rmsd()` (in module `pysisyphus.stochastic.align`), 304
`matrix_power()` (in module `pysisyphus.linalg`), 323
`matrix_types` (`pysisyphus.calculators.WFOWrapper.WFOWrapper` attribute), 189
`matrix_types` (`pysisyphus.calculators.WFOWrapper2.WFOWrapper2` attribute), 190
`max_ang_moms` (`pysisyphus.calculators.DFTBp` attribute), 197
`max_ang_moms` (`pysisyphus.calculators.DFTBp.DFTBp` attribute), 63, 163
`max_force_converged` (`pysisyphus.optimizers.Optimizer.ConvInfo` attribute), 84, 290
`max_image_num` (`pysisyphus.cos.ChainOfStates.ChainOfStates` property), 109, 216
`max_image_num` (`pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates` property), 112, 218
`max_step_converged` (`pysisyphus.optimizers.Optimizer.ConvInfo` attribute), 84, 290
`md()` (in module `pysisyphus.dynamics.driver`), 233
`mdp()` (in module `pysisyphus.dynamics.mdp`), 236
`mdp_result` (`pysisyphus.run.RunResult` attribute), 330
`MDPResult` (in module `pysisyphus.dynamics.mdp`), 236
`MDResult` (class in `pysisyphus.dynamics.driver`), 232
`merge_coordinate_definitions()` (in module `pysisyphus.intcoords.helpers`), 252
`merge_geoms()` (in module `pysisyphus.drivers.merge`), 224
`merge_opt()` (in module `pysisyphus.drivers.merge`), 224
`merge_sets()` (in module `pysisyphus.helpers_pure`), 321
`merge_with_frozen_geom()` (in module `pysisyphus.drivers.merge`), 224
`METHODS` (`pysisyphus.calculators.MOPAC` attribute), 203
`METHODS` (`pysisyphus.calculators.MOPAC.MOPAC` attribute), 64, 174
`micro_step()` (`pysisyphus.irc.GonzalezSchlegel` method), 276
`micro_step()` (`pysisyphus.irc.GonzalezSchlegel.GonzalezSchlegel` method), 127, 269
`MicroOptimizer` (class in `pysisyphus.optimizers`), 301
`MicroOptimizer` (class in `pysisyphus.optimizers.MicroOptimizer`), 289
`min_width_fmts()` (`pysisyphus.TableFormatter.TableFormatter` method), 317
`mm_inv` (`pysisyphus.Geometry.Geometry` property), 40, 315
`mm_sqrt_inv` (`pysisyphus.Geometry.Geometry` property), 40, 315
`mod_flowchart_update()` (in module `pysisyphus.optimizers.hessian_updates`), 299
`mode_inds` (`pysisyphus.modefollow.davidson.DavidsonResult` attribute), 282
`ModeKill` (class in `pysisyphus.irc`), 277
`ModeKill` (class in `pysisyphus.irc.ModeKill`), 274
`Model` (class in `pysisyphus.calculators.ONIOMv2`), 74, 176
`model_iter` (`pysisyphus.calculators.ONIOM` property), 205
`model_iter` (`pysisyphus.calculators.ONIOMv2.ONIOM` property), 76, 178
`model_opt` (`pysisyphus.optimizers.LayerOpt.LayerOpt` property), 96, 288
`ModelDummyCalc` (class in `pysisyphus.calculators.ONIOMv2`), 75, 177
`modes3d()` (`pysisyphus.Geometry.Geometry` method), 40, 315
`modified_broyden_closure()` (in module `pysisyphus.optimizers.closures`), 297
module
 `pysisyphus`, 334
 `pysisyphus.benchmarks`, 152
 `pysisyphus.benchmarks.Benchmark`, 151
 `pysisyphus.benchmarks.data`, 151
 `pysisyphus.calculators`, 193
 `pysisyphus.calculators.AFIR`, 72, 152
 `pysisyphus.calculators.AnaPot`, 154
 `pysisyphus.calculators.AnaPot2`, 154
 `pysisyphus.calculators.AnaPot3`, 155
 `pysisyphus.calculators.AnaPot4`, 155
 `pysisyphus.calculators.AnaPotBase`, 78, 155
 `pysisyphus.calculators.AnaPotCBM`, 156
 `pysisyphus.calculators.AtomAtomTransTorque`, 156
 `pysisyphus.calculators.Calculator`, 48, 156
 `pysisyphus.calculators.CerjanMiller`, 162
 `pysisyphus.calculators.CFOUR`, 68
 `pysisyphus.calculators.Composite`, 162
 `pysisyphus.calculators.ConicalIntersection`, 162
 `pysisyphus.calculators.Dalton`, 68, 164
 `pysisyphus.calculators.DFTBp`, 63, 163
 `pysisyphus.calculators.DFTD3`, 71
 `pysisyphus.calculators.Dimer`, 76, 121, 164
 `pysisyphus.calculators.Dummy`, 166
 `pysisyphus.calculators.EGO`, 166
 `pysisyphus.calculators.EnergyMin`, 167

pysisyphus.calculators.ExternalPotential, 168

pysisyphus.calculators.ExternalPotential.ExternalPotentialAdos.ChainOfStates, 108, 215

pysisyphus.calculators.ExternalPotential.HarmonicSphere, 71

pysisyphus.calculators.ExternalPotential.LogFermi, 71

pysisyphus.calculators.ExternalPotential.Restricted, 71

pysisyphus.calculators.FakeASE, 79, 169

pysisyphus.calculators.FourWellAnaPot, 169

pysisyphus.calculators.FreeEndNEBPot, 170

pysisyphus.calculators.Gaussian09, 55, 170

pysisyphus.calculators.Gaussian16, 56, 170

pysisyphus.calculators.HardSphere, 171

pysisyphus.calculators.IDPPCalculator, 172

pysisyphus.calculators.IPIClient, 172

pysisyphus.calculators.IPIServer, 172

pysisyphus.calculators.LennardJones, 79, 174

pysisyphus.calculators.LEPSBase, 173

pysisyphus.calculators.LEPSExpr, 173

pysisyphus.calculators.MOPAC, 64, 174

pysisyphus.calculators.MullerBrownSymPyPot, 175

pysisyphus.calculators.MultiCalc, 175

pysisyphus.calculators.OBabel, 68, 175

pysisyphus.calculators.ONIOMv2, 74, 176

pysisyphus.calculators.OpenMM, 180

pysisyphus.calculators.OpenMolcas, 58, 181

pysisyphus.calculators.ORCA, 59, 178

pysisyphus.calculators.ORCA5, 180

pysisyphus.calculators.OverlapCalculator, 53, 182

pysisyphus.calculators.parser, 192

pysisyphus.calculators.Psi4, 65, 184

pysisyphus.calculators.PyPsi4, 184

pysisyphus.calculators.PyXTB, 184

pysisyphus.calculators.Rastrigin, 185

pysisyphus.calculators.Remote, 185

pysisyphus.calculators.Rosenbrock, 185

pysisyphus.calculators.SocketCalc, 185

pysisyphus.calculators.TIP3P, 79, 186

pysisyphus.calculators.TransTorque, 186

pysisyphus.calculators.Turbomole, 61, 187

pysisyphus.calculators.WFOWrapper, 189

pysisyphus.calculators.WFOWrapper2, 189

pysisyphus.calculators.XTB, 66, 190

pysisyphus.color, 318

pysisyphus.config, 318

pysisyphus.constants, 318

pysisyphus.cos, 221

pysisyphus.cos.AdaptiveNEB, 111, 214

pysisyphus.cos.ChainOfStates, 108, 215

pysisyphus.cos.FreeEndNEB, 112, 217

pysisyphus.cos.FreezingString, 217

pysisyphus.cos.GrowingChainOfStates, 112, 218

pysisyphus.cos.GrowingNT, 218

pysisyphus.cos.GrowingString, 113, 219

pysisyphus.cos.NEB, 110, 220

pysisyphus.cos.SimpleZTS, 112, 221

pysisyphus.db, 222

pysisyphus.db.generate_db, 221

pysisyphus.db.helpers, 221

pysisyphus.db.level, 221

pysisyphus.db.molecules, 221

pysisyphus.drivers, 231

pysisyphus.drivers.afir, 222

pysisyphus.drivers.barriers, 224

pysisyphus.drivers.birkholz, 224

pysisyphus.drivers.merge, 224

pysisyphus.drivers.opt, 225

pysisyphus.drivers.perf, 226

pysisyphus.drivers.pka, 226

pysisyphus.drivers.precon_pos_rot, 226

pysisyphus.drivers.rates, 227

pysisyphus.drivers.replace, 230

pysisyphus.drivers.scan, 231

pysisyphus.drivers.thermo, 231

pysisyphus.dynamics, 237

pysisyphus.dynamics.colvars, 232

pysisyphus.dynamics.driver, 232

pysisyphus.dynamics.Gaussian, 231

pysisyphus.dynamics.helpers, 233

pysisyphus.dynamics.lincs, 236

pysisyphus.dynamics.mdp, 236

pysisyphus.dynamics.rattle, 236

pysisyphus.dynamics.thermostats, 237

pysisyphus.elem_data, 318

pysisyphus.exceptions, 318

pysisyphus.filtertrj, 318

pysisyphus.Geometry, 35, 309

pysisyphus.helpers, 318

pysisyphus.helpers_pure, 320

pysisyphus.init_logging, 322

pysisyphus.intcoords, 257

pysisyphus.intcoords.augment_bonds, 248

pysisyphus.intcoords.Bend, 237

pysisyphus.intcoords.Bend2, 237

pysisyphus.intcoords.BondedFragment, 238

pysisyphus.intcoords.Cartesian, 238

pysisyphus.intcoords.CartesianCoords, 238

pysisyphus.intcoords.Coords, 239

pysisyphus.intcoords.derivatives, 248

pysisyphus.intcoords.DistanceFunction, 241

pysisyphus.intcoords.DLC, 240

pysisyphus.intcoords.DummyTorsion, 241

pysisyphus.intcoords.eval, 249

pysisyphus.intcoords.exceptions, 250

pysisyphus.intcoords.findiffs, 250

pysisyphus.intcoords.generate_derivatives, 250

pysisyphus.intcoords.helpers, 251

pysisyphus.intcoords.LinearBend, 241

pysisyphus.intcoords.LinearDisplacement, 241

pysisyphus.intcoords.mp_derivatives, 252

pysisyphus.intcoords.OutOfPlane, 242

pysisyphus.intcoords.Primitive, 244

pysisyphus.intcoords.PrimTypes, 242

pysisyphus.intcoords.RedundantCoords, 42, 244

pysisyphus.intcoords.Rotation, 246

pysisyphus.intcoords.setup, 253

pysisyphus.intcoords.setup_fast, 256

pysisyphus.intcoords.Stretch, 247

pysisyphus.intcoords.Torsion, 247

pysisyphus.intcoords.Torsion2, 247

pysisyphus.intcoords.Translation, 247

pysisyphus.intcoords.update, 256

pysisyphus.intcoords.valid, 257

pysisyphus.interpolate, 264

pysisyphus.interpolate.helpers, 264

pysisyphus.interpolate.IDPP, 263

pysisyphus.interpolate.Interpolator, 263

pysisyphus.interpolate.LST, 263

pysisyphus.interpolate.Redund, 263

pysisyphus.io, 267

pysisyphus.io.cjson, 264

pysisyphus.io.crd, 264

pysisyphus.io.hdf5, 264

pysisyphus.io.hessian, 265

pysisyphus.io.mol2, 265

pysisyphus.io.molden, 265

pysisyphus.io.pdb, 265

pysisyphus.io.pubchem, 265

pysisyphus.io.sdf, 266

pysisyphus.io.xyz, 266

pysisyphus.io.zmat, 266

pysisyphus.irc, 276

pysisyphus.irc.DampedVelocityVerlet, 126, 268

pysisyphus.irc.DWI, 267

pysisyphus.irc.Euler, 127, 268

pysisyphus.irc.EulerPC, 127, 268

pysisyphus.irc.GonzalezSchlegel, 127, 269

pysisyphus.irc.IMKMod, 128, 269

pysisyphus.irc.initial_displ, 275

pysisyphus.irc.Instanton, 272

pysisyphus.irc.IRC, 124, 269

pysisyphus.irc.IRCDummy, 272

pysisyphus.irc.LQA, 128, 273

pysisyphus.irc.ModeKill, 274

pysisyphus.irc.ParamPlot, 274

pysisyphus.irc.RK4, 128, 274

pysisyphus.linalg, 323

pysisyphus.line_searches, 280

pysisyphus.line_searches.Backtracking, 278

pysisyphus.line_searches.HagerZhang, 278

pysisyphus.line_searches.interpol, 280

pysisyphus.line_searches.LineSearch, 279

pysisyphus.line_searches.StrongWolfe, 280

pysisyphus.modelfollow, 283

pysisyphus.modelfollow.davidson, 282

pysisyphus.modelfollow.lanczos, 283

pysisyphus.modelfollow.NormalMode, 282

pysisyphus.optimizers, 301

pysisyphus.optimizers.BacktrackingOptimizer, 284

pysisyphus.optimizers.BFGS, 284

pysisyphus.optimizers.closures, 296

pysisyphus.optimizers.cls_map, 297

pysisyphus.optimizers.ConjugateGradient, 285

pysisyphus.optimizers.CubicNewton, 285

pysisyphus.optimizers.exceptions, 297

pysisyphus.optimizers.FIRE, 285

pysisyphus.optimizers.gdiis, 297

pysisyphus.optimizers.guess_hessians, 298

pysisyphus.optimizers.hessian_updates, 299

pysisyphus.optimizers.HessianOptimizer, 87, 285

pysisyphus.optimizers.LayerOpt, 96, 288

pysisyphus.optimizers.LBFGS, 96, 287

pysisyphus.optimizers.MicroOptimizer, 289

pysisyphus.optimizers.NCOptimizer, 290

pysisyphus.optimizers.Optimizer, 84, 290

pysisyphus.optimizers.poly_fit, 300

pysisyphus.optimizers.precon, 301

pysisyphus.optimizers.PreconLBFGS, 97, 293

pysisyphus.optimizers.PreconSteepestDescent, 294

pysisyphus.optimizers.QuickMin, 294

pysisyphus.optimizers.restrict_step, 301

pysisyphus.optimizers.RFOptimizer, 89, 294

pysisyphus.optimizers.RSA, 295

pysisyphus.optimizers.StabilizedQNMethod, 295

pysisyphus.optimizers.SteepestDescent, 296
 pysisyphus.optimizers.StringOptimizer, 296
 pysisyphus.pack, 325
 pysisyphus.peakdetect, 325
 pysisyphus.plot, 328
 pysisyphus.plotters, 302
 pysisyphus.plotters.AnimPlot, 302
 pysisyphus.run, 329
 pysisyphus.socket_helper, 331
 pysisyphus.stochastic, 304
 pysisyphus.stochastic.align, 304
 pysisyphus.stochastic.FragmentKick, 302
 pysisyphus.stochastic.Kick, 303
 pysisyphus.stochastic.Pipeline, 303
 pysisyphus.TableFormatter, 317
 pysisyphus.TablePrinter, 317
 pysisyphus.testing, 332
 pysisyphus.tests, 305
 pysisyphus.thermo, 332
 pysisyphus.trj, 332
 pysisyphus.tsoptimizers, 308
 pysisyphus.tsoptimizers.RSIRFOptimizer, 120, 305
 pysisyphus.tsoptimizers.RSPRFOptimizer, 120, 305
 pysisyphus.tsoptimizers.TRIM, 120, 306
 pysisyphus.tsoptimizers.TSHessianOptimizer, 118, 306
 pysisyphus.version, 333
 pysisyphus.wrapper, 309
 pysisyphus.wrapper.jmol, 308
 pysisyphus.wrapper.mwfn, 309
 pysisyphus.wrapper.packmol, 309
 pysisyphus.xyzloader, 333
 pysisyphus.yaml_mods, 334
 module_available() (in module pysisyphus.testing), 332
 molecular_volume() (in module pysisyphus.helpers_pure), 321
 Molecule (class in pysisyphus.db.molecules), 221
 MOPAC (class in pysisyphus.calculators), 203
 MOPAC (class in pysisyphus.calculators.MOPAC), 64, 174
 moving_atoms (pysisyphus.Geometry.Geometry property), 40, 315
 moving_atoms_jmol() (pysisyphus.Geometry.Geometry method), 40, 315
 moving_images (pysisyphus.cos.ChainOfStates.ChainOfStates property), 109, 216
 moving_indices (pysisyphus.cos.ChainOfStates.ChainOfStates property), 109, 216
 MullerBrownPot (class in pysisyphus.calculators.MullerBrownSympyPot), 175
 mult (pysisyphus.calculators.AFIR property), 194
 mult (pysisyphus.calculators.AFIR.AFIR property), 73, 154
 mult (pysisyphus.calculators.ONIOM property), 205
 mult (pysisyphus.calculators.ONIOMv2.LayerCalc property), 74, 176
 mult (pysisyphus.calculators.ONIOMv2.ONIOM property), 76, 178
 mult (pysisyphus.db.molecules.Molecule attribute), 221
 mult (pysisyphus.drivers.afir.AFIRPath attribute), 222
 MULT_STRS (pysisyphus.calculators.MOPAC attribute), 203
 MULT_STRS (pysisyphus.calculators.MOPAC.MOPAC attribute), 64, 174
 multi_component_sym_mat() (in module pysisyphus.linalg), 323
 MultiCalc (class in pysisyphus.calculators), 204
 MultiCalc (class in pysisyphus.calculators.MultiCalc), 175
 mw_coords (pysisyphus.Geometry.Geometry property), 40, 315
 mw_coords (pysisyphus.irc.IRC.IRC property), 126, 271
 mw_grad_to_acc() (pysisyphus.irc.DampedVelocityVerlet method), 276
 mw_grad_to_acc() (pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet method), 126, 268
 mw_gradient (pysisyphus.Geometry.Geometry property), 40, 315
 mw_gradient (pysisyphus.irc.IRC.IRC property), 126, 271
 mw_hessian (pysisyphus.Geometry.Geometry property), 41, 315
 mw_norm_for_norm() (pysisyphus.modefollow.NormalMode method), 283
 mw_norm_for_norm() (pysisyphus.modefollow.NormalMode.NormalMode method), 282
 MWCartesianCoords (class in pysisyphus.intcoords), 259
 MWCartesianCoords (class in pysisyphus.intcoords.CartesianCoords), 239

N

N (pysisyphus.calculators.Dimer property), 198
 N (pysisyphus.calculators.Dimer.Dimer property), 76, 121, 164
 N (pysisyphus.optimizers.gdiis.DIISResult attribute), 297

n_hist (*pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod* property), 295
name (*pysisyphus.calculators.Calculator.Calculator* property), 50, 158
name (*pysisyphus.calculators.Calculator.SetPlan* attribute), 53, 161
name (*pysisyphus.db.molecules.Molecule* attribute), 221
NCOptimizer (class in *pysisyphus.optimizers.NCOptimizer*), 290
NEB (class in *pysisyphus.cos.NEB*), 110, 220
NeedNewInternalsException, 250
neg_eigvals (*pysisyphus.helpers.FinalHessianResult* attribute), 319
new_node_coords() (*pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates* method), 112, 218
nodes_missing (*pysisyphus.cos.GrowingString.GrowingString* property), 113, 220
NONE (*pysisyphus.calculators.Calculator.KeepKind* attribute), 53, 161
norm3() (in module *pysisyphus.linalg*), 323
norm_inf() (*pysisyphus.line_searches.HagerZhang* method), 281
norm_inf() (*pysisyphus.line_searches.HagerZhang.HagerZhang* method), 278
norm_max_rms() (in module *pysisyphus.helpers*), 319
normalize_prim_input() (in module *pysisyphus.intcoords.PrimTypes*), 243
normalize_prim_inputs() (in module *pysisyphus.intcoords.PrimTypes*), 243
normalize_replacements() (in module *pysisyphus.drivers.replace*), 230
NormalMode (class in *pysisyphus.modefollow*), 283
NormalMode (class in *pysisyphus.modefollow.NormalMode*), 282
np_print() (in module *pysisyphus.helpers*), 320
NTOs (class in *pysisyphus.calculators.OverlapCalculator*), 53, 182
ntos (*pysisyphus.calculators.OverlapCalculator.NTOs* attribute), 53, 182
nuc_charges_for_atoms() (in module *pysisyphus.elem_data*), 318
NUMERICAL (*pysisyphus.calculators.Calculator.HessKind* attribute), 53, 161
nus (*pysisyphus.helpers.FinalHessianResult* attribute), 319
nus (*pysisyphus.modefollow.davidson.DavidsonResult* attribute), 282
O
OBabel (class in *pysisyphus.calculators*), 204
OBabel (class in *pysisyphus.calculators.OBabel*), 68, 175
onKeyPress() (*pysisyphus.plotters.AnimPlot.AnimPlot* method), 302
ONIOM (class in *pysisyphus.calculators*), 205
ONIOM (class in *pysisyphus.calculators.ONIOMv2*), 75, 177
OpenMM (class in *pysisyphus.calculators.OpenMM*), 180
OpenMolcas (class in *pysisyphus.calculators*), 208
OpenMolcas (class in *pysisyphus.calculators.OpenMolcas*), 58, 181
opt (*pysisyphus.drivers.opt.OptResult* attribute), 225
opt (*pysisyphus.run.RunResult* attribute), 330
opt_afir_path() (in module *pysisyphus.drivers.afir*), 223
opt_davidson() (in module *pysisyphus.drivers.opt*), 226
opt_geom (*pysisyphus.calculators.XTB.OptResult* attribute), 66, 190
opt_geom (*pysisyphus.run.RunResult* attribute), 330
opt_is_converged (*pysisyphus.drivers.afir.AFIRPath* attribute), 222
opt_log (*pysisyphus.calculators.XTB.OptResult* attribute), 66, 190
OptimizationError, 297
optimize() (*pysisyphus.optimizers.BFGS.BFGS* method), 284
optimize() (*pysisyphus.optimizers.ConjugateGradient.ConjugateGradient* method), 285
optimize() (*pysisyphus.optimizers.CubicNewton* method), 301
optimize() (*pysisyphus.optimizers.CubicNewton.CubicNewton* method), 285
optimize() (*pysisyphus.optimizers.FIRE.FIRE* method), 285
optimize() (*pysisyphus.optimizers.LayerOpt.LayerOpt* method), 96, 288
optimize() (*pysisyphus.optimizers.LBFGS.LBFGS* method), 97, 288
optimize() (*pysisyphus.optimizers.MicroOptimizer* method), 301
optimize() (*pysisyphus.optimizers.MicroOptimizer.MicroOptimizer* method), 289
optimize() (*pysisyphus.optimizers.NCOptimizer.NCOptimizer* method), 290
optimize() (*pysisyphus.optimizers.Optimizer.Optimizer* method), 86, 292
optimize() (*pysisyphus.optimizers.PreconLBFGS.PreconLBFGS* method), 98, 294
optimize() (*pysisyphus.optimizers.QuickMin.QuickMin* method), 294
optimize() (*pysisyphus.optimizers.RFOptimizer.RFOptimizer* method), 89, 295
optimize() (*pysisyphus.optimizers.RSA.RSA* method), 295
optimize() (*pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod* method), 295

- method), 295
- optimize() (pysisyphus.optimizers.SteepestDescent.SteepestDescent method), 296
- optimize() (pysisyphus.optimizers.StringOptimizer.StringOptimizer method), 296
- optimize() (pysisyphus.tsoptimizers.RSIRFOptimizer.RSIRFOptimizer method), 308
- optimize() (pysisyphus.tsoptimizers.RSIRFOptimizer.RSIRFOptimizer method), 120, 305
- optimize() (pysisyphus.tsoptimizers.RSPRFOptimizer.RSPRFOptimizer method), 308
- optimize() (pysisyphus.tsoptimizers.RSPRFOptimizer.RSPRFOptimizer method), 120, 305
- optimize() (pysisyphus.tsoptimizers.TRIM method), 308
- optimize() (pysisyphus.tsoptimizers.TRIM.TRIM method), 120, 306
- Optimizer (class in pysisyphus.optimizers.Optimizer), 84, 290
- OptResult (class in pysisyphus.calculators.XTB), 66, 190
- OptResult (class in pysisyphus.drivers.opt), 225
- ORCA (class in pysisyphus.calculators), 206
- ORCA (class in pysisyphus.calculators.ORCA), 59, 178
- ORCA5 (class in pysisyphus.calculators), 207
- ORCA5 (class in pysisyphus.calculators.ORCA5), 180
- OrderedEnum (class in pysisyphus.helpers_pure), 320
- ORG (pysisyphus.calculators.Calculator.HessKind attribute), 53, 161
- origin() (in module pysisyphus.trj), 332
- orthogonalize_against() (in module pysisyphus.linalg), 323
- OUT_OF_PLANE (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 242
- OutOfPlane (class in pysisyphus.intcoords), 259
- OutOfPlane (class in pysisyphus.intcoords.OutOfPlane), 242
- outofplane_indices (pysisyphus.intcoords.RedundantCoords property), 261
- outofplane_indices (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 245
- OverlapCalculator (class in pysisyphus.calculators.OverlapCalculator), 53, 182
- OVL_P_TYPE_VERBOSE (pysisyphus.calculators.OverlapCalculator.OverlapCalculator attribute), 54, 182
- P
- P (pysisyphus.calculators.ConicalIntersection.CIQuantities attribute), 162
- P (pysisyphus.cos.GrowingNT.GrowingNT property), 218
- P (pysisyphus.intcoords.RedundantCoords property), 260
- P (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 43, 245
- P (pysisyphus.irc.Instanton.Instanton property), 272
- P_bh (pysisyphus.irc.Instanton.Instanton attribute), 272
- par_image_calc() (pysisyphus.cos.ChainOfStates.ChainOfStates method), 109, 216
- parallel() (pysisyphus.intcoords.Primitive.Primitive static method), 244
- ParallelForces (pysisyphus.cos.NEB.NEB property), 111, 220
- ParamPlot (class in pysisyphus.irc.ParamPlot), 274
- parent_ind (pysisyphus.calculators.ONIOMv2.Link attribute), 74, 176
- parse_635r_dump() (pysisyphus.calculators.Gaussian16 method), 202
- parse_635r_dump() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170
- parse_all_energies() (pysisyphus.calculators.DFTBp method), 197
- parse_all_energies() (pysisyphus.calculators.DFTBp.DFTBp method), 63, 163
- parse_all_energies() (pysisyphus.calculators.Gaussian16 method), 202
- parse_all_energies() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170
- parse_all_energies() (pysisyphus.calculators.ORCA method), 207
- parse_all_energies() (pysisyphus.calculators.ORCA.ORCA method), 59, 179
- parse_all_energies() (pysisyphus.calculators.Turbomole method), 211
- parse_all_energies() (pysisyphus.calculators.Turbomole.Turbomole method), 62, 187
- parse_args() (in module pysisyphus.config), 318
- parse_args() (in module pysisyphus.db.generate_db), 221
- parse_args() (in module pysisyphus.drivers.merge), 225
- parse_args() (in module pysisyphus.drivers.replace), 230
- parse_args() (in module pysisyphus.drivers.thermo), 231
- parse_args() (in module pysisyphus.filtertrj), 318
- parse_args() (in module pysisyphus.pack), 325
- parse_args() (in module pysisyphus.plot), 329
- parse_args() (in module pysisyphus.run), 331

- [parse_args\(\)](#) (in module *pysisyphus.trj*), 332
[parse_atom_name\(\)](#) (in module *pysisyphus.io.pdb*), 265
[parse_atoms_coords\(\)](#) (*pysisyphus.calculators.ORCA* static method), 207
[parse_atoms_coords\(\)](#) (*pysisyphus.calculators.ORCA.ORCA* static method), 59, 179
[parse_cc2_vectors\(\)](#) (*pysisyphus.calculators.Turbomole* method), 211
[parse_cc2_vectors\(\)](#) (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
[parse_cfour_energy\(\)](#) (in module *pysisyphus.calculators.CFOUR*), 70
[parse_cfour_gradient\(\)](#) (in module *pysisyphus.calculators.CFOUR*), 70
[parse_charges\(\)](#) (*pysisyphus.calculators.Gaussian16* method), 202
[parse_charges\(\)](#) (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 56, 170
[parse_charges\(\)](#) (*pysisyphus.calculators.ONIOMv2.Model* method), 75, 177
[parse_charges\(\)](#) (*pysisyphus.calculators.XTB* method), 213
[parse_charges\(\)](#) (*pysisyphus.calculators.XTB.XTB* method), 67, 191
[parse_charges_from_json\(\)](#) (*pysisyphus.calculators.XTB* method), 213
[parse_charges_from_json\(\)](#) (*pysisyphus.calculators.XTB.XTB* method), 67, 191
[parse_ci_coeffs\(\)](#) (*pysisyphus.calculators.Turbomole* method), 211
[parse_ci_coeffs\(\)](#) (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
[parse_cis\(\)](#) (*pysisyphus.calculators.ORCA* static method), 207
[parse_cis\(\)](#) (*pysisyphus.calculators.ORCA.ORCA* static method), 60, 179
[parse_cjson\(\)](#) (in module *pysisyphus.io.cjson*), 264
[parse_coord_line\(\)](#) (in module *pysisyphus.io.sdf*), 266
[parse_double_mol\(\)](#) (*pysisyphus.calculators.Gaussian16* method), 202
[parse_double_mol\(\)](#) (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 56, 170
[parse_double_mol\(\)](#) (*pysisyphus.calculators.Turbomole* method), 211
[parse_double_mol\(\)](#) (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
[parse_energies\(\)](#) (*pysisyphus.calculators.OpenMolcas* method), 208
[parse_energies\(\)](#) (*pysisyphus.calculators.OpenMolcas.OpenMolcas* method), 58, 181
[parse_energy\(\)](#) (*pysisyphus.calculators.CFOUR* method), 195
[parse_energy\(\)](#) (*pysisyphus.calculators.CFOUR.CFOUR* method), 69
[parse_energy\(\)](#) (*pysisyphus.calculators.DFTBp* method), 197
[parse_energy\(\)](#) (*pysisyphus.calculators.DFTBp.DFTBp* method), 63, 163
[parse_energy\(\)](#) (*pysisyphus.calculators.DFTD3.DFTD3* method), 71
[parse_energy\(\)](#) (*pysisyphus.calculators.MOPAC* method), 204
[parse_energy\(\)](#) (*pysisyphus.calculators.MOPAC.MOPAC* method), 65, 174
[parse_energy\(\)](#) (*pysisyphus.calculators.OpenMolcas* method), 208
[parse_energy\(\)](#) (*pysisyphus.calculators.OpenMolcas.OpenMolcas* method), 58, 181
[parse_energy\(\)](#) (*pysisyphus.calculators.ORCA* method), 207
[parse_energy\(\)](#) (*pysisyphus.calculators.ORCA.ORCA* method), 60, 179
[parse_energy\(\)](#) (*pysisyphus.calculators.Psi4* method), 209
[parse_energy\(\)](#) (*pysisyphus.calculators.Psi4.Psi4* method), 65, 184
[parse_energy\(\)](#) (*pysisyphus.calculators.Turbomole* method), 211
[parse_energy\(\)](#) (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
[parse_energy\(\)](#) (*pysisyphus.calculators.XTB* method), 213
[parse_energy\(\)](#) (*pysisyphus.calculators.XTB.XTB* method), 67, 191
[parse_energy_from_aux\(\)](#) (*pysisyphus.calculators.MOPAC* static method), 204
[parse_energy_from_aux\(\)](#) (*pysisyphus.calculators.MOPAC.MOPAC* static method), 65, 174
[parse_engrad\(\)](#) (*pysisyphus.calculators.ORCA* method), 207

`parse_engrad()` (*pysisyphus.calculators.ORCA.ORCA* method), 60, 179
`parse_engrad_info()` (*pysisyphus.calculators.ORCA* static method), 207
`parse_engrad_info()` (*pysisyphus.calculators.ORCA.ORCA* static method), 60, 179
`parse_exc_dat()` (*pysisyphus.calculators.DFTBp* static method), 197
`parse_exc_dat()` (*pysisyphus.calculators.DFTBp.DFTBp* static method), 64, 163
`parse_fchk()` (*pysisyphus.calculators.Gaussian16* static method), 202
`parse_fchk()` (*pysisyphus.calculators.Gaussian16.Gaussian16* static method), 56, 170
`parse_filter()` (in module *pysisyphus.filtertrj*), 318
`parse_force()` (*pysisyphus.calculators.Gaussian16* method), 202
`parse_force()` (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 56, 170
`parse_force()` (*pysisyphus.calculators.Turbomole* method), 211
`parse_force()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
`parse_forces()` (*pysisyphus.calculators.DFTBp* method), 197
`parse_forces()` (*pysisyphus.calculators.DFTBp.DFTBp* method), 64, 164
`parse_gbw()` (*pysisyphus.calculators.ORCA* static method), 207
`parse_gbw()` (*pysisyphus.calculators.ORCA.ORCA* static method), 60, 179
`parse_grad()` (*pysisyphus.calculators.MOPAC* method), 204
`parse_grad()` (*pysisyphus.calculators.MOPAC.MOPAC* method), 65, 174
`parse_grad()` (*pysisyphus.calculators.Psi4* method), 209
`parse_grad()` (*pysisyphus.calculators.Psi4.Psi4* method), 66, 184
`parse_gradient()` (*pysisyphus.calculators.CFOUR* method), 195
`parse_gradient()` (*pysisyphus.calculators.CFOUR.CFOUR* method), 69
`parse_gradient()` (*pysisyphus.calculators.DFTD3.DFTD3* method), 71
`parse_gradient()` (*pysisyphus.calculators.OpenMolcas* method), 208
`parse_gradient()` (*pysisyphus.calculators.OpenMolcas.OpenMolcas* method), 58, 181
`parse_gradient()` (*pysisyphus.calculators.XTB* method), 213
`parse_gradient()` (*pysisyphus.calculators.XTB.XTB* method), 67, 191
`parse_gs_energy()` (*pysisyphus.calculators.Turbomole* method), 211
`parse_gs_energy()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
`parse_hess_file()` (*pysisyphus.calculators.ORCA* static method), 207
`parse_hess_file()` (*pysisyphus.calculators.ORCA.ORCA* static method), 60, 179
`parse_hessian()` (*pysisyphus.calculators.Gaussian16* method), 202
`parse_hessian()` (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 56, 170
`parse_hessian()` (*pysisyphus.calculators.MOPAC* method), 204
`parse_hessian()` (*pysisyphus.calculators.MOPAC.MOPAC* method), 65, 174
`parse_hessian()` (*pysisyphus.calculators.ORCA* method), 207
`parse_hessian()` (*pysisyphus.calculators.ORCA.ORCA* method), 60, 179
`parse_hessian()` (*pysisyphus.calculators.Psi4* method), 209
`parse_hessian()` (*pysisyphus.calculators.Psi4.Psi4* method), 66, 184
`parse_hessian()` (*pysisyphus.calculators.Turbomole* method), 211
`parse_hessian()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 62, 187
`parse_hessian()` (*pysisyphus.calculators.XTB* method), 213
`parse_hessian()` (*pysisyphus.calculators.XTB.XTB* method), 67, 191
`parse_hessian_from_aux()` (*pysisyphus.calculators.MOPAC* static method), 204
`parse_hessian_from_aux()` (*pysisyphus.calculators.MOPAC.MOPAC* static method), 65, 174
`parse_keyword()` (*pysisyphus.calculators.Gaussian16* method), 202

`parse_keyword()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 170
`parse_log()` (pysisyphus.calculators.Gaussian16 method), 202
`parse_log()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 171
`parse_md()` (pysisyphus.calculators.XTB method), 213
`parse_md()` (pysisyphus.calculators.XTB.XTB method), 67, 191
`parse_mo()` (in module pysisyphus.calculators.DFTBp), 64, 164
`parse_mo()` (in module pysisyphus.io.molden), 265
`parse_mo_numbers()` (pysisyphus.calculators.ORCA method), 207
`parse_mo_numbers()` (pysisyphus.calculators.ORCA.ORCA method), 60, 179
`parse_molden_atoms()` (in module pysisyphus.io.molden), 265
`parse_mos()` (pysisyphus.calculators.Turbomole method), 211
`parse_mos()` (pysisyphus.calculators.Turbomole.Turbomole method), 62, 187
`parse_opt()` (pysisyphus.calculators.XTB method), 213
`parse_opt()` (pysisyphus.calculators.XTB.XTB method), 67, 191
`parse_orca_cis()` (in module pysisyphus.calculators.ORCA), 60, 180
`parse_orca_gbw()` (in module pysisyphus.calculators.ORCA), 61, 180
`parse_rassi_track()` (pysisyphus.calculators.OpenMolcas method), 208
`parse_rassi_track()` (pysisyphus.calculators.OpenMolcas.OpenMolcas method), 58, 181
`parse_raw_term_func()` (in module pysisyphus.dynamics.mdp), 236
`parse_raw_term_funcs()` (in module pysisyphus.dynamics.mdp), 236
`parse_results()` (pysisyphus.calculators.Remote method), 210
`parse_results()` (pysisyphus.calculators.Remote.Remote method), 185
`parse_sdf()` (in module pysisyphus.io.sdf), 266
`parse_stable()` (pysisyphus.calculators.Gaussian16 method), 202
`parse_stable()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 171
`parse_stable()` (pysisyphus.calculators.ORCA method), 207
`parse_stable()` (pysisyphus.calculators.ORCA.ORCA method), 60, 179
`parse_td_vectors()` (pysisyphus.calculators.Turbomole method), 211
`parse_td_vectors()` (pysisyphus.calculators.Turbomole.Turbomole method), 62, 187
`parse_tddft()` (pysisyphus.calculators.Gaussian16 method), 202
`parse_tddft()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 171
`parse_tddft_tden()` (pysisyphus.calculators.Turbomole static method), 211
`parse_tddft_tden()` (pysisyphus.calculators.Turbomole.Turbomole static method), 62, 188
`parse_topo()` (pysisyphus.calculators.XTB method), 213
`parse_topo()` (pysisyphus.calculators.XTB.XTB method), 67, 191
`parse_total_energy()` (pysisyphus.calculators.DFTBp method), 197
`parse_total_energy()` (pysisyphus.calculators.DFTBp.DFTBp method), 64, 164
`parse_trj_file()` (in module pysisyphus.xyzloader), 333
`parse_trj_str()` (in module pysisyphus.xyzloader), 333
`parse_turbo_ccre0_ascii()` (in module pysisyphus.calculators.parser), 192
`parse_turbo_exstates()` (in module pysisyphus.calculators.parser), 192
`parse_turbo_exstates_re()` (in module pysisyphus.calculators.parser), 192
`parse_turbo_gradient()` (in module pysisyphus.calculators.parser), 192
`parse_turbo_mos()` (in module pysisyphus.calculators.parser), 192
`parse_wfoverlap_out()` (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189
`parse_wfoverlap_out()` (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190
`parse_xplusy()` (in module pysisyphus.calculators.DFTBp), 64, 164
`parse_xyz_file()` (in module pysisyphus.xyzloader), 333
`parse_xyz_str()` (in module pysisyphus.xyzloader), 333
`path_indices` (pysisyphus.drivers.afir.AFIRPath attribute), 222

- `path_length` (*pysisyphus.irc.Instanton.Instanton* property), 273
- `peakdetect()` (in module *pysisyphus.peakdetect*), 325
- `peakdetect_fft()` (in module *pysisyphus.peakdetect*), 325
- `peakdetect_parabola()` (in module *pysisyphus.peakdetect*), 326
- `peakdetect_sine()` (in module *pysisyphus.peakdetect*), 326
- `peakdetect_sine_locked()` (in module *pysisyphus.peakdetect*), 327
- `peakdetect_spline()` (in module *pysisyphus.peakdetect*), 327
- `peakdetect_zero_crossing()` (in module *pysisyphus.peakdetect*), 327
- `perf_results` (*pysisyphus.run.RunResult* attribute), 330
- `perp_comp()` (in module *pysisyphus.linalg*), 323
- `perp_component()` (*pysisyphus.irc.GonzalezSchlegel* method), 277
- `perp_component()` (*pysisyphus.irc.GonzalezSchlegel.GonzalezSchlegel* method), 127, 269
- `perpendicular_forces` (*pysisyphus.cos.ChainOfStates.ChainOfStates* property), 109, 216
- `pick_image_inds()` (in module *pysisyphus.helpers*), 320
- `Pipeline` (class in *pysisyphus.stochastic.Pipeline*), 303
- `pivoted_cholesky()` (in module *pysisyphus.linalg*), 323
- `pivoted_cholesky2()` (in module *pysisyphus.linalg*), 324
- `plot()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot()` (*pysisyphus.irc.ParamPlot.ParamPlot* method), 274
- `plot3d()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_afir()` (in module *pysisyphus.plot*), 329
- `plot_all_energies()` (in module *pysisyphus.plot*), 329
- `plot_coords()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_cos_energies()` (in module *pysisyphus.plot*), 329
- `plot_cos_forces()` (in module *pysisyphus.plot*), 329
- `plot_cycle()` (in module *pysisyphus.plot*), 329
- `plot_eigenvalue_structure()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_gau()` (in module *pysisyphus.plot*), 329
- `plot_geoms()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_irc()` (in module *pysisyphus.plot*), 329
- `plot_irc()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_irc_h5()` (in module *pysisyphus.plot*), 329
- `plot_md()` (in module *pysisyphus.plot*), 329
- `plot_opt()` (in module *pysisyphus.plot*), 329
- `plot_opt()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase* method), 78, 155
- `plot_overlaps()` (in module *pysisyphus.plot*), 329
- `plot_scan()` (in module *pysisyphus.plot*), 329
- `plot_trj_energies()` (in module *pysisyphus.plot*), 329
- `poly_line_search()` (in module *pysisyphus.optimizers.poly_fit*), 300
- `polys` (*pysisyphus.optimizers.poly_fit.FitResult* attribute), 300
- `popen()` (*pysisyphus.calculators.Calculator.Calculator* method), 50, 158
- `postprocess()` (*pysisyphus.irc.GonzalezSchlegel* method), 277
- `postprocess()` (*pysisyphus.irc.GonzalezSchlegel.GonzalezSchlegel* method), 127, 269
- `postprocess()` (*pysisyphus.irc.IRC.IRC* method), 126, 271
- `postprocess()` (*pysisyphus.irc.ModeKill* method), 277
- `postprocess()` (*pysisyphus.irc.ModeKill.ModeKill* method), 274
- `postprocess_opt()` (*pysisyphus.optimizers.CubicNewton* method), 301
- `postprocess_opt()` (*pysisyphus.optimizers.CubicNewton.CubicNewton* method), 285
- `postprocess_opt()` (*pysisyphus.optimizers.LayerOpt.LayerOpt* method), 96, 289
- `postprocess_opt()` (*pysisyphus.optimizers.LBFGS.LBFGS* method), 97, 288
- `postprocess_opt()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 86, 292
- `postprocess_opt()` (*pysisyphus.optimizers.RFOptimizer.RFOptimizer* method), 89, 295
- `precon_getter()` (in module *pysisyphus.optimizers.precon*), 301
- `precon_pos_rot()` (in module *pysisyphus.drivers.precon_pos_rot*), 227
- `precondition_gradient()` (*pysisyphus.optimizers.StabilizedQNMMethod.StabilizedQNMMethod* method), 296
- `PreconLBFGS` (class in *pysisyphus.optimizers.PreconLBFGS*), 97, 293
- `PreconSteepestDescent` (class in *pysisyphus.optimizers.PreconSteepestDescent*),

- 294
- `preopt_first_geom` (`pysisyphus.run.RunResult` attribute), 330
- `preopt_last_geom` (`pysisyphus.run.RunResult` attribute), 330
- `prepare()` (`pysisyphus.calculators.Calculator.Calculator` method), 50, 158
- `prepare()` (`pysisyphus.calculators.CFOUR` method), 195
- `prepare()` (`pysisyphus.calculators.CFOUR.CFOUR` method), 69
- `prepare()` (`pysisyphus.irc.DampedVelocityVerlet` method), 276
- `prepare()` (`pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet` method), 126, 268
- `prepare()` (`pysisyphus.irc.EulerPC` method), 276
- `prepare()` (`pysisyphus.irc.EulerPC.EulerPC` method), 127, 268
- `prepare()` (`pysisyphus.irc.IRC.IRC` method), 126, 271
- `prepare()` (`pysisyphus.irc.ModeKill` method), 277
- `prepare()` (`pysisyphus.irc.ModeKill.ModeKill` method), 274
- `prepare_add_args()` (`pysisyphus.calculators.XTB` method), 213
- `prepare_add_args()` (`pysisyphus.calculators.XTB.XTB` method), 67, 191
- `prepare_coords()` (`pysisyphus.calculators.Calculator.Calculator` method), 50, 158
- `prepare_coords()` (`pysisyphus.calculators.CFOUR` method), 195
- `prepare_coords()` (`pysisyphus.calculators.CFOUR.CFOUR` method), 69
- `prepare_coords()` (`pysisyphus.calculators.MOPAC` method), 204
- `prepare_coords()` (`pysisyphus.calculators.MOPAC.MOPAC` method), 65, 174
- `prepare_coords()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `prepare_coords()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `prepare_coords()` (`pysisyphus.calculators.XTB` method), 213
- `prepare_coords()` (`pysisyphus.calculators.XTB.XTB` method), 67, 191
- `prepare_input()` (`pysisyphus.calculators.Calculator.Calculator` method), 50, 159
- `prepare_input()` (`pysisyphus.calculators.CFOUR` method), 196
- `prepare_input()` (`pysisyphus.calculators.CFOUR.CFOUR` method), 70
- `prepare_input()` (`pysisyphus.calculators.Dalton.Dalton` method), 68, 164
- `prepare_input()` (`pysisyphus.calculators.DFTBp` method), 197
- `prepare_input()` (`pysisyphus.calculators.DFTBp.DFTBp` method), 64, 164
- `prepare_input()` (`pysisyphus.calculators.Gaussian16` method), 202
- `prepare_input()` (`pysisyphus.calculators.Gaussian16.Gaussian16` method), 56, 171
- `prepare_input()` (`pysisyphus.calculators.MOPAC` method), 204
- `prepare_input()` (`pysisyphus.calculators.MOPAC.MOPAC` method), 65, 175
- `prepare_input()` (`pysisyphus.calculators.OpenMolcas` method), 208
- `prepare_input()` (`pysisyphus.calculators.OpenMolcas.OpenMolcas` method), 58, 181
- `prepare_input()` (`pysisyphus.calculators.ORCA` method), 207
- `prepare_input()` (`pysisyphus.calculators.ORCA.ORCA` method), 60, 179
- `prepare_input()` (`pysisyphus.calculators.Psi4` method), 209
- `prepare_input()` (`pysisyphus.calculators.Psi4.Psi4` method), 66, 184
- `prepare_input()` (`pysisyphus.calculators.Turbomole` method), 211
- `prepare_input()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 62, 188
- `prepare_input()` (`pysisyphus.calculators.XTB` method), 214
- `prepare_input()` (`pysisyphus.calculators.XTB.XTB` method), 67, 192
- `prepare_line_search()` (`pysisyphus.line_searches.HagerZhang` method), 281
- `prepare_line_search()` (`pysisyphus.line_searches.HagerZhang.HagerZhang` method), 278
- `prepare_line_search()` (`pysisyphus.line_searches.LineSearch.LineSearch` method), 279
- `prepare_mc_afir()` (in module `pysisy-`

`phus.drivers.afir`), 223
`prepare_merge()` (in module `pysisyphus.drivers.merge`), 225
`prepare_opt()` (`pysisyphus.optimizers.BFGS.BFGS` method), 284
`prepare_opt()` (`pysisyphus.optimizers.ConjugateGradient.ConjugateGradient` method), 285
`prepare_opt()` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 287
`prepare_opt()` (`pysisyphus.optimizers.Optimizer.Optimizer` method), 86, 292
`prepare_opt()` (`pysisyphus.optimizers.PreconLBFGS.PreconLBFGS` method), 98, 294
`prepare_opt()` (`pysisyphus.optimizers.QuickMin.QuickMin` method), 294
`prepare_opt()` (`pysisyphus.optimizers.StabilizedQNMethod.StabilizedQNMethod` method), 296
`prepare_opt()` (`pysisyphus.optimizers.SteepestDescent.SteepestDescent` method), 296
`prepare_opt()` (`pysisyphus.optimizers.StringOptimizer.StringOptimizer` method), 296
`prepare_opt()` (`pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer` method), 119, 307
`prepare_opt_cycle()` (`pysisyphus.cos.AdaptiveNEB.AdaptiveNEB` method), 111, 215
`prepare_opt_cycle()` (`pysisyphus.cos.ChainOfStates.ChainOfStates` method), 109, 216
`prepare_opt_cycle()` (`pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates` method), 112, 218
`prepare_overlap_data()` (`pysisyphus.calculators.DFTBp` method), 197
`prepare_overlap_data()` (`pysisyphus.calculators.DFTBp.DFTBp` method), 64, 164
`prepare_overlap_data()` (`pysisyphus.calculators.Gaussian16` method), 202
`prepare_overlap_data()` (`pysisyphus.calculators.Gaussian16.Gaussian16` method), 56, 171
`prepare_overlap_data()` (`pysisyphus.calculators.ORCA` method), 207
`prepare_overlap_data()` (`pysisyphus.calculators.ORCA.ORCA` method), 60, 179
`prepare_overlap_data()` (`pysisyphus.calculators.OverlapCalculator.OverlapCalculator` method), 55, 183
`prepare_overlap_data()` (`pysisyphus.calculators.Turbomole` method), 212
`prepare_overlap_data()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 62, 188
`prepare_path()` (`pysisyphus.calculators.Calculator.Calculator` method), 50, 159
`prepare_pattern()` (`pysisyphus.calculators.Calculator.Calculator` method), 51, 159
`prepare_point_charges()` (`pysisyphus.calculators.Turbomole` method), 212
`prepare_point_charges()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 62, 188
`prepare_sc_afir()` (in module `pysisyphus.drivers.afir`), 223
`prepare_td()` (`pysisyphus.calculators.Turbomole` method), 212
`prepare_td()` (`pysisyphus.calculators.Turbomole.Turbomole` method), 62, 188
`prepare_turbo_coords()` (`pysisyphus.calculators.Calculator.Calculator` method), 51, 159
`prepare_xyz_string()` (`pysisyphus.calculators.Calculator.Calculator` method), 51, 159
`prev_eigvec_max` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` property), 88, 287
`prev_eigvec_min` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` property), 88, 287
`prim_coords` (`pysisyphus.intcoords.RedundantCoords` property), 261
`prim_coords` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 44, 246
`prim_for_human()` (in module `pysisyphus.intcoords.PrimTypes`), 243
`prim_indices_set` (`pysisyphus.intcoords.RedundantCoords` property), 261
`prim_indices_set` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` property), 44, 246
`prim_internals` (`pysisyphus.intcoords.RedundantCoords` property),

261

`prim_internals` (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 246

`PrimInternal` (class in pysisyphus.intcoords.eval), 249

`Primitive` (class in pysisyphus.intcoords.Primitive), 244

`PrimitiveNotDefinedException`, 250, 260

`primitives` (pysisyphus.intcoords.RedundantCoords property), 261

`primitives` (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 246

`prims_from_prim_inputs()` (in module pysisyphus.intcoords.PrimTypes), 243

`PrimTypes` (class in pysisyphus.intcoords.PrimTypes), 242

`principal_axes_are_aligned()` (pysisyphus.Geometry.Geometry method), 41, 315

`print()` (pysisyphus.TablePrinter.TablePrinter method), 317

`print_barrier()` (in module pysisyphus.helpers), 320

`print_bibtex()` (in module pysisyphus.run), 331

`print_capabilities()` (pysisyphus.calculators.Calculator.Calculator method), 51, 160

`print_fragments()` (pysisyphus.stochastic.FragmentKick method), 304

`print_fragments()` (pysisyphus.stochastic.FragmentKick.FragmentKick method), 302

`print_header()` (in module pysisyphus.run), 331

`print_header()` (pysisyphus.TablePrinter.TablePrinter method), 317

`print_info()` (in module pysisyphus.pack), 325

`print_internal_vals()` (in module pysisyphus.trj), 333

`print_internals()` (in module pysisyphus.trj), 333

`print_opt_progress()` (pysisyphus.optimizers.LayerOpt.LayerOpt method), 96, 289

`print_opt_progress()` (pysisyphus.optimizers.Optimizer.Optimizer method), 86, 292

`print_out_fn()` (pysisyphus.calculators.Calculator.Calculator method), 51, 160

`print_perf_results()` (in module pysisyphus.drivers.perf), 226

`print_row()` (pysisyphus.TablePrinter.TablePrinter method), 317

`print_sep()` (pysisyphus.TablePrinter.TablePrinter method), 317

`print_thermoanalysis()` (in module pysisyphus.thermo), 332

`print_typed_prims()` (pysisyphus.intcoords.RedundantCoords method), 261

`print_typed_prims()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 246

`procrustes()` (in module pysisyphus.helpers), 320

`procrustes()` (pysisyphus.optimizers.Optimizer.Optimizer method), 86, 292

`project_coord_hessian()` (pysisyphus.intcoords.CartesianCoords method), 257

`project_hessian()` (pysisyphus.intcoords.CartesianCoords.CartesianCoords method), 238

`project_hessian()` (pysisyphus.intcoords.Coords.CoordSys method), 239

`project_hessian()` (pysisyphus.intcoords.DLC method), 258

`project_hessian()` (pysisyphus.intcoords.DLC.DLC method), 240

`project_hessian()` (pysisyphus.intcoords.RedundantCoords method), 261

`project_hessian()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 246

`project_primitive_on_active_set()` (pysisyphus.intcoords.DLC method), 258

`project_primitive_on_active_set()` (pysisyphus.intcoords.DLC.DLC method), 240

`project_vector()` (pysisyphus.intcoords.RedundantCoords method), 261

`project_vector()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 246

`PROPER_DIHEDRAL` (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

`PROPER_DIHEDRAL2` (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

`proper_dihedrals` (pysisyphus.intcoords.setup.CoordInfo attribute), 254

`psb_update()` (in module pysisyphus.optimizers.hessian_updates), 299

`Psi4` (class in pysisyphus.calculators), 208

`Psi4` (class in pysisyphus.calculators.Psi4), 65, 184

`PT` (in module pysisyphus.intcoords.PrimTypes), 242

`PWHardSphere` (class in pysisyphus.calculators.HardSphere), 171

`PyPsi4` (*class in* `pysisyphus.calculators`), 209
`PyPsi4` (*class in* `pysisyphus.calculators.PyPsi4`), 184
`pysisyphus`
 module, 334
`pysisyphus.benchmarks`
 module, 152
`pysisyphus.benchmarks.Benchmark`
 module, 151
`pysisyphus.benchmarks.data`
 module, 151
`pysisyphus.calculators`
 module, 193
`pysisyphus.calculators.AFIR`
 module, 72, 152
`pysisyphus.calculators.AnaPot`
 module, 154
`pysisyphus.calculators.AnaPot2`
 module, 154
`pysisyphus.calculators.AnaPot3`
 module, 155
`pysisyphus.calculators.AnaPot4`
 module, 155
`pysisyphus.calculators.AnaPotBase`
 module, 78, 155
`pysisyphus.calculators.AnaPotCBM`
 module, 156
`pysisyphus.calculators.AtomAtomTransTorque`
 module, 156
`pysisyphus.calculators.Calculator`
 module, 48, 156
`pysisyphus.calculators.CerjanMiller`
 module, 162
`pysisyphus.calculators.CFOUR`
 module, 68
`pysisyphus.calculators.Composite`
 module, 162
`pysisyphus.calculators.ConicalIntersection`
 module, 162
`pysisyphus.calculators.Dalton`
 module, 68, 164
`pysisyphus.calculators.DFTBp`
 module, 63, 163
`pysisyphus.calculators.DFTD3`
 module, 71
`pysisyphus.calculators.Dimer`
 module, 76, 121, 164
`pysisyphus.calculators.Dummy`
 module, 166
`pysisyphus.calculators.EGO`
 module, 166
`pysisyphus.calculators.EnergyMin`
 module, 167
`pysisyphus.calculators.ExternalPotential`
 module, 168

`pysisyphus.calculators.ExternalPotential.ExternalPotential`
 module, 70
 `pysisyphus.calculators.ExternalPotential.HarmonicSphere`
 module, 71
 `pysisyphus.calculators.ExternalPotential.LogFermi`
 module, 71
 `pysisyphus.calculators.ExternalPotential.Restraint`
 module, 71
 `pysisyphus.calculators.FakeASE`
 module, 79, 169
 `pysisyphus.calculators.FourWellAnaPot`
 module, 169
 `pysisyphus.calculators.FreeEndNEBPot`
 module, 170
 `pysisyphus.calculators.Gaussian09`
 module, 55, 170
 `pysisyphus.calculators.Gaussian16`
 module, 56, 170
 `pysisyphus.calculators.HardSphere`
 module, 171
 `pysisyphus.calculators.IDPPCalculator`
 module, 172
 `pysisyphus.calculators.IPIClient`
 module, 172
 `pysisyphus.calculators.IPIServer`
 module, 172
 `pysisyphus.calculators.LennardJones`
 module, 79, 174
 `pysisyphus.calculators.LEPSBase`
 module, 173
 `pysisyphus.calculators.LEPSExpr`
 module, 173
 `pysisyphus.calculators.MOPAC`
 module, 64, 174
 `pysisyphus.calculators.MullerBrownSymPyPot`
 module, 175
 `pysisyphus.calculators.MultiCalc`
 module, 175
 `pysisyphus.calculators.OBabel`
 module, 68, 175
 `pysisyphus.calculators.ONIOMv2`
 module, 74, 176
 `pysisyphus.calculators.OpenMM`
 module, 180
 `pysisyphus.calculators.OpenMolcas`
 module, 58, 181
 `pysisyphus.calculators.ORCA`
 module, 59, 178
 `pysisyphus.calculators.ORCA5`
 module, 180
 `pysisyphus.calculators.OverlapCalculator`
 module, 53, 182
 `pysisyphus.calculators.parser`
 module, 192

pysisyphus.calculators.Psi4
module, 65, 184

pysisyphus.calculators.PyPsi4
module, 184

pysisyphus.calculators.PyXTB
module, 184

pysisyphus.calculators.Rastrigin
module, 185

pysisyphus.calculators.Remote
module, 185

pysisyphus.calculators.Rosenbrock
module, 185

pysisyphus.calculators.SocketCalc
module, 185

pysisyphus.calculators.TIP3P
module, 79, 186

pysisyphus.calculators.TransTorque
module, 186

pysisyphus.calculators.Turbomole
module, 61, 187

pysisyphus.calculators.WFOWrapper
module, 189

pysisyphus.calculators.WFOWrapper2
module, 189

pysisyphus.calculators.XTB
module, 66, 190

pysisyphus.color
module, 318

pysisyphus.config
module, 318

pysisyphus.constants
module, 318

pysisyphus.cos
module, 221

pysisyphus.cos.AdaptiveNEB
module, 111, 214

pysisyphus.cos.ChainOfStates
module, 108, 215

pysisyphus.cos.FreeEndNEB
module, 112, 217

pysisyphus.cos.FreezingString
module, 217

pysisyphus.cos.GrowingChainOfStates
module, 112, 218

pysisyphus.cos.GrowingNT
module, 218

pysisyphus.cos.GrowingString
module, 113, 219

pysisyphus.cos.NEB
module, 110, 220

pysisyphus.cos.SimpleZTS
module, 112, 221

pysisyphus.db
module, 222

pysisyphus.db.generate_db
module, 221

pysisyphus.db.helpers
module, 221

pysisyphus.db.level
module, 221

pysisyphus.db.molecules
module, 221

pysisyphus.drivers
module, 231

pysisyphus.drivers.afir
module, 222

pysisyphus.drivers.barriers
module, 224

pysisyphus.drivers.birkholz
module, 224

pysisyphus.drivers.merge
module, 224

pysisyphus.drivers.opt
module, 225

pysisyphus.drivers.perf
module, 226

pysisyphus.drivers.pka
module, 226

pysisyphus.drivers.precon_pos_rot
module, 226

pysisyphus.drivers.rates
module, 227

pysisyphus.drivers.replace
module, 230

pysisyphus.drivers.scan
module, 231

pysisyphus.drivers.thermo
module, 231

pysisyphus.dynamics
module, 237

pysisyphus.dynamics.colvars
module, 232

pysisyphus.dynamics.driver
module, 232

pysisyphus.dynamics.Gaussian
module, 231

pysisyphus.dynamics.helpers
module, 233

pysisyphus.dynamics.lincs
module, 236

pysisyphus.dynamics.mdp
module, 236

pysisyphus.dynamics.rattle
module, 236

pysisyphus.dynamics.thermostats
module, 237

pysisyphus.elem_data
module, 318

pysisyphus.exceptions
 module, 318

pysisyphus.filtertrj
 module, 318

pysisyphus.Geometry
 module, 35, 309

pysisyphus.helpers
 module, 318

pysisyphus.helpers_pure
 module, 320

pysisyphus.init_logging
 module, 322

pysisyphus.intcoords
 module, 257

pysisyphus.intcoords.augment_bonds
 module, 248

pysisyphus.intcoords.Bend
 module, 237

pysisyphus.intcoords.Bend2
 module, 237

pysisyphus.intcoords.BondedFragment
 module, 238

pysisyphus.intcoords.Cartesian
 module, 238

pysisyphus.intcoords.CartesianCoords
 module, 238

pysisyphus.intcoords.Coords
 module, 239

pysisyphus.intcoords.derivatives
 module, 248

pysisyphus.intcoords.DistanceFunction
 module, 241

pysisyphus.intcoords.DLC
 module, 240

pysisyphus.intcoords.DummyTorsion
 module, 241

pysisyphus.intcoords.eval
 module, 249

pysisyphus.intcoords.exceptions
 module, 250

pysisyphus.intcoords.findiffs
 module, 250

pysisyphus.intcoords.generate_derivatives
 module, 250

pysisyphus.intcoords.helpers
 module, 251

pysisyphus.intcoords.LinearBend
 module, 241

pysisyphus.intcoords.LinearDisplacement
 module, 241

pysisyphus.intcoords.mp_derivatives
 module, 252

pysisyphus.intcoords.OutOfPlane
 module, 242

pysisyphus.intcoords.Primitive
 module, 244

pysisyphus.intcoords.PrimTypes
 module, 242

pysisyphus.intcoords.RedundantCoords
 module, 42, 244

pysisyphus.intcoords.Rotation
 module, 246

pysisyphus.intcoords.setup
 module, 253

pysisyphus.intcoords.setup_fast
 module, 256

pysisyphus.intcoords.Stretch
 module, 247

pysisyphus.intcoords.Torsion
 module, 247

pysisyphus.intcoords.Torsion2
 module, 247

pysisyphus.intcoords.Translation
 module, 247

pysisyphus.intcoords.update
 module, 256

pysisyphus.intcoords.valid
 module, 257

pysisyphus.interpolate
 module, 264

pysisyphus.interpolate.helpers
 module, 264

pysisyphus.interpolate.IDPP
 module, 263

pysisyphus.interpolate.Interpolator
 module, 263

pysisyphus.interpolate.LST
 module, 263

pysisyphus.interpolate.Redund
 module, 263

pysisyphus.io
 module, 267

pysisyphus.io.cjson
 module, 264

pysisyphus.io.crd
 module, 264

pysisyphus.io.hdf5
 module, 264

pysisyphus.io.hessian
 module, 265

pysisyphus.io.mol2
 module, 265

pysisyphus.io.molden
 module, 265

pysisyphus.io.pdb
 module, 265

pysisyphus.io.pubchem
 module, 265

pysisyphus.io.sdf
module, 266

pysisyphus.io.xyz
module, 266

pysisyphus.io.zmat
module, 266

pysisyphus.irc
module, 276

pysisyphus.irc.DampedVelocityVerlet
module, 126, 268

pysisyphus.irc.DWI
module, 267

pysisyphus.irc.Euler
module, 127, 268

pysisyphus.irc.EulerPC
module, 127, 268

pysisyphus.irc.GonzalezSchlegel
module, 127, 269

pysisyphus.irc.IMKMod
module, 128, 269

pysisyphus.irc.initial_displ
module, 275

pysisyphus.irc.Instanton
module, 272

pysisyphus.irc.IRC
module, 124, 269

pysisyphus.irc.IRCDummy
module, 272

pysisyphus.irc.LQA
module, 128, 273

pysisyphus.irc.ModeKill
module, 274

pysisyphus.irc.ParamPlot
module, 274

pysisyphus.irc.RK4
module, 128, 274

pysisyphus.linalg
module, 323

pysisyphus.line_searches
module, 280

pysisyphus.line_searches.Backtracking
module, 278

pysisyphus.line_searches.HagerZhang
module, 278

pysisyphus.line_searches.interpol
module, 280

pysisyphus.line_searches.LineSearch
module, 279

pysisyphus.line_searches.StrongWolfe
module, 280

pysisyphus.modelfollow
module, 283

pysisyphus.modelfollow.davidson
module, 282

pysisyphus.modelfollow.lanczos
module, 283

pysisyphus.modelfollow.NormalMode
module, 282

pysisyphus.optimizers
module, 301

pysisyphus.optimizers.BacktrackingOptimizer
module, 284

pysisyphus.optimizers.BFGS
module, 284

pysisyphus.optimizers.closures
module, 296

pysisyphus.optimizers.cls_map
module, 297

pysisyphus.optimizers.ConjugateGradient
module, 285

pysisyphus.optimizers.CubicNewton
module, 285

pysisyphus.optimizers.exceptions
module, 297

pysisyphus.optimizers.FIRE
module, 285

pysisyphus.optimizers.gdiis
module, 297

pysisyphus.optimizers.guess_hessians
module, 298

pysisyphus.optimizers.hessian_updates
module, 299

pysisyphus.optimizers.HessianOptimizer
module, 87, 285

pysisyphus.optimizers.LayerOpt
module, 96, 288

pysisyphus.optimizers.LBFGS
module, 96, 287

pysisyphus.optimizers.MicroOptimizer
module, 289

pysisyphus.optimizers.NCOptimizer
module, 290

pysisyphus.optimizers.Optimizer
module, 84, 290

pysisyphus.optimizers.poly_fit
module, 300

pysisyphus.optimizers.precon
module, 301

pysisyphus.optimizers.PreconLBFGS
module, 97, 293

pysisyphus.optimizers.PreconSteepestDescent
module, 294

pysisyphus.optimizers.QuickMin
module, 294

pysisyphus.optimizers.restrict_step
module, 301

pysisyphus.optimizers.RFOptimizer
module, 89, 294

`pysisyphus.optimizers.RSA`
module, 295

`pysisyphus.optimizers.StabilizedQNMethod`
module, 295

`pysisyphus.optimizers.SteepestDescent`
module, 296

`pysisyphus.optimizers.StringOptimizer`
module, 296

`pysisyphus.pack`
module, 325

`pysisyphus.peakdetect`
module, 325

`pysisyphus.plot`
module, 328

`pysisyphus.plotters`
module, 302

`pysisyphus.plotters.AnimPlot`
module, 302

`pysisyphus.run`
module, 329

`pysisyphus.socket_helper`
module, 331

`pysisyphus.stochastic`
module, 304

`pysisyphus.stochastic.align`
module, 304

`pysisyphus.stochastic.FragmentKick`
module, 302

`pysisyphus.stochastic.Kick`
module, 303

`pysisyphus.stochastic.Pipeline`
module, 303

`pysisyphus.TableFormatter`
module, 317

`pysisyphus.TablePrinter`
module, 317

`pysisyphus.testing`
module, 332

`pysisyphus.tests`
module, 305

`pysisyphus.thermo`
module, 332

`pysisyphus.trj`
module, 332

`pysisyphus.tsoptimizers`
module, 308

`pysisyphus.tsoptimizers.RSIRFOptimizer`
module, 120, 305

`pysisyphus.tsoptimizers.RSPRFOptimizer`
module, 120, 305

`pysisyphus.tsoptimizers.TRIM`
module, 120, 306

`pysisyphus.tsoptimizers.TSHessianOptimizer`
module, 118, 306

`pysisyphus.version`
module, 333

`pysisyphus.wrapper`
module, 309

`pysisyphus.wrapper.jmol`
module, 308

`pysisyphus.wrapper.mwfn`
module, 309

`pysisyphus.wrapper.packmol`
module, 309

`pysisyphus.xyzloader`
module, 333

`pysisyphus.yaml_mods`
module, 334

`PyXTB` (class in `pysisyphus.calculators`), 209

`PyXTB` (class in `pysisyphus.calculators.PyXTB`), 184

Q

`Q()` (in `pysisyphus.calculators.LEPSEExpr.LEPSEExpr` method), 173

`q_a()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_a()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_a2()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_a2()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_b()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_b()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_d()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_d()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_d2()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_d2()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_lb()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_lb()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_ld()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_ld()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_oop()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_oop()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_rd1()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_rd1()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`q_rd2()` (in module `pysisyphus.intcoords.derivatives`), 249

`q_rd2()` (in module `pysisyphus.intcoords.mp_derivatives`), 253

`qH` (`pysisyphus.calculators.TIP3P` attribute), 210

- qH (*pysisyphus.calculators.TIP3P.TIP3P* attribute), 79, 186
- q0 (*pysisyphus.calculators.TIP3P* attribute), 210
- q0 (*pysisyphus.calculators.TIP3P.TIP3P* attribute), 80, 186
- qs (*pysisyphus.modelfollow.davidson.DavidsonResult* attribute), 282
- quadratic_model() (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* static method), 88, 287
- quartic_fit() (in module *pysisyphus.optimizers.poly_fit*), 300
- quaternion_to_rot_mat() (in module *pysisyphus.linalg*), 324
- QuickMin (class in *pysisyphus.optimizers.QuickMin*), 294
- quintic_fit() (in module *pysisyphus.optimizers.poly_fit*), 301
- ## R
- r (*pysisyphus.cos.GrowingNT.GrowingNT* property), 219
- r (*pysisyphus.io.zmat.ZLine* attribute), 266
- raise_exception() (*pysisyphus.calculators.Dummy* method), 199
- raise_exception() (*pysisyphus.calculators.Dummy.Dummy* method), 166
- Rastrigin (class in *pysisyphus.calculators.Rastrigin*), 185
- rate_bell (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
- rate_eckart (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
- rate_eyring (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
- rate_wigner (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
- rattle_closure() (in module *pysisyphus.dynamics.rattle*), 236
- ReactionRates (class in *pysisyphus.drivers.rates*), 227
- read_aux() (*pysisyphus.calculators.MOPAC* method), 204
- read_aux() (*pysisyphus.calculators.MOPAC.MOPAC* method), 65, 175
- read_cfour_geom() (in module *pysisyphus.calculators.CFOUR*), 70
- read_geoms() (in module *pysisyphus.trj*), 333
- reattach() (*pysisyphus.calculators.OpenMolcas* method), 208
- reattach() (*pysisyphus.calculators.OpenMolcas.OpenMolcas* method), 58, 181
- reattach() (*pysisyphus.calculators.ORCA* method), 207
- reattach() (*pysisyphus.calculators.ORCA.ORCA* method), 60, 179
- reattach() (*pysisyphus.calculators.XTB* method), 214
- reattach() (*pysisyphus.calculators.XTB.XTB* method), 67, 192
- RebuiltInternalsException, 250
- recursive_update() (in module *pysisyphus.helpers_pure*), 322
- recv_closure() (in module *pysisyphus.socket_helper*), 331
- red() (in module *pysisyphus.color*), 318
- red_mass (*pysisyphus.modelfollow.NormalMode* property), 283
- red_mass (*pysisyphus.modelfollow.NormalMode.NormalMode* property), 282
- Redund (class in *pysisyphus.interpolate.Redund*), 263
- RedundantCoords (class in *pysisyphus.intcoords*), 260
- RedundantCoords (class in *pysisyphus.intcoords.RedundantCoords*), 43, 244
- ref_hessian (*pysisyphus.calculators.EGO* property), 199
- ref_hessian (*pysisyphus.calculators.EGO.EGO* property), 166
- relax_afir_path() (in module *pysisyphus.drivers.afir*), 223
- relaxed_1d_scan() (in module *pysisyphus.drivers.scan*), 231
- relaxed_scan() (in module *pysisyphus.drivers.scan*), 231
- Remote (class in *pysisyphus.calculators*), 209
- Remote (class in *pysisyphus.calculators.Remote*), 185
- remove_com_velocity() (in module *pysisyphus.dynamics.helpers*), 234
- remove_duplicates() (in module *pysisyphus.helpers_pure*), 322
- remove_translation() (*pysisyphus.calculators.Dimer* method), 198
- remove_translation() (*pysisyphus.calculators.Dimer.Dimer* method), 77, 122, 165
- render_cdd_cube() (in module *pysisyphus.wrapper.jmol*), 308
- render_cdd_cube() (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* method), 55, 183
- render_cdds() (in module *pysisyphus.plot*), 329
- render_data_groups() (in module *pysisyphus.calculators.Turbomole*), 63, 188
- render_geom_and_charges() (in module *pysisyphus.wrapper.jmol*), 308
- render_rx_rates() (in module *pysisyphus.drivers.rates*), 229
- render_xyz() (in module *pysisyphus.io.mol2*), 265
- renorm_mos() (*pysisyphus* module), 265

`phus.calculators.OverlapCalculator.OverlapCalculator` (static method), 55, 183
`reparam_cart()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`reparam_dlc()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`reparametrize()` (`pysisyphus.cos.FreezingString.FreezingString` method), 218
`reparametrize()` (`pysisyphus.cos.GrowingNT.GrowingNT` method), 219
`reparametrize()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`reparametrize()` (`pysisyphus.cos.SimpleZTS.SimpleZTS` method), 112, 221
`reparametrize()` (`pysisyphus.Geometry.Geometry` method), 41, 315
`replace_atom()` (in module `pysisyphus.drivers.replace`), 230
`replace_atoms()` (in module `pysisyphus.drivers.replace`), 230
`report_bonds()` (`pysisyphus.irc.IRC.IRC` method), 126, 271
`report_charges()` (`pysisyphus.calculators.OpenMM.OpenMM` method), 180
`report_conv_thresholds()` (`pysisyphus.irc.IRC.IRC` method), 126, 271
`report_conv_thresholds()` (`pysisyphus.optimizers.Optimizer.Optimizer` method), 86, 293
`report_frgs()` (in module `pysisyphus.drivers.precon_pos_rot`), 227
`report_frozen_atoms()` (in module `pysisyphus.helpers_pure`), 322
`report_isotopes()` (in module `pysisyphus.helpers_pure`), 322
`report_mats()` (in module `pysisyphus.drivers.precon_pos_rot`), 227
`res_rms` (`pysisyphus.modefollow.davidson.DavidsonResult` attribute), 282
`reset()` (`pysisyphus.optimizers.BacktrackingOptimizer.BacktrackingOptimizer` method), 284
`reset()` (`pysisyphus.optimizers.ConjugateGradient.ConjugateGradient` method), 285
`reset()` (`pysisyphus.optimizers.FIRE.FIRE` method), 285
`reset()` (`pysisyphus.optimizers.HessianOptimizer.HessianOptimizer` method), 88, 287
`reset()` (`pysisyphus.optimizers.LBFGS.LBFGS` method), 97, 288
`reset()` (`pysisyphus.optimizers.QuickMin.QuickMin` method), 294
`reset()` (`pysisyphus.optimizers.StringOptimizer.StringOptimizer` method), 296
`reset_client_connection()` (`pysisyphus.calculators.IPIServer` method), 203
`reset_client_connection()` (`pysisyphus.calculators.IPIServer.IPIServer` method), 172
`reset_constraints()` (`pysisyphus.intcoords.DLC` method), 258
`reset_constraints()` (`pysisyphus.intcoords.DLC.DLC` method), 240
`reset_coords()` (`pysisyphus.Geometry.Geometry` method), 41, 315
`reset_geometries()` (`pysisyphus.cos.GrowingString.GrowingString` method), 113, 220
`resize_h5_group()` (in module `pysisyphus.io.hdf5`), 264
`resname` (`pysisyphus.db.molecules.Molecule` attribute), 222
`restart_interpolate()` (`pysisyphus.interpolate.Redund.Redund` method), 263
`restore_org_hessian()` (`pysisyphus.calculators.Calculator.Calculator` method), 52, 160
`Restraint` (class in `pysisyphus.calculators.ExternalPotential`), 169
`restrict_step()` (in module `pysisyphus.optimizers.restrict_step`), 301
`restrict_step_components()` (`pysisyphus.optimizers.StringOptimizer.StringOptimizer` method), 296
`results` (`pysisyphus.cos.ChainOfStates.ChainOfStates` property), 110, 216
`results_to_json()` (in module `pysisyphus.helpers_pure`), 322
`retrieved_list_for()` (`pysisyphus.calculators.IPIServer` method), 203
`retrieved_list_for()` (`pysisyphus.calculators.IPIServer.IPIServer` method), 172
`return_inds()` (`pysisyphus.intcoords.RedundantCoords` method), 261
`return_inds()` (`pysisyphus.intcoords.RedundantCoords.RedundantCoords` method), 44, 246
`save_data()` (`pysisyphus.calculators.Gaussian16` method), 202

reuse_data() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 56, 171

rf_ind (pysisyphus.cos.GrowingString.GrowingString property), 113, 220

rfo_dict (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer attribute), 88, 287

rfo_model() (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer static method), 89, 287

RFOptimizer (class in pysisyphus.optimizers.RFOptimizer), 89, 294

rho() (pysisyphus.intcoords.Primitive.Primitive static method), 244

right_frontier (pysisyphus.cos.FreezingString.FreezingString property), 218

right_size (pysisyphus.cos.GrowingString.GrowingString property), 113, 220

rind (pysisyphus.io.zmat.ZLine attribute), 266

RK4 (class in pysisyphus.irc), 277

RK4 (class in pysisyphus.irc.RK4), 128, 274

rms() (in module pysisyphus.helpers_pure), 322

rms() (pysisyphus.cos.ChainOfStates.ChainOfStates method), 110, 216

rms_force_converged (pysisyphus.optimizers.Optimizer.ConvInfo attribute), 84, 290

rms_step_converged (pysisyphus.optimizers.Optimizer.ConvInfo attribute), 84, 290

RMSD (class in pysisyphus.calculators.ExternalPotential), 168

rmsd() (pysisyphus.Geometry.Geometry method), 41, 315

rmsd_grad() (in module pysisyphus.linalg), 324

ROBUST_TORSION1 (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

ROBUST_TORSION2 (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

RobustTorsion1 (class in pysisyphus.intcoords), 262

RobustTorsion2 (class in pysisyphus.intcoords), 262

rOH (pysisyphus.calculators.TIP3P attribute), 210

rOH (pysisyphus.calculators.TIP3P.TIP3P attribute), 80, 186

root (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer property), 119, 307

roots (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer property), 119, 307

Rosenbrock (class in pysisyphus.calculators.Rosenbrock), 185

rot_force (pysisyphus.calculators.Dimer property), 199

rot_force (pysisyphus.calculators.Dimer.Dimer property), 77, 122, 165

rot_quaternion() (in module pysisyphus.linalg), 324

rotate() (pysisyphus.Geometry.Geometry method), 41, 315

rotate_coords1() (pysisyphus.calculators.Dimer.Dimer method), 77, 122, 165

rotate_coords1() (pysisyphus.calculators.Dimer.Dimer method), 77, 122, 165

rotate_gradient() (in module pysisyphus.calculators.CFOUR), 70

rotate_inplace() (in module pysisyphus.drivers.precon_pos_rot), 227

Rotation (class in pysisyphus.intcoords.Rotation), 246

ROTATION (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

ROTATION_A (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

ROTATION_B (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

ROTATION_C (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

rotation_indices (pysisyphus.intcoords.RedundantCoords property), 261

rotation_indices (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 44, 246

rotation_inds (pysisyphus.intcoords.setup.CoordInfo attribute), 254

RotationA (class in pysisyphus.intcoords), 262

RotationA (class in pysisyphus.intcoords.Rotation), 246

RotationB (class in pysisyphus.intcoords), 262

RotationB (class in pysisyphus.intcoords.Rotation), 246

RotationC (class in pysisyphus.intcoords), 262

RotationC (class in pysisyphus.intcoords.Rotation), 247

RotationConverged, 77, 122, 166

RSA (class in pysisyphus.optimizers.RSA), 295

RSIRFOptimizer (class in pysisyphus.tsoptimizers), 308

RSIRFOptimizer (class in pysisyphus.tsoptimizers.RSIRFOptimizer), 120, 305

RSPRFOptimizer (class in pysisyphus.tsoptimizers), 308

RSPRFOptimizer (class in pysisyphus.tsoptimizers.RSPRFOptimizer), 120, 305

run() (in module pysisyphus.db.generate_db), 221

run() (in module pysisyphus.drivers.replace), 230

run() (in module pysisyphus.filtertrj), 318

run() (in module pysisyphus.pack), 325

run() (in module pysisyphus.plot), 329

run() (in module pysisyphus.run), 331

run() (in module pysisyphus.TableFormatter), 317

run() (in module pysisyphus.trj), 333

- phus.calculators.ONIOMv2.ONIOM* method), 76, 178
- `run_cycle()` (*pysisyphus.stochastic.Kick* method), 304
- `run_cycle()` (*pysisyphus.stochastic.Kick.Kick* method), 303
- `run_detect_paths()` (in module *pysisyphus.config*), 318
- `run_double_mol_calculation()` (*pysisyphus.calculators.Gaussian16* method), 202
- `run_double_mol_calculation()` (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 57, 171
- `run_double_mol_calculation()` (*pysisyphus.calculators.Turbomole* method), 212
- `run_double_mol_calculation()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 63, 188
- `run_endopt()` (in module *pysisyphus.run*), 331
- `run_from_dict()` (in module *pysisyphus.run*), 331
- `run_geom_opt()` (*pysisyphus.stochastic.Pipeline.Pipeline* method), 303
- `run_irc()` (in module *pysisyphus.run*), 331
- `run_line_search()` (*pysisyphus.line_searches.Backtracking* method), 281
- `run_line_search()` (*pysisyphus.line_searches.Backtracking.Backtracking* method), 278
- `run_line_search()` (*pysisyphus.line_searches.HagerZhang* method), 281
- `run_line_search()` (*pysisyphus.line_searches.HagerZhang.HagerZhang* method), 278
- `run_line_search()` (*pysisyphus.line_searches.LineSearch.LineSearch* method), 279
- `run_line_search()` (*pysisyphus.line_searches.StrongWolfe* method), 281
- `run_line_search()` (*pysisyphus.line_searches.StrongWolfe.StrongWolfe* method), 280
- `run_mc_afir_paths()` (in module *pysisyphus.drivers.afir*), 223
- `run_md()` (in module *pysisyphus.dynamics.mdp*), 236
- `run_md()` (in module *pysisyphus.run*), 331
- `run_md()` (*pysisyphus.calculators.XTB* method), 214
- `run_md()` (*pysisyphus.calculators.XTB.XTB* method), 68, 192
- `run_mdp()` (in module *pysisyphus.run*), 331
- `run_merge()` (in module *pysisyphus.drivers.merge*), 225
- `run_opt()` (in module *pysisyphus.drivers.opt*), 226
- `run_opt()` (in module *pysisyphus.drivers.replace*), 230
- `run_opt()` (*pysisyphus.calculators.XTB* method), 214
- `run_opt()` (*pysisyphus.calculators.XTB.XTB* method), 68, 192
- `run_perf()` (in module *pysisyphus.drivers.perf*), 226
- `run_precontr()` (in module *pysisyphus.drivers.precon_pos_rot*), 227
- `run_preopt()` (in module *pysisyphus.run*), 331
- `run_rwfdump()` (*pysisyphus.calculators.Gaussian16* method), 202
- `run_rwfdump()` (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 57, 171
- `run_sc_afir_paths()` (in module *pysisyphus.drivers.afir*), 223
- `run_scan()` (in module *pysisyphus.run*), 331
- `run_stable()` (*pysisyphus.calculators.Gaussian16* method), 202
- `run_stable()` (*pysisyphus.calculators.Gaussian16.Gaussian16* method), 57, 171
- `run_stochastic()` (in module *pysisyphus.run*), 331
- `run_thermo()` (in module *pysisyphus.drivers.thermo*), 231
- `run_topo()` (*pysisyphus.calculators.XTB* method), 214
- `run_topo()` (*pysisyphus.calculators.XTB.XTB* method), 68, 192
- `run_tsopt_from_cos()` (in module *pysisyphus.run*), 331
- `RunResult` (class in *pysisyphus.run*), 329
- ## S
- `s` (*pysisyphus.calculators.EGO* property), 199
- `s` (*pysisyphus.calculators.EGO.EGO* property), 166
- `s` (*pysisyphus.dynamics.Gaussian.Gaussian* property), 231
- `save()` (*pysisyphus.irc.ParamPlot.ParamPlot* method), 274
- `save_coords()` (*pysisyphus.irc.ParamPlot.ParamPlot* method), 274
- `save_hessian()` (in module *pysisyphus.io*), 267
- `save_hessian()` (in module *pysisyphus.io.hessian*), 265
- `save_hessian()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 89, 287
- `save_orca_pc_file()` (in module *pysisyphus.calculators.ORCA*), 61, 180
- `save_third_deriv()` (in module *pysisyphus.io*), 267
- `save_third_deriv()` (in module *pysisyphus.io.hessian*), 265
- `scale_by_max_step()` (in module *pysisyphus.optimizers.restrict_step*), 301
- `scale_by_max_step()` (*pysisyphus.optimizers.Optimizer.Optimizer* method),

- 86, 293
- `scale_max_element()` (pysisyphus.optimizers.PreconLBFGS.PreconLBFGS method), 98, 294
- `scale_velocities_to_energy()` (in module pysisyphus.dynamics.helpers), 235
- `scale_velocities_to_temperatue()` (in module pysisyphus.dynamics.helpers), 235
- `scan_energies` (pysisyphus.run.RunResult attribute), 330
- `scan_geoms` (pysisyphus.run.RunResult attribute), 330
- `scan_vals` (pysisyphus.run.RunResult attribute), 330
- `scipy_corrector_step()` (pysisyphus.irc.EulerPC method), 276
- `scipy_corrector_step()` (pysisyphus.irc.EulerPC.EulerPC method), 127, 268
- `sd_step()` (pysisyphus.optimizers.MicroOptimizer method), 301
- `sd_step()` (pysisyphus.optimizers.MicroOptimizer.MicroOptimizer method), 289
- `sdf_from_cid()` (in module pysisyphus.io.pubchem), 265
- `secant()` (pysisyphus.line_searches.HagerZhang method), 281
- `secant()` (pysisyphus.line_searches.HagerZhang.HagerZhang method), 278
- `send_closure()` (in module pysisyphus.socket_helper), 331
- `set_active_set()` (pysisyphus.intcoords.DLC method), 258
- `set_active_set()` (pysisyphus.intcoords.DLC.DLC method), 240
- `set_atoms_and_funcs()` (pysisyphus.calculators.AFIR method), 194
- `set_atoms_and_funcs()` (pysisyphus.calculators.AFIR.AFIR method), 73, 154
- `set_calculator()` (pysisyphus.Geometry.Geometry method), 41, 315
- `set_chkfiles()` (pysisyphus.calculators.EnergyMin method), 200
- `set_chkfiles()` (pysisyphus.calculators.EnergyMin.EnergyMin method), 167
- `set_chkfiles()` (pysisyphus.calculators.Gaussian16 method), 202
- `set_chkfiles()` (pysisyphus.calculators.Gaussian16.Gaussian16 method), 57, 171
- `set_chkfiles()` (pysisyphus.calculators.ORCA method), 207
- `set_chkfiles()` (pysisyphus.calculators.ORCA.ORCA method), 60, 179
- `set_chkfiles()` (pysisyphus.calculators.Turbomole method), 212
- `set_chkfiles()` (pysisyphus.calculators.Turbomole.Turbomole method), 63, 188
- `set_climbing_forces()` (pysisyphus.cos.ChainOfStates.ChainOfStates method), 110, 217
- `set_coord()` (pysisyphus.Geometry.Geometry method), 41, 316
- `set_coords()` (pysisyphus.cos.GrowingString.GrowingString method), 113, 220
- `set_coords()` (pysisyphus.Geometry.Geometry method), 41, 316
- `set_coords_at()` (pysisyphus.cos.ChainOfStates.ChainOfStates method), 110, 217
- `set_cross_vec()` (pysisyphus.intcoords.Primitive.Primitive method), 244
- `set_data()` (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190
- `set_data()` (pysisyphus.irc.IRC.IRC method), 126, 271
- `set_from_nested_list()` (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189
- `set_from_nested_list()` (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190
- `set_h5_hessian()` (pysisyphus.Geometry.Geometry method), 41, 316
- `set_images()` (pysisyphus.cos.ChainOfStates.ChainOfStates method), 110, 217
- `set_inds_from_typed_prims()` (pysisyphus.intcoords.RedundantCoords method), 261
- `set_inds_from_typed_prims()` (pysisyphus.intcoords.RedundantCoords.RedundantCoords method), 44, 246
- `set_mo_coeffs()` (pysisyphus.calculators.ORCA method), 207
- `set_mo_coeffs()` (pysisyphus.calculators.ORCA.ORCA method), 60, 180
- `set_mo_coeffs_in_gbw()` (pysisyphus.calculators.ORCA static method), 207
- `set_mo_coeffs_in_gbw()` (pysisyphus.calculators.ORCA.ORCA static method), 60, 180
- `set_N_invs()` (pysisyphus.calculators.TransTorque method), 210
- `set_N_invs()` (pysisy-

- phus.calculators.TransTorque.TransTorque* method), 186
- `set_N_raw()` (*pysisyphus.calculators.Dimer* method), 199
- `set_N_raw()` (*pysisyphus.calculators.Dimer.Dimer* method), 77, 122, 165
- `set_new_frontier_nodes()` (*pysisyphus.cos.FreezingString.FreezingString* method), 218
- `set_new_node()` (*pysisyphus.cos.GrowingChainOfStates.GrowingChainOfStates* method), 112, 218
- `set_new_trust_radius()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 89, 287
- `set_occ_and_mo_nums()` (*pysisyphus.calculators.Turbomole* method), 212
- `set_occ_and_mo_nums()` (*pysisyphus.calculators.Turbomole.Turbomole* method), 63, 188
- `set_primitive_indices()` (*pysisyphus.intcoords.RedundantCoords* method), 261
- `set_primitive_indices()` (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 246
- `set_restart_info()` (*pysisyphus.calculators.Calculator.Calculator* method), 52, 161
- `set_restart_info()` (*pysisyphus.Geometry.Geometry* method), 41, 316
- `set_restart_info()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 87, 293
- `set_results()` (*pysisyphus.Geometry.Geometry* method), 41, 316
- `set_variable_springs()` (*pysisyphus.cos.NEB.NEB* method), 111, 220
- `set_vector()` (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 110, 217
- `set_zero_forces_for_fixed_images()` (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 110, 217
- `SetPlan` (class in *pysisyphus.calculators.Calculator*), 53, 161
- `setup()` (*pysisyphus.calculators.OBabel* method), 205
- `setup()` (*pysisyphus.calculators.OBabel.OBabel* method), 68, 175
- `setup_redundant()` (in module *pysisyphus.intcoords.setup*), 255
- `setup_redundant_from_geom()` (in module *pysisyphus.intcoords.setup*), 255
- `setup_run_dict()` (in module *pysisyphus.run*), 331
- `shake()` (in module *pysisyphus.trj*), 333
- `shake_coords()` (in module *pysisyphus.helpers*), 320
- `should_bias_f0` (*pysisyphus.calculators.Dimer* property), 199
- `should_bias_f0` (*pysisyphus.calculators.Dimer.Dimer* property), 77, 122, 165
- `should_bias_f1` (*pysisyphus.calculators.Dimer* property), 199
- `should_bias_f1` (*pysisyphus.calculators.Dimer.Dimer* property), 77, 122, 165
- `show()` (*pysisyphus.irc.ParamPlot.ParamPlot* method), 274
- `sigma` (*pysisyphus.calculators.TIP3P* attribute), 210
- `sigma` (*pysisyphus.calculators.TIP3P.TIP3P* attribute), 80, 186
- `simple_guess()` (in module *pysisyphus.optimizers.guess_hessians*), 298
- `simple_peaks()` (in module *pysisyphus.helpers*), 320
- `SimpleZTS` (class in *pysisyphus.cos.SimpleZTS*), 112, 221
- `slugify_worker()` (in module *pysisyphus.helpers*), 320
- `small_lbfgs_closure()` (in module *pysisyphus.optimizers.closures*), 297
- `SocketCalc` (class in *pysisyphus.calculators.SocketCalc*), 185
- `solve_rfo()` (*pysisyphus.optimizers.HessianOptimizer.HessianOptimizer* method), 89, 287
- `sort_by_central()` (in module *pysisyphus.helpers_pure*), 322
- `sort_by_prim_type()` (in module *pysisyphus.intcoords.setup*), 255
- `sphere_radius_from_volume()` (in module *pysisyphus.pack*), 325
- `spline()` (*pysisyphus.cos.GrowingString.GrowingString* method), 113, 220
- `spline_plot_cycles()` (in module *pysisyphus.plot*), 329
- `spline_redistribute()` (in module *pysisyphus.trj*), 333
- `split_xyz_str()` (in module *pysisyphus.xyzloader*), 333
- `srl_update()` (in module *pysisyphus.optimizers.hessian_updates*), 299
- `StabilizedQNMethod` (class in *pysisyphus.optimizers.StabilizedQNMethod*), 295
- `standard_orientation()` (in module *pysisyphus.trj*), 333
- `standard_orientation()` (*pysisyphus.Geometry.Geometry* method), 41, 316
- `standard_state_corr()` (in module *pysisyphus.helpers_pure*), 322
- `standardize_geoms()` (in module *pysisyphus.trj*), 333
- `statistics()` (*pysisyphus.calculators.AnaPotBase.AnaPotBase*

- method), 78, 156
- SteepestDescent (class in pysisyphus.drivers.precon_pos_rot), 226
- SteepestDescent (class in pysisyphus.optimizers.SteepestDescent), 296
- step (pysisyphus.dynamics.driver.MDResult attribute), 232
- step() (pysisyphus.irc.DampedVelocityVerlet method), 276
- step() (pysisyphus.irc.DampedVelocityVerlet.DampedVelocityVerlet method), 127, 268
- step() (pysisyphus.irc.Euler method), 276
- step() (pysisyphus.irc.Euler.Euler method), 127, 268
- step() (pysisyphus.irc.EulerPC method), 276
- step() (pysisyphus.irc.EulerPC.EulerPC method), 127, 268
- step() (pysisyphus.irc.GonzalezSchlegel method), 277
- step() (pysisyphus.irc.GonzalezSchlegel.GonzalezSchlegel method), 127, 269
- step() (pysisyphus.irc.IMKMod method), 277
- step() (pysisyphus.irc.IMKMod.IMKMod method), 128, 269
- step() (pysisyphus.irc.LQA method), 277
- step() (pysisyphus.irc.LQA.LQA method), 128, 273
- step() (pysisyphus.irc.ModeKill method), 277
- step() (pysisyphus.irc.ModeKill.ModeKill method), 274
- step() (pysisyphus.irc.RK4 method), 277
- step() (pysisyphus.irc.RK4.RK4 method), 128, 274
- step_along_tangent() (pysisyphus.interpolate.Redund.Redund method), 263
- step_and_grad_from_line_search() (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer method), 119, 307
- stochastic (pysisyphus.run.RunResult attribute), 330
- store_and_track() (pysisyphus.calculators.DFTBp method), 197
- store_and_track() (pysisyphus.calculators.DFTBp.DFTBp method), 64, 164
- store_and_track() (pysisyphus.calculators.Gaussian16 method), 202
- store_and_track() (pysisyphus.calculators.Gaussian16.Gaussian16 method), 57, 171
- store_and_track() (pysisyphus.calculators.ORCA method), 207
- store_and_track() (pysisyphus.calculators.ORCA.ORCA method), 60, 180
- store_and_track() (pysisyphus.calculators.Turbomole method), 212
- store_and_track() (pysisyphus.calculators.Turbomole.Turbomole method), 63, 188
- store_overlap_data() (pysisyphus.calculators.OverlapCalculator.OverlapCalculator method), 55, 183
- stored_calculations (pysisyphus.calculators.OverlapCalculator.OverlapCalculator property), 55, 183
- Stretch (class in pysisyphus.intcoords), 262
- Stretch (class in pysisyphus.intcoords.Stretch), 247
- string_size (pysisyphus.cos.GrowingString.GrowingString property), 113, 220
- StringOptimizer (class in pysisyphus.optimizers.StringOptimizer), 296
- strong_curvature_condition() (pysisyphus.line_searches.LineSearch.LineSearch method), 279
- strong_wolfe_condition() (pysisyphus.line_searches.LineSearch.LineSearch method), 279
- StrongWolfe (class in pysisyphus.line_searches), 281
- StrongWolfe (class in pysisyphus.line_searches.StrongWolfe), 280
- sub_control() (pysisyphus.calculators.Turbomole method), 212
- sub_control() (pysisyphus.calculators.Turbomole.Turbomole method), 63, 188
- sufficiently_decreased() (pysisyphus.line_searches.LineSearch.LineSearch method), 279
- sum_formula (pysisyphus.Geometry.Geometry property), 41, 316
- sum_hessians() (in module pysisyphus.dynamics.thermostats), 237
- surface_area (pysisyphus.calculators.ExternalPotential.HarmonicSphere property), 168
- svd_inv() (in module pysisyphus.linalg), 324
- swart_guess() (in module pysisyphus.optimizers.guess_hessians), 298
- sym_mat_from_tril() (in module pysisyphus.linalg), 324
- sym_mat_from_triu() (in module pysisyphus.linalg), 324
- ## T
- T (pysisyphus.dynamics.driver.MDResult attribute), 232
- T_crossover_from_eigval() (in module pysisyphus.irc.Instanton), 273
- T_crossover_from_ts() (in module pysisyphus.irc.Instanton), 273
- t_ps (pysisyphus.dynamics.driver.MDResult attribute), 232

TableFormatter (class in *pysisyphus.TableFormatter*), 317
TablePrinter (class in *pysisyphus.TablePrinter*), 317
take_quad_step() (*pysisyphus.line_searches.HagerZhang* method), 281
take_quad_step() (*pysisyphus.line_searches.HagerZhang.HagerZhang* method), 279
take_step() (*pysisyphus.optimizers.MicroOptimizer* method), 301
take_step() (*pysisyphus.optimizers.MicroOptimizer.MicroOptimizer* method), 289
taylor() (in module *pysisyphus.irc.DWI*), 267
taylor_closure() (in module *pysisyphus.irc.initial_displ*), 275
taylor_grad() (in module *pysisyphus.irc.DWI*), 267
temperature (*pysisyphus.drivers.rates.ReactionRates* attribute), 227
temperature_for_kinetic_energy() (in module *pysisyphus.dynamics.helpers*), 235
terminated (*pysisyphus.dynamics.driver.MDResult* attribute), 232
thermo (*pysisyphus.helpers.FinalHessianResult* attribute), 319
third_deriv_fd() (in module *pysisyphus.irc.initial_displ*), 276
ThirdDerivResult (class in *pysisyphus.irc.initial_displ*), 275
timed() (in module *pysisyphus.helpers_pure*), 322
TIP3P (class in *pysisyphus.calculators*), 210
TIP3P (class in *pysisyphus.calculators.TIP3P*), 79, 186
tmp_xyz_handle() (*pysisyphus.Geometry.Geometry* method), 42, 316
TMTRIC (class in *pysisyphus.intcoords*), 262
TMTRIC (class in *pysisyphus.intcoords.RedundantCoords*), 45, 246
to_float() (in module *pysisyphus.calculators.parser*), 192
to_sets() (in module *pysisyphus.helpers_pure*), 322
to_subscript_num() (in module *pysisyphus.helpers_pure*), 322
Torsion (class in *pysisyphus.intcoords*), 262
Torsion (class in *pysisyphus.intcoords.Torsion*), 247
Torsion2 (class in *pysisyphus.intcoords*), 262
Torsion2 (class in *pysisyphus.intcoords.Torsion2*), 247
total_mass (*pysisyphus.Geometry.Geometry* property), 42, 316
touch() (in module *pysisyphus.helpers_pure*), 322
track_root() (*pysisyphus.calculators.OverlapCalculator.OverlapCalculator* method), 55, 183
transform_forces() (*pysisyphus.intcoords.CartesianCoords* method), 257
transform_forces() (*pysisyphus.intcoords.CartesianCoords.CartesianCoords* method), 238
transform_forces() (*pysisyphus.intcoords.Coords.CoordSys* method), 239
transform_forces() (*pysisyphus.intcoords.RedundantCoords* method), 261
transform_forces() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 246
transform_hessian() (*pysisyphus.intcoords.CartesianCoords* method), 257
transform_hessian() (*pysisyphus.intcoords.CartesianCoords.CartesianCoords* method), 238
transform_hessian() (*pysisyphus.intcoords.Coords.CoordSys* method), 239
transform_hessian() (*pysisyphus.intcoords.DLC* method), 259
transform_hessian() (*pysisyphus.intcoords.DLC.DLC* method), 240
transform_hessian() (*pysisyphus.intcoords.RedundantCoords* method), 261
transform_hessian() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 44, 246
transform_int_step() (in module *pysisyphus.intcoords.update*), 256
transform_int_step() (*pysisyphus.intcoords.CartesianCoords* method), 257
transform_int_step() (*pysisyphus.intcoords.CartesianCoords.CartesianCoords* method), 238
transform_int_step() (*pysisyphus.intcoords.Coords.CoordSys* method), 239
transform_int_step() (*pysisyphus.intcoords.DLC* method), 259
transform_int_step() (*pysisyphus.intcoords.DLC.DLC* method), 240
transform_int_step() (*pysisyphus.intcoords.RedundantCoords* method), 262
transform_int_step() (*pysisyphus.intcoords.RedundantCoords.RedundantCoords* method), 45, 246
translate() (in module *pysisyphus.trj*), 333

Translation (class in pysisyphus.intcoords.Translation), 247

TRANSLATION (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

translation_indices (pysisyphus.intcoords.RedundantCoords property), 262

translation_indices (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 45, 246

translation_inds (pysisyphus.intcoords.setup.CoordInfo attribute), 254

TRANSLATION_X (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

TRANSLATION_Y (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

TRANSLATION_Z (pysisyphus.intcoords.PrimTypes.PrimTypes attribute), 243

TranslationX (class in pysisyphus.intcoords), 262

TranslationX (class in pysisyphus.intcoords.Translation), 247

TranslationY (class in pysisyphus.intcoords), 262

TranslationY (class in pysisyphus.intcoords.Translation), 247

TranslationZ (class in pysisyphus.intcoords), 262

TranslationZ (class in pysisyphus.intcoords.Translation), 247

TransTorque (class in pysisyphus.calculators), 210

TransTorque (class in pysisyphus.calculators.TransTorque), 186

TRIC (class in pysisyphus.intcoords), 262

TRIC (class in pysisyphus.intcoords.RedundantCoords), 45, 246

TRIM (class in pysisyphus.tsoptimizers), 308

TRIM (class in pysisyphus.tsoptimizers.TRIM), 120, 306

ts_bfgs_update() (in module pysisyphus.optimizers.hessian_updates), 299

ts_bfgs_update_org() (in module pysisyphus.optimizers.hessian_updates), 299

ts_bfgs_update_revised() (in module pysisyphus.optimizers.hessian_updates), 299

ts_geom (pysisyphus.run.RunResult attribute), 330

ts_hessian() (in module pysisyphus.optimizers.guess_hessians), 298

ts_opt (pysisyphus.run.RunResult attribute), 330

TSHessianOptimizer (class in pysisyphus.tsoptimizers.TSHessianOptimizer), 118, 306

tunl() (in module pysisyphus.drivers.rates), 229

Turbomole (class in pysisyphus.calculators), 210

Turbomole (class in pysisyphus.calculators.Turbomole), 61, 187

type (pysisyphus.optimizers.gdiis.DIISResult attribute), 297

typed_prims (pysisyphus.intcoords.CartesianCoords property), 258

typed_prims (pysisyphus.intcoords.CartesianCoords.CartesianCoords property), 239

typed_prims (pysisyphus.intcoords.Coords.CoordSys property), 239

typed_prims (pysisyphus.intcoords.RedundantCoords property), 262

typed_prims (pysisyphus.intcoords.RedundantCoords.RedundantCoords property), 45, 246

typed_prims (pysisyphus.intcoords.setup.CoordInfo attribute), 254

U

U (pysisyphus.intcoords.DLC property), 258

U (pysisyphus.intcoords.DLC.DLC property), 240

unlink() (pysisyphus.calculators.IPIServer method), 203

unlink() (pysisyphus.calculators.IPIServer.IPIServer method), 172

unscaled_velocity_distribution() (in module pysisyphus.dynamics.helpers), 236

unweight_mw_hessian() (pysisyphus.Geometry.Geometry method), 42, 316

unweight_vec() (pysisyphus.irc.IRC.IRC method), 126, 271

update() (pysisyphus.irc.DWI.DWI method), 267

update_adapt_thresh() (pysisyphus.cos.AdaptiveNEB.AdaptiveNEB method), 112, 215

update_hessian() (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer method), 89, 287

update_internals() (in module pysisyphus.intcoords.update), 256

update_mw_down_step() (pysisyphus.irc.ModeKill method), 277

update_mw_down_step() (pysisyphus.irc.ModeKill.ModeKill method), 274

update_orientation() (pysisyphus.calculators.Dimer method), 199

update_orientation() (pysisyphus.calculators.Dimer.Dimer method), 77, 122, 165

update_springs() (pysisyphus.cos.NEB.NEB method), 111, 220

update_trust_radius() (pysisyphus.optimizers.HessianOptimizer.HessianOptimizer method), 89, 287

- `update_ts_mode()` (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer method), 119, 307
- `update_y()` (in module pysisyphus.calculators.ConicalIntersection), 163
- `using()` (in module pysisyphus.testing), 332
- ## V
- `V_dimer()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
- `V_harmonic()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
- `V_LEPS()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
- `V_tot()` (pysisyphus.calculators.LEPSEExpr.LEPSEExpr method), 173
- `VALID_CDDS` (pysisyphus.calculators.OverlapCalculator.OverlapCalculator attribute), 54, 182
- `valid_coord_types` (pysisyphus.cos.ChainOfStates.ChainOfStates attribute), 110, 217
- `valid_diis_direction()` (in module pysisyphus.optimizers.gdiis), 298
- `valid_displs` (pysisyphus.irc.IRC.IRC attribute), 126, 271
- `VALID_KEYS` (pysisyphus.calculators.OverlapCalculator.OverlapCalculator attribute), 54, 182
- `valid_requests` (pysisyphus.calculators.SocketCalc.SocketCalc attribute), 185
- `valid_updates` (pysisyphus.tsoptimizers.TSHessianOptimizer.TSHessianOptimizer attribute), 119, 307
- `VALID_XY` (pysisyphus.calculators.OverlapCalculator.OverlapCalculator attribute), 54, 182
- `value()` (pysisyphus.dynamics.colvars.Colvar method), 232
- `value()` (pysisyphus.dynamics.colvars.CVBend method), 232
- `value()` (pysisyphus.dynamics.colvars.CVDistance method), 232
- `value()` (pysisyphus.dynamics.colvars.CVTorsion method), 232
- `value()` (pysisyphus.dynamics.Gaussian.Gaussian method), 231
- `vdw_radii` (pysisyphus.Geometry.Geometry property), 42, 316
- `vdw_volume()` (pysisyphus.Geometry.Geometry method), 42, 316
- `vec` (pysisyphus.irc.initial_displ.ThirdDerivResult attribute), 275
- `verbose_bond_difference()` (in module pysisyphus.intcoords.helpers), 252
- `verify_chkfiles()` (pysisyphus.calculators.Calculator.Calculator method), 52, 161
- `view_cdd_cube()` (in module pysisyphus.wrapper.jmol), 308
- `volume_for_density()` (in module pysisyphus.pack), 325
- ## W
- `w` (pysisyphus.dynamics.Gaussian.Gaussian property), 232
- `weight()` (pysisyphus.intcoords.Primitive.Primitive method), 244
- `weight_function()` (in module pysisyphus.drivers.afir), 224
- `wf_overlap()` (pysisyphus.calculators.WFOWrapper.WFOWrapper method), 189
- `wf_overlap()` (pysisyphus.calculators.WFOWrapper2.WFOWrapper2 method), 190
- `WFOWrapper` (class in pysisyphus.calculators.WFOWrapper), 189
- `WFOWrapper2` (class in pysisyphus.calculators.WFOWrapper2), 189
- `wigner_corr()` (in module pysisyphus.drivers.rates), 280
- `without_hydrogens()` (pysisyphus.Geometry.Geometry method), 42, 316
- `wolfe_condition()` (pysisyphus.line_searches.LineSearch.LineSearch method), 279
- `wrap_stdin()` (in module pysisyphus.wrapper.mwfn), 309
- `write_cycle_to_file()` (pysisyphus.optimizers.Optimizer.Optimizer method), 87, 293
- `write_fragment_geoms()` (pysisyphus.calculators.AFIR method), 194
- `write_fragment_geoms()` (pysisyphus.calculators.AFIR.AFIR method), 73, 154
- `write_geoms_to_trj()` (in module pysisyphus.xyzloader), 334
- `write_geoms_to_trj()` (pysisyphus.stochastic.Pipeline.Pipeline method), 303
- `write_image_trjs()` (pysisyphus.optimizers.Optimizer.Optimizer method), 87, 293
- `write_mdrestart()` (pysisyphus.calculators.XTB method), 214
- `write_mdrestart()` (pysisyphus.calculators.XTB.XTB method), 68, 192

`write_results()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 87, 293

`write_to_out_dir()` (*pysisyphus.optimizers.Optimizer.Optimizer* method), 87, 293

X

`x` (*pysisyphus.calculators.ConicalIntersection.CIQuantities* attribute), 162

`x` (*pysisyphus.optimizers.poly_fit.FitResult* attribute), 300

`XTB` (class in *pysisyphus.calculators*), 212

`XTB` (class in *pysisyphus.calculators.XTB*), 66, 190

`xtb_hessian()` (in module *pysisyphus.optimizers.guess_hessians*), 298

Y

`y` (*pysisyphus.calculators.ConicalIntersection.CIQuantities* attribute), 162

`y` (*pysisyphus.optimizers.poly_fit.FitResult* attribute), 300

Z

`zero_crossings()` (in module *pysisyphus.peakdetect*), 328

`zero_crossings_sine_fit()` (in module *pysisyphus.peakdetect*), 328

`zero_fixed_vector()` (*pysisyphus.cos.ChainOfStates.ChainOfStates* method), 110, 217

`zero_frozen_forces()` (*pysisyphus.Geometry.Geometry* method), 42, 316

`ZeroStepLength`, 297

`ZLine` (class in *pysisyphus.io.zmat*), 266

`zmat_from_fn()` (in module *pysisyphus.io.zmat*), 266

`zmat_from_str()` (in module *pysisyphus.io.zmat*), 266

`zoom()` (*pysisyphus.line_searches.StrongWolfe* method), 281

`zoom()` (*pysisyphus.line_searches.StrongWolfe.StrongWolfe* method), 280