torsiondrive Documentation

Release 0.9.6

torsiondrive

Sep 11, 2021

Getting Started

TorsionDrive is a software package for scanning the potential energy surface of molecules along the torsional degrees of freedom.

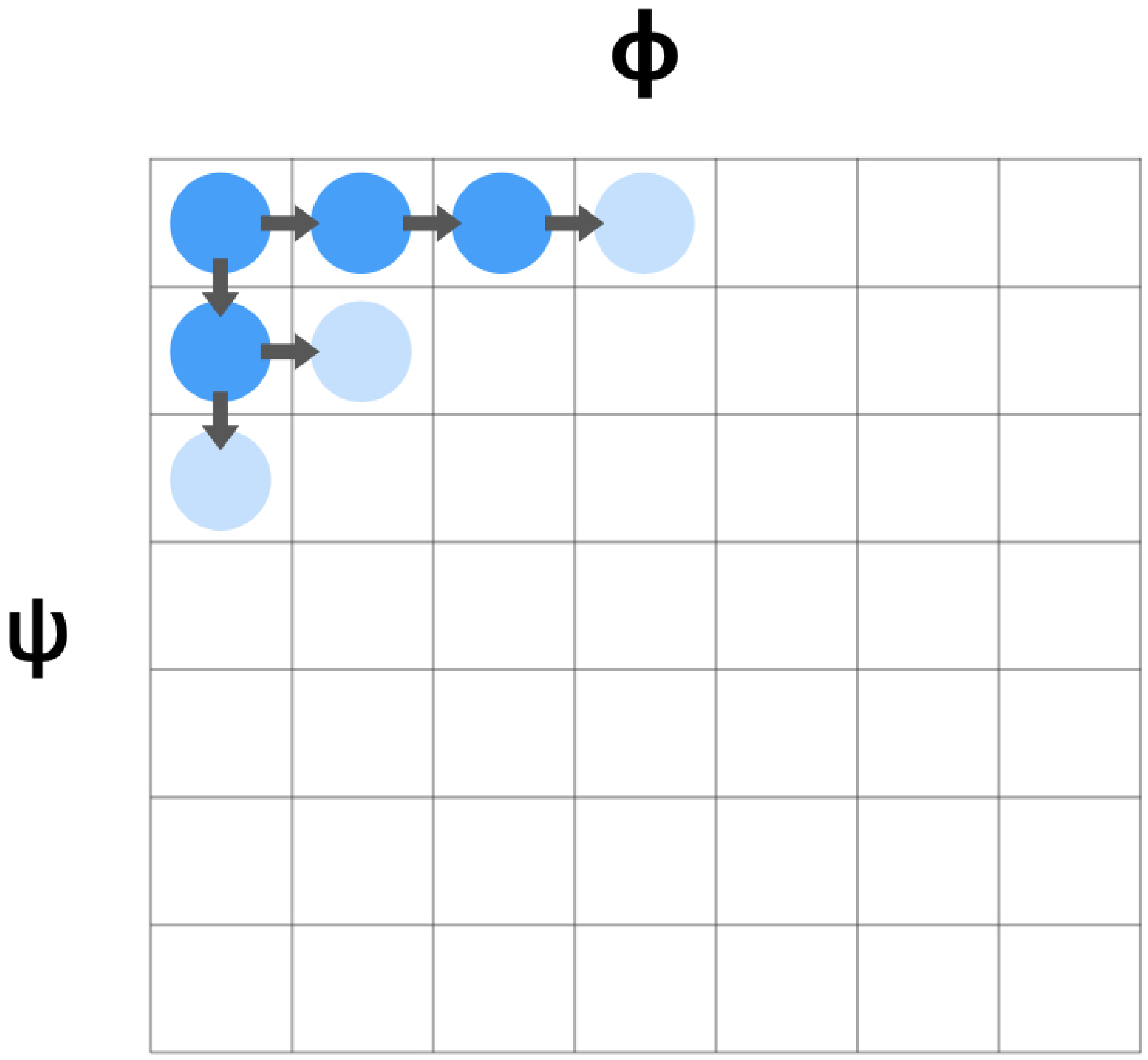
CHAPTER 1

Method

A N-dimensional torsion scan can be visualized as filling a N-dimensional grid of dihedral angles.

- Each grid point represents a unique combination of dihedral angles, (ψ,ϕ)
- The value of each grid point is the energy of the structure that has the (ψ,ϕ) torsion angles

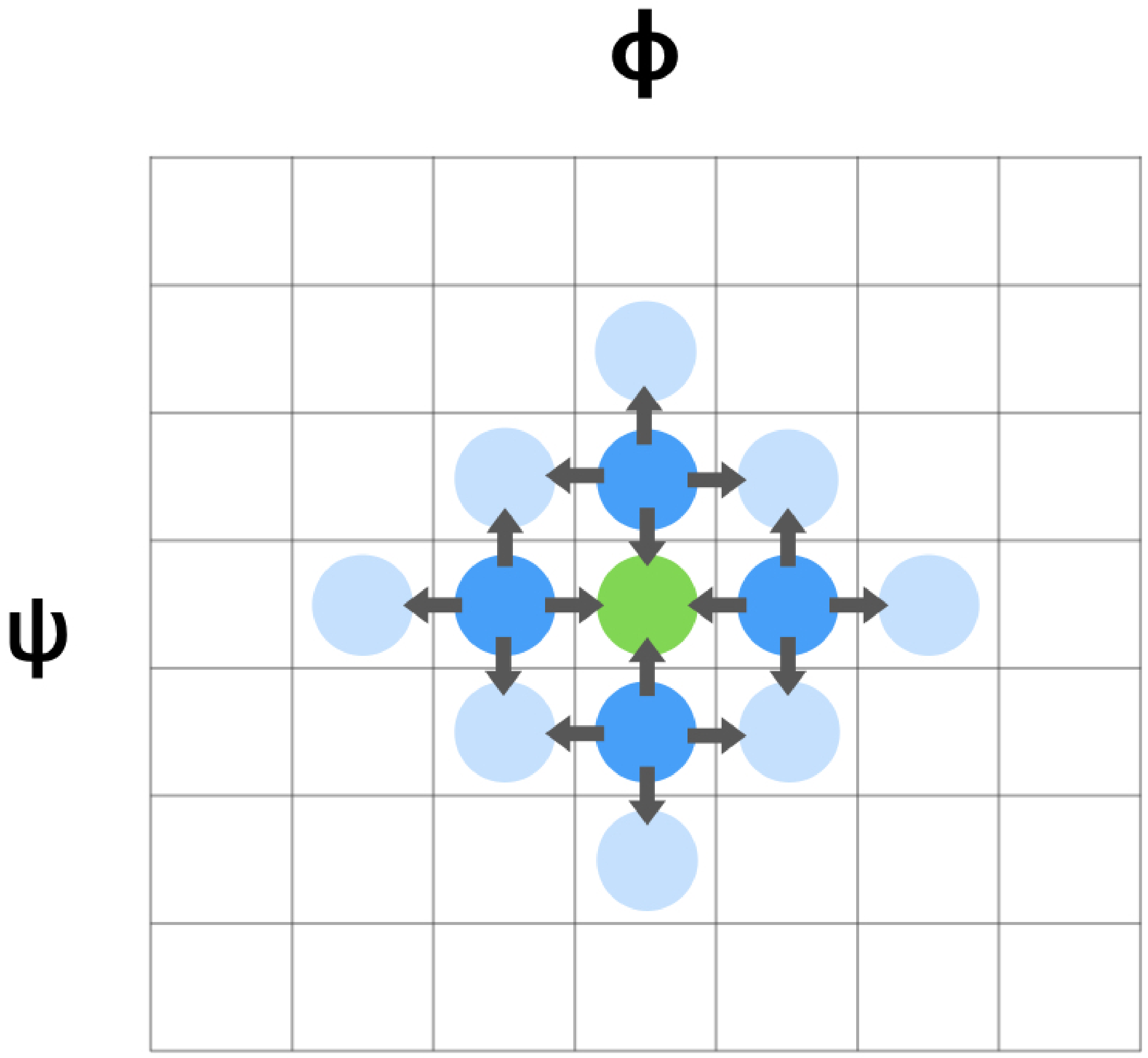
For example, to perform a 2-D torsion scan, one may proceed with a "regular scan" – scan a leading dimension then go through each value of the second dimension.



The arrows in the diagram represent “constrained optimizations”, starting from one structure matches the torsion angels of the grid, ending at a new structure with torsion angles equal to a neighboring grid.

In comparison, a “torsiondrive” scan fills the dihedral grid by “wavefront propagation”.

- Starting with one or more “seed” geometries, each optimize to their closest grid point.
- All initial grid points are set as “active”.
- All “active” grid points create constrained optimizations towards each of its neighboring grid points.
- The neighboring grid points gets their first energy, and they’re set to “active”, starting new optimizations.
- When a grid point gets an energy lower than it’s previous minimum, it’s set to “active”.
- Repeat the propagation until there are no more “active” grid points.


ϕ
ψ

CHAPTER 2

Scaling

The total number of constrained optimizations is approximately

$$2 \times N_{dim} \times N_{grid}$$

where N_{dim} is the number of dimensions. The term N_{grid} is the total number of grid points in the scan, which is the product of the number of grid points of each dimension.

$$N_{grid} = \prod_{i}^{N_{dim}} n_i$$

Extra calculations will be needed for the initial optimization, and grid points activated by a lower energy found.

CHAPTER 3

Index

Getting Started

3.1 Install TorsionDrive

You can install *torsiondrive* with `conda`, with `pip`, or by installing from source.

3.1.1 Conda

You can update torsiondrive using conda:

```
conda install torsiondrive -c conda-forge
```

This installs torsiondrive and its dependancies.

The torsiondrive package is maintained on the conda-forge channel.

3.1.2 Pip

To install torsiondrive with `pip`

```
pip install torsiondrive
```

3.1.3 Install from Source

To install qcfractal from source, clone the repository from github:

```
git clone https://github.com/lpwgroup/torsiondrive.git
cd torsiondrive
python setup.py install
```

or use pip for a local install:

```
pip install -e .
```

It is recommended to setup a testing environment using conda. This can be accomplished by:

```
cd torsiondrive
python devtools/scripts/conda_env.py -n=td_test -p=3.7 devtools/conda-envs/psi.yaml
```

3.1.4 Test

Test torsiondrive with py.test:

```
cd torsiondrive
py.test
```

3.1.5 Installation of cctools

The library cctools.work_queue is utilized to provide distributed computing feature in TorsionDrive. https://github.com/cooperative-computing-lab/cctools

Installation of cctools is provided separately. A convenient bash script has been made to simplify the process:

```
$bash torsiondrive/devtools/travis-ci/install-cctools.sh
```

3.2 Run TorsionDrive

3.2.1 Using the Command Line

Once installed, you can start torsiondrive scans from command line:

```
$ torsiondrive-launch -h
usage: torsiondrive-launch [-h] [--init_coords INIT_COORDS]
                        [-g [GRID_SPACING [GRID_SPACING ...]]]
                        [-e {qchem,psi4,terachem}] [-c CONSTRAINTS]
                        [--native_opt] [--energy_thresh ENERGY_THRESH]
                        [--energy_upper_limit ENERGY_UPPER_LIMIT]
                        [--wq_port WQ_PORT] [--zero_based_numbering] [-v]
                        inputfile dihedralfile

Potential energy scan of dihedral angle from 1 to 360 degree

positional arguments:
inputfile             Input template file for QMEngine. Geometry will be
                        used as starting point for scanning.
dihedralfile          File defining all dihedral angles to be scanned.
```

(continues on next page)

(continued from previous page)

```
optional arguments:
-h, --help            show this help message and exit
--init_coords INIT_COORDS
                        File contain a list of geometries, that will be used
                        as multiple starting points, overwriting the geometry
                        in input file. (default: None)
-g [GRID_SPACING [GRID_SPACING ...]], --grid_spacing [GRID_SPACING [GRID_SPACING ...]]
                        Grid spacing for dihedral scan, i.e. every 15 degrees,
                        multiple values will be mapped to each dihedral angle
                        (default: [15])
-e {qchem,psi4,terachem}, --engine {qchem,psi4,terachem}
                        Engine for running scan (default: psi4)
-c CONSTRAINTS, --constraints CONSTRAINTS
                        Provide a constraints file in geomeTRIC format for
                        additional freeze or set constraints (geomeTRIC or
                        TeraChem only) (default: None)
--native_opt          Use QM program native constrained optimization
                        algorithm. This will turn off geomeTRIC package.
                        (default: False)
--energy_thresh ENERGY_THRESH
                        Only activate grid points if the new optimization is
                        <thre> lower than the previous lowest energy (in
                        a.u.). (default: 1e-05)
--energy_upper_limit ENERGY_UPPER_LIMIT
                        Only activate grid points if the new optimization is
                        less than <thre> higher than the global lowest energy
                        (in a.u.). (default: None)
--wq_port WQ_PORT     Specify port number to use Work Queue to distribute
                        optimization jobs. (default: None)
--zero_based_numbering
                        Use zero_based_numbering in dihedrals file. (default:
                        False)
-v, --verbose         Print more information while running. (default: False)
```

3.2.2 Using the API (advanced)

An API interface of torsiondrive is provided for interfacing with QCFractal servers. The main difference of the API method is that the API is designed as a “service”, which generates one iteration of constrained optimizations each time.

```
$ torsiondrive-api -h
usage: torsiondrive-api [-h] [-v] statefile

Take a scan state and return the next set of optimizations

positional arguments:
statefile      File contains the current state in JSON format

optional arguments:
-h, --help     show this help message and exit
-v, --verbose  Print more information while running. (default: False)
```

A json file containing the scan options and the “current state” of torsion scan is passed to the API, then the API program will reproduce the **entire** torsion scan from the beginning, until some new optimiations are needed.

The new optimiations will be returned also in JSON format. If the scan is finished, the return will be empty.

Examples

- *TorsionDrive Examples*

3.3 TorsionDrive Examples

Example runs of torsiondrive can be found in repository https://github.com/lpwgroup/torsiondrive_examples

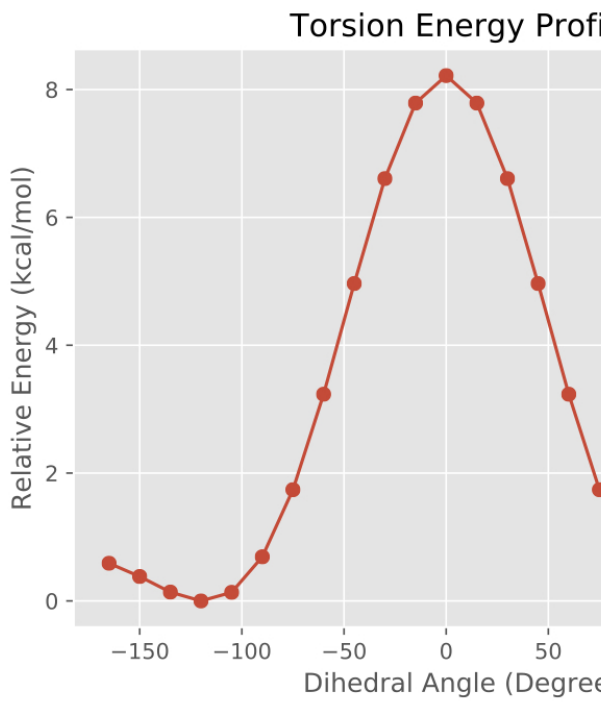
3.3.1 1-D Examples

Example input, output and running commands can be found in `torsiondrive_examples/examples/hooh-1d`, including

- Quantum chemistry program used as engine: QChem, TeraChem, Psi4
- Optimizer: geomeTRIC or the built-in optimizer from the QM program
- Distributed: run optimization locally or distribute them using `cctools.work_queue`

geomeTRIC + Psi4

- Location: torsiondrive_examples/examples/hooh-1d/psi4/run_local/geomeTRIC/
- Run command: `torsiondrive-launch input.dat dihedrals.txt -g 15 -e psi4 -v`
- Output log: scan.log



- **Energy plot can be generated using `torsiondrive-plot1d`**

3.3.2 2-D Examples

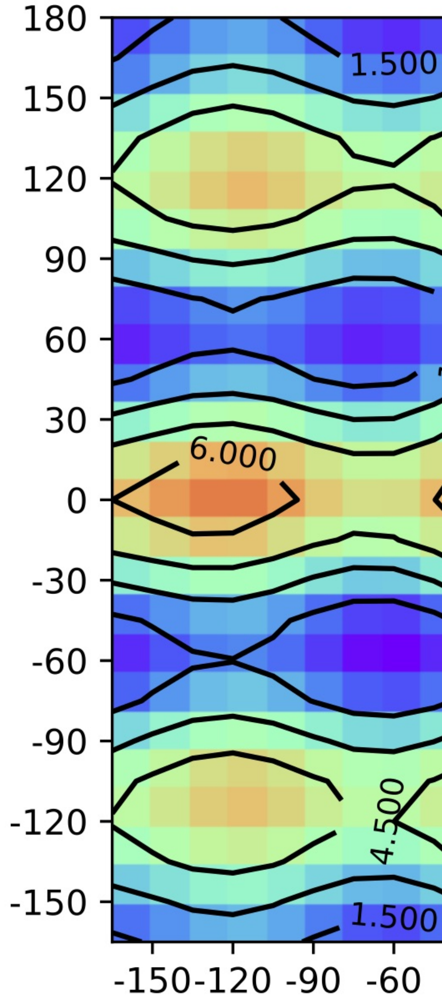
2-D torsion scans are relatively expensive. Therefore it is recommended to use a cheap QM method, or use the distributed method calling *cctools.work_queue*.

geomeTRIC + Psi4 distributed

- Location: torsiondrive_examples/examples/propanol-2d/work_queue_qchem_geomeTRIC/
- Run command: `torsiondrive-launch qc.in dihedrals.txt -g 15 -e qchem --wq_port 50124 -v 2>worker.log`
- Two dihedrals are specified in input `dihedrals.txt` to create a 2-D scan:

```
# dihedral definition by atom indices starting from 1
# i      j      k      l
  1      2      7      11
  2      7      11     12
```

- Output log: scan.log



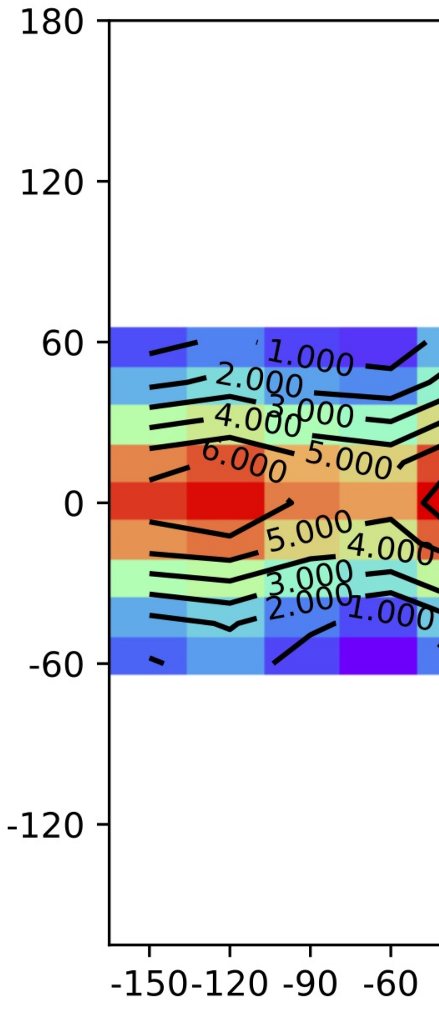
- **Energy heatmap can be generated using `torsiondrive-plot2d`**

range limited scan

- Location: torsiondrive_examples/examples/range_limited_split/
- Run command: `torsiondrive-launch qc.in dihedrals.txt -g 15 30 -e qchem -v --wq_port 50124 2>worker.log`
- Input dihedrals.txt:

```
# dihedral definition by atom indices starting from 1
# i     j     k     l      (range_low)     (range_high)
  1     2     7     11         -60              60
  2     7     11    12         150             330
```

- Output log: scan.log



- **Energy heatmap can be generated using `torsiondrive-plot2d`**

Developer Documentation

Contains in-depth developer documentation.

- *TorsionDrive API*

3.4 TorsionDrive API

3.4.1 dihedral_scanner

class torsiondrive.dihedral_scanner.**DihedralScanner**(*engine*, *dihedrals*, *grid_spacing*, *init_coords_M=None*, *energy_decrease_thresh=None*, *dihedral_ranges=None*, *energy_upper_limit=None*, *extra_constraints=None*, *verbose=False*)

DihedralScanner class is designed to create a dihedral grid, and fill in optimized geometries and energies into the grid, by running wavefront propagations of constrained optimizations

Parameters

engine: QMEngine() instance An QMEngine object, e.g. EnginePsi4, EngineQChem or EngineTerachem

dihedrals: List[(d1, d2, d3, d4), ..] list of dihedral index tuples (d1, d2, d3, d4). The length of list determines the dimension of the grid i.e. dihedrals = [(0,1,2,3)] –> 1-D scan, dihedrals = [(0,1,2,3),(1,2,3,4)] –> 2-D Scan

grid_spacing: Int Distance (in Degrees) between grid points, correspond to each dihedral, every value must be a divisor of 360

init_coords_M: geometric.molecule.Molecule() instance A Molecule constains a series of initial geometries to start with

energy_decrease_thresh: Float The threshold of the smallest energy decrease amount to trigger activating optimizations from grid point.

dihedral_ranges: List[(lower, upper), ..] A list of dihedral range limits as a pair (lower, upper), each range corresponds to the dihedrals in input.

energy_upper_limit: Float or None The threshold if the energy of a grid point that is higher than the current global minimum, to start new optimizations, in unit of a.u. i.e. if energy_upper_limit = 0.05, current global minimum energy is -9.9 , then a new task starting with energy -9.8 will be skipped.

extra_constraints: Dict A nested dictionary specifing extra constraints in geomeTRIC format. Details in extra_constraints.py

verbose: bool let methods print more information when running

build_dihedral_mask(*dihedral_ranges*)

Build a dihedral mask based on specified ranges

Parameters

dihedral_ranges: List[(lower: Int, upper: Int), ..] The range limits corresponding to each dihedral angle A full dihedral range is [-180, 180] The upper limit up to 360 is supported for the purpose of specifying range limits crossing the boundary, e.g. [80, 240], which effectively become [-180, 120] + [80, 180]

Returns

dihedral_mask: List[set(), ..] The dihedral mask is a list of sets, each set contains all available values for one dihedral angle

Notes

This function should be called after self.setup_grid()

create_tmp_folder()
Create an empty tmp folder structure, save the paths for each grid point into self.tmp_folder_dict

Examples

self.tmp_folder_dict = {(30,-70): "opt_tmp/gid_+030_-070", ..}

draw_ansi_image()
Return a string with ANSI colors showing current running status

draw_ramachandran_plot()
Return a string of Ramachandran plot showing current running status

finish()
Write qdata.txt and scan.xyz file based on converged scan results

get_dihedral_id(*molecule*, *check_grid_id=None*)
Compute the closest grid ID for molecule (only first frame) If check_grid_id is given, will perform a check if the computed dihedral_values are close to the grid_id provided If the check is not passed, this function will return None

get_new_scr_folder(*grid_id*)
create a job scratch folder inside tmp_folder_dict[grid_id] name starting from '1', and will use larger numbers if exist return the new folder name that's been created

grid_full_neighbors(*grid_id*)
Take a center grid id, return all the neighboring grid ids, in all dimensions

grid_neighbors(*grid_id*)
Take a center grid id, return all the neighboring grid ids, in each dimension

launch_constrained_opt(*molecule*, *grid_id*)
Called by launch_opt_jobs() to launch one opt job in a new scr folder Return the new folder path

launch_opt_jobs()
Launch constrained optimizations for molecules in opt_queue Tasks current opt_queue will be popped in order. If a task exist in self.task_cache, the cached result will be checked, then put into self.current_finished_job_results Else, the task will be launched by self.launch_constrained_opt, and information is saved as self.running_job_path_info[job_path] = m, from_grid_id, to_grid_id

master()
The master function that calls all other functions. This function will run the following steps: 1. Launch a new set of jobs from self.opt_queue, add their job path to a dictionary 2. Check if any running job has finished 3. For each finished job, check if energy is lower than existing one, if so, add its neighbor grid points to opt_queue 4. Go back to the 1st step, loop until all jobs finished, indicated by opt_queue and running jobs both empty.

push_initial_opt_tasks()
Push a set of initial tasks to self.opt_queue A task is defined as (m, from_grid_id, to_grid_id) tuple, where geometry is stored in m

restore_task_cache()
Restore previous finished tasks from tmp folder. 1. Look into tmp folder and read scanner_settings.json, check if it matches current setting 2. Read the result pickle file from each leaf folder, into task_cache If successful, self.tmp_folder_dict will be initialized, same as self.create_tmp_folder(), and self.task_cache will be populated, with task caches, defined in this way:

self.task_cache = {(30,-60): {geo_key: (final_geo, final_energy, final_gradient, job_folder)}}

final_gradient will be None if it's not available.

save_task_cache(*job_path*, *m_init*, *m_final*)
Save a file containing the finished job information to a pickle file on disk. The format should be consistent with self.restore_task_cache()

setup_grid()
Set up grid ids, each as a tuple with size corresponding to grid dimension. i.e. 1-D: grid_ids = ((-165,), (-150,), … (180,)) 2-D: grid_ids = ((-165,-165), (-165,-150), … (180,180)) This function is called by the initializer.

self.grid_axes is also initialized, to be a full range of grid values for each dihedral, i.e., 1-D: grid_axes = [range(-165, 195, 15)] 2-D: grid_axes = [range(-165, 195, 15), range(-165, 195, 15)]

validate_task(*task*)
Validate a constrained optimization task before pushing to the queue. This is useful to limit the dihedrals into a range of interest.

Parameters

task: (m, from_grid_id, to_grid_id) A constrained optimization task

Returns

isValid: bool True if the task is valid

wait_extract_finished_jobs()
Interface with engine to check if any job finished. Will wait infinitely here until at least one job finished. The finished job paths will be removed from self.running_job_path_info. The finished job results (m, grid_id) will be checked, if the result geometry is not close enough to target grid id, the result will be ignored. Results passed the check will be added to self.current_finished_job_results.

torsiondrive.dihedral_scanner.**cross3**(*v1*, *v2*)
Quick convenient function to compute cross product betwee two 3-element vectors cross3: 326 ns | np.cross: 35.8 us

torsiondrive.dihedral_scanner.**dot3**(*v1*, *v2*)
Quick convenient function to compute dot product betwee two 3-element vectors dot3: 231 ns | np.dot: 745 ns

torsiondrive.dihedral_scanner.**get_geo_key**(*coords*)
Convert an numpy array of xyz coordinate to a hashable object, keeping 0.001 precision This function has the limitation that 3.1999 and 3.2000 will produce different results due to the limitation of float point representation.

torsiondrive.dihedral_scanner.**measure_dihedrals**(*molecule*, *dihedral_list*, *check_linear=True*, *check_bonded=True*)
Measure dihedral values from molecule coordinates.

Parameters

molecule: geometric.molecule.Molecule The molecule object that contains atom coordinates. Only the first frame will be used.

dihedral_list: List[List[Int]] A list of dihedrals to compute their value. Each diedral is represented by a list of tuple of four integers, each is a 0-based atom index.

check_linear: Bool If True, will check if i-j-k or j-k-l angles in each dihedral is close to linear (> 165 degree), print a warning if found.

check_bonded: Bool If True, will check if all i-j, j-k, k-l are bonded for each dihedral, print a warning if not.

`torsiondrive.dihedral_scanner.`**`norm3`**(*vec3*)

Quick convenient function to get the norm of a 3-element vector norm3: 475 ns | np.linalg.norm: 4.31 us

`torsiondrive.dihedral_scanner.`**`normalize_dihedral`**(*d*)

Normalize any number to the range (-180, 180], including 180

3.4.2 qm_engine

`class` `torsiondrive.qm_engine.`**`EngineBlank`**(*input_file=None*, *work_queue=None*, *native_opt=False*, *extra_constraints=None*)

A blank engine only used in testing

3.4.3 extra_constraints

`torsiondrive.extra_constraints.`**`build_geometric_constraint_string`**(*constraints_dict*, *dihedral_idx_values=None*)

Build the geomeTRIC constraint string with constraints_dict and a set of dihedral_idx_values

Parameters

constraints_dict: Dict constraints dict built by make_constraints_dict() function

dihedral_idx_values: List[List[d1, d2, d3, d4, v]] A list containing the definition of dihedrals and their values Example: [(0,1,2,3,90.0), (1,2,3,4,100.0)]

Returns

constraints_string: string A string with multiple lines, to be used as the geomeTRIC constraints.txt

`torsiondrive.extra_constraints.`**`build_terachem_constraint_string`**(*constraints_dict*, *dihedral_idx_values=None*)

Build the TeraChem constraint string with constraints_dict and a set of dihedral_idx_values

Parameters

constraints_dict: Dict constraints dict built by make_constraints_dict() function

dihedral_idx_values: List[List[d1, d2, d3, d4, v]] A list containing the definition of dihedrals and their values Example: [(0,1,2,3,90), (1,2,3,4,100)]

Returns

constraints_string: string A string with multiple lines, to be used as the TeraChem constraints format

`torsiondrive.extra_constraints.`**`check_conflict_constraints`**(*constraints_dict*, *dihedral_idxs*)

Utility function to check if any extra constraints in constraints_dict is conflict with the scanning dihedrals

`torsiondrive.extra_constraints.`**`make_constraints_dict`**(*constraints_string*)

Create an ordered dictionary with constraints specification, consistent with geomeTRIC

Parameters

constraints_string: str String-formatted constraint specification consistent with geomeTRIC constraints.txt

Returns

constraints_dict: dict A dictionary contains the definition of the extra constraints. The format is consistant with the JSON interface of geomeTRIC.

Notes

1. Only constraints of type "freeze" and "set" are supported, since extra "scan" is undefined with torsiondrive scan.
2. Four attributes are allowed to be constrained: 'distance', 'angle', 'dihedral', 'xyz'
3. The input string is one-indexed, the output dictionary is zero-indexed.
4. For "xyz", dashed inputs like "1-3,7-9" (no space) is allowed, and will be converted to [0,1,2,6,7,8].

Examples

```
>>> make_constraints_dict(r"$freeze\nxyz 1-3\n$set\nangle 2 1 5 110.0")
{
    'freeze': [{
        'type': 'xyz',
        'indices': [0, 1, 2]
    }],
    'set': [{
        'type': 'angle',
        'indices': [1, 0, 4],
        'value': 110.0}]
}
```

Python Module Index

Index