TorsiFlex v2021.3

A program for the conformational search in flexible molecules

David Ferro-Costas

Antonio Fernández-Ramos

Universidade de Santiago de Compostela (SPAIN)

November 19, 2021

About `TorsiFlex` License

MIT/X11 License

Copyright (c) 2021 David Ferro Costas (david.ferro@usc.es) and Antonio Fernández Ramos (qf.ramos@usc.es)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

November 2021

Contents

1 TorsiFlex software **5**

1.1 About TorsiFlex 5

1.2 Terms of use 5

1.3 TorsiFlex software contents 5

1.4 Setting up TorsiFlex 6

1.5 Electronic structure software 6

1.6 How to cite 6

1.7 Version history 6

1.7.1 List of changes in version 2021.3 6

1.7.2 List of changes in version 2021.2 7

2 The algorithm in a nutshell **9**

2.1 The low-level search 10

2.2 The high-level reoptimization 10

2.3 The validation tests 10

3 TorsiFlex options **13**

3.1 Input generation and Gaussian templates 15

3.2 Low-level search 16

3.3 High-level re-optimization 17

3.4 Conformer analysis 17

3.5 Torsional anhamonicity 17

4 The reference Z-matrix **19**

5 The TorsiFlex input file **21**

5.1 Keywords 21

5.1.1 System-specific keywords 21

5.1.2 Keywords associated with the target torsions 21

5.1.3 Keywords associated with the search procedure 22

5.1.4 Keywords associated with the HL re-optimization 22
5.1.5 Keywords associated with the validation tests 22
5.1.6 Keywords associated with the calculations with Gaussian . . . 23
5.1.7 Keywords associated with the partition functions 24
5.1.8 Keywords associated with the storage of conformers 25
5.2 The file with preconditioned angles: `pcfile` 27
5.3 Specifying a domain 27

6 Some tips 29
6.1 Low-level calculations at the HF level 29
6.2 Systems with many LL conformers 29
6.3 Focusing on low energy conformers 29
6.4 Dealing with similar molecules 30

7 Worked examples 33
7.1 WE1: conformers of n-butanol 34
7.2 WE2: conformers of L-alanine 37
7.3 WE3: conformers of proline 40
7.4 WE4: conformers of a transition state 43

8 Example files 47

Bibliography 54

1. `TorsiFlex` software

This document is a guide to use `TorsiFlex`, a program designed to find the conformers of flexible molecules using a combination of a preconditioned variation of the torsions and a Monte Carlo search.

The example files cited along the document are listed in Chapter 8.

1.1 About `TorsiFlex`

- Programming language: Python 3
- Operating systems: Linux and macOS

`TorsiFlex` is an user-friendly code written in Python 3 designed to seek the conformers of a given molecule by adopting a combined low-level/high-level (LL/HL) methodology.[1,2] `TorsiFlex` is part of a bigger project where other programs of interest are included: `the Cathedral package`. We encourage the user to visit our GitHub webpage to know about the software we are developing in our group.[3]

1.2 Terms of use

`TorsiFlex` is free software under MIT license. We refer to the GitHub webpage for more details about the license:

https://github.com/cathedralpkg/torsiflex

1.3 `TorsiFlex` software contents

The source code is located in the `src/` folder and contains the following files/subdirectories:

- `torsiflex.py`: the main executable file;
- `common/`: modules with diverse functions of general use;
- `modtorsiflex/`: modules that deal with (i) `TorsiFlex` options, (ii) electronic structure softwares, (iii) reading/writing of files.

1.4 Setting up TorsiFlex

TorsiFlex can be executed using the python3 executable:

```
  >> python3 path_to_torsiflex/torsiflex.py
```

or it can be directly executed provided that execution permissions are granted:

```
  >> chmod u+x path_to_torsiflex/torsiflex.py
```

and as long as the path to the python3 executable is /usr/bin/python3. If this is not the case, the user can modify the *shebang* line of torsiflex.py to define the path of the Python interpreter:

```
#!/usr/bin/python3
```

1.5 Electronic structure software

At this stage, TorsiFlex can only perform electronic structure calculations using the Gaussian software (versions 09 and 16).[4,5] The path to the Gaussian executable is read from an environment variable denoted as GauExe, which can be defined and exported in the .bashrc file. For example:

```
export GauExe="/home/programs/Gaussian/g09"
```

1.6 How to cite

The following publications must be cited in any work presenting results obtained with TorsiFlex:

- D. Ferro-Costas and A. Fernández-Ramos, *Front. Chem.*, 2020, **8**:16 [A Combined Systematic-Stochastic Algorithm for the Conformational Search in Flexible Acyclic Molecules].

- D. Ferro-Costas, I. Mosquera-Lois and A. Fernández-Ramos, *J. Cheminformatics*, 2021, **13**:100 [TorsiFlex: an automatic generator of torsional conformers. Application to the twenty proteinogenic amino acids].

1.7 Version history

1.7.1 List of changes in version 2021.3

- New keywords are available: nprocll, nprochl, memll, memhl, skipcon, ifqrangeLL, and ifqrangeHL.

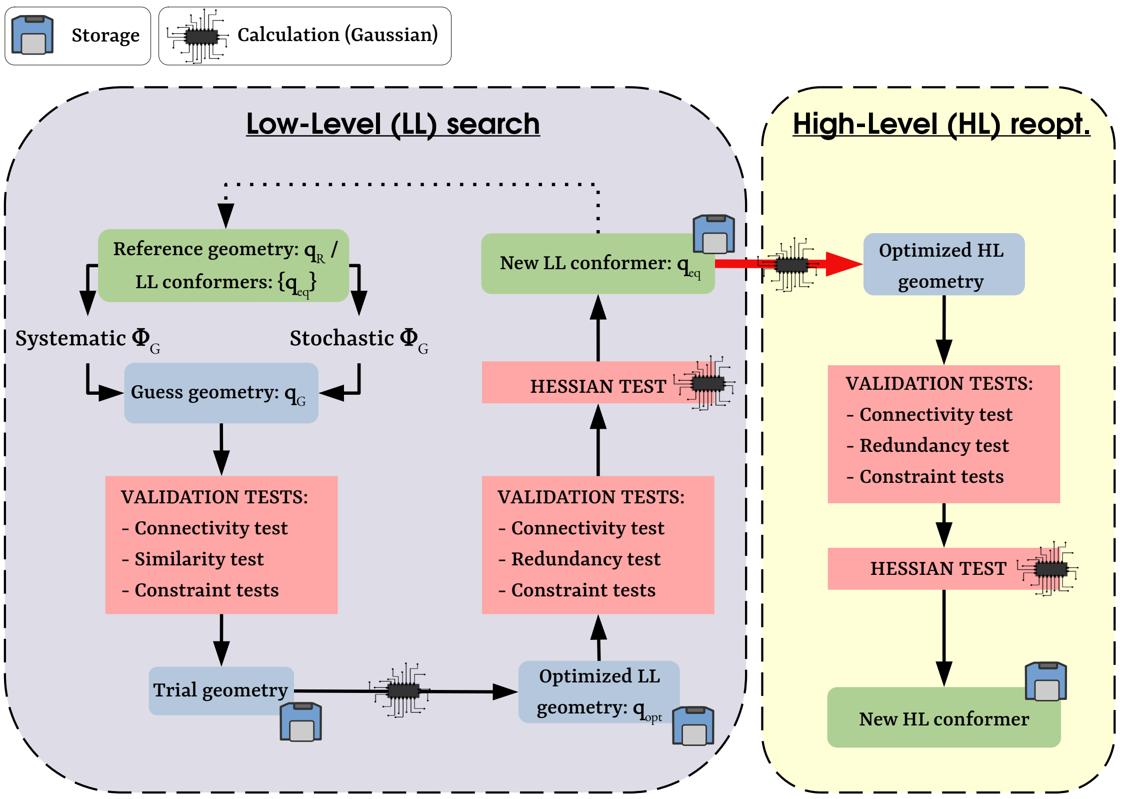
- A bug related to the projection of the rotational degrees of freedom has been fixed.
- Other minor changes.

1.7.2 List of changes in version 2021.2

- Option `--syst` is now renamed as `--prec`
- A bug related to the calculation of the rovibrational partition function for transition state structures has been fixed.

2. The algorithm in a nutshell

This Chapter briefly describes the algorithm implemented in `TorsiFlex`, which is also schematized in the flowchart of Figure 2.1. The first part of the algorithm consists on a low-level search of conformers (section 2.1). The second part comprises a geometry re-optimization at a higher level of calculation (section 2.2). We refer to our previous works[1,2] for a more detailed explanation.



Storage
Calculation (Gaussian)
Low-Level (LL) search
High-Level (HL) reopt.
Reference geometry: q_R / LL conformers: $\{q_{eq}\}$
Systematic Φ_G
Stochastic Φ_G
Guess geometry: q_G
VALIDATION TESTS: - Connectivity test - Similarity test - Constraint tests
Trial geometry
Optimized LL geometry: q_{opt}
VALIDATION TESTS: - Connectivity test - Redundancy test - Constraint tests
HESSIAN TEST
New LL conformer: q_{eq}
Optimized HL geometry
VALIDATION TESTS: - Connectivity test - Redundancy test - Constraint tests
HESSIAN TEST
New HL conformer

Figure 2.1: Flowchart of the search and reoptimization processes implemented in `TorsiFlex`.

2.1 The low-level search

`TorsiFlex` requires a reference geometry, $\mathbf{q}_{\mathrm{R}}$, which is given in the Z-matrix format. This geometry explicitly includes the K proper torsions of interest, which can be collected into a K-dimensional torsion vector, $\mathbf{\Phi}^{\mathrm{R}}$. The replacement of $\mathbf{\Phi}^{\mathrm{R}}$ by a guess vector, $\mathbf{\Phi}^{\mathrm{G}}$, renders a new configuration or guess geometry, $\mathbf{q}_{\mathrm{G}}$.[*] There are two strategies for the generation of $\mathbf{\Phi}^{\mathrm{G}}$: preconditioned or stochastic. The former is based on chemically-intuitive dihedral angles,[†] whereas the latter is based on the random generation of K integers between 0 and 360 (degrees).

Each guess Z-matrix, $\mathbf{q}_{\mathrm{G}}$, is validated through a series of tests. Guess geometries that fail the tests are discarded, whereas those that pass them turn into trial structures that will be optimized geometrically. In this manner, unnecessary calculations are avoided, accelerating the search process.

The resulting optimized geometry, $\mathbf{q}_{\mathrm{opt}}$, is validated through a new set of tests and, if all of them result positive, the Hessian matrix is calculated in order to confirm that we are dealing with a conformer of the system (Hessian test).

2.2 The high-level reoptimization

The search task described in the previous section is intended to be carried out at an inexpensive level of calculation (low-level, LL), in order to reduce the computational cost and speed up the location of conformers.

`TorsiFlex` can use the LL conformers as trial geometries for more accurate electronic structure calculations (high-level, HL). The HL optimized structures are validated by a new set of tests. If all of them result positive, the Hessian test is carried out by calculating the Hessian matrix. If this test is also positive, the HL optimized geometry corresponds to a new conformer.

2.3 The validation tests

The following tests are implemented in `TorsiFlex`:

- **The connectivity test**: positive if the connectivity of the guess/optimized structure corresponds to that of the reference Z-matrix (`i.e.` this test asserts that we are dealing with the same constitutional isomer).

[*]The Z-matrix of the conformers located along the search process is generally more convenient than the reference one. For this reason, whenever possible, `TorsiFlex` uses the Z-matrix of the conformer with the closest torsion vector to $\mathbf{\Phi}^{\mathrm{G}}$ instead of resorting to the reference Z-matrix, speeding up the geometry optimization.

[†]By default, the dihedral angles for the expected gauche (60° and 300°) and anti (180°) arrangements in hydrocarbons.

- **The similarity test**: the torsion vector of a guess structure, $\boldsymbol{\Phi}^{\mathrm{G}}$, is compared to a pool of stored torsion vectors from previous iterations, $\{\boldsymbol{\Phi}^{\mathrm{st}}\}$; if $\boldsymbol{\Phi}^{\mathrm{G}}$ falls outside of the domain associated to each stored vector, *i.e.*:

  $$\forall\, p, \quad \exists\, \tau : |(\boldsymbol{\Phi}^{\mathrm{G}})_\tau - (\boldsymbol{\Phi}^{\mathrm{st}}_p)_\tau| > d \tag{2.1}$$

  then the test is positive. In the previous equation, d defines the domain of each stored point and p and τ run over all stored points and over each target torsion, respectively.

- **The redundancy test**: compares the optimized torsion vector, $\boldsymbol{\Phi}^{\mathrm{opt}}$, against the pool of stored conformers, $\{\boldsymbol{\Phi}^{\mathrm{eq}}\}$. If $\boldsymbol{\Phi}^{\mathrm{opt}}$ is not stored in $\{\boldsymbol{\Phi}^{\mathrm{eq}}\}$, *i.e.* $\boldsymbol{\Phi}^{\mathrm{opt}} \not\subset \{\boldsymbol{\Phi}^{\mathrm{eq}}\}$, the test results positive. From a practical point of view this test is positive if:

  $$\forall\, p, \quad \exists\, \tau : |(\boldsymbol{\Phi}^{\mathrm{opt}})_\tau - (\boldsymbol{\Phi}^{\mathrm{eq}}_p)_\tau| > \epsilon \tag{2.2}$$

  where ϵ is a threshold (not greater than 2 degrees) that accounts for numerical errors.

- **The constraint tests**. For different reasons, the user may be interested in a certain set of conformers. For such cases, `TorsiFlex` accepts the definition of constraints based on the distance between a pair of atoms, the angle between a triad of atoms and the dihedral angle between four atoms. Specifically, `TorsiFlex` differenciates between two types of constraints:
  - *hard* constraints; the test is positive if all the specific requirements for the constraints are fulfilled.
  - *soft* constraints; the test is positive if one or more of the specific requirements for the constraints are fulfilled.

- **The Hessian test**: performs a test on the Hessian matrix of the optimized geometry. If all its vibrational frequencies are real, the test is positive. When dealing with the conformers of a transition state, the test results positive when all the vibrational frequencies are real but one. Notice that the calculation of the Hessian matrix is time-demanding.

In Figure 2.1 we illustrate the tests that are used to validate trial and optimized structures.

3. TorsiFlex options

The list of available options in TorsiFlex can be displayed by executing:

```
  >> path_to_torsiflex/torsiflex.py --help
```

or just:

```
  >> path_to_torsiflex/torsiflex.py -h
```

The corresponding help message is shown below.

```
======================
 Welcome to TORSIFLEX
======================


Current version: TorsiFlex v2021.3 (2021-Nov-19)

Description:

  A program to seek the conformers of flexible molecules
  by means of a combined preconditioned-stochastic algorithm.
  The located conformers, calculated with a low-level (LL)
  electronic structure method, can be re-optimized using
  a high-level (HL) method.

Main authors:

  Dr.   David Ferro-Costas
  Prof. Antonio Fernandez-Ramos

  Centro Singular de Investigacion en Quimica Bioloxica
  e Materiais Moleculares (CIQUS), Universidade de
  Santiago de Compostela, Galicia, Spain

Execution:

  torsiflex.py [--help /-h]  [--version/-v]  [--inp/--input]
               [--prec    ]  [--stoc      ]  [--hlopt      ]
               [--msho    ]  [--mstor     ]  [--regen      ]

-------------------
  Program options
-------------------

   --inp / --input

     Generates the input file and the templates for Gaussian.


   --prec [M m]

     Uses the preconditioned algorithm for the conformer location.

     In order to divide the preconditioned guesses into M groups
     and deal with the m-th group, use this option as follows:
          --hlopt M m
     For example:
          torsiflex --hlopt 10 2
     divides the guesses into 10 groups and only carries out
     the calculations associated to the 2nd group.

     --> Calculations with Gaussian are carried out <--


   --stoc

     Uses the stochastic algorithm for the conformer location.

     --> Calculations with Gaussian are carried out <--


   --hlopt [nocalc]

     Re-optimizes LL conformers at HL.

     When followed by 'nocalc':
          --hlopt nocalc
     it generates the gjf files (Gaussian inputs) without
     carrying out calculations.
     Useful to send the calculations on your own.

     --> Calculations with Gaussian are carried out <--


   --msho [ll/hl]

     Checks the located conformers and calculates the
     multi-structure harmonic-oscillator (MS-HO) partition
     functions.

     This option can be carried out exclusively for the
     low-level (ll) or the high-level (hl) conformers if
     followed by the corresponding abbreviation:
          --msho ll
          --msho hl


   --mstor [ll/hl]

     Generates the MsTor input files.

     This option can be carried out exclusively for the
     low-level (ll) or the high-level (hl) conformers if
     followed by the corresponding abbreviation:
          --mstor ll
          --mstor hl


   --regen

     Regenerates the domains.txt file using the temporal files.


   --help (also -h)

     Prints this help message.


   --version (also -v)

     Prints the program version.

---------------------
  Extra Information
```

```
---------------------

(a) Assert the path to the Gaussian executable
    is defined in your .bashrc file under the name
    'GauExe' and export it. Example:

    export GauExe='/home/programs/Gaussian/g09'

(b) Modify the Gaussian templates (in GauTemplates/),
    taking into account that:

    * [nproc], [mem], [level], [optmode], [charge],
      [multipl], [zmat], [modred], and [fccards]
      are TorsiFlex indications.
      They should not be removed.

    * Gaussian command line must start with '#p'

    * The command line includes 'iop(99/9=1,99/14=3)'.
      This is mandatory and should not be deleted:
         99/9=1  --> rotates to Z-matrix orientation first
         99/14=3 --> expresses final optimized structure
                     in terms of the input Z-matrix

    * 'scf=(incore)' is recommended with Hartree-Fock
      calculations. With it, Gaussian stores the full
      integral list in memory, speeding up the calculation.

(c) We highly recommend to firstly carry out the
    preconditioned search, which should found the
    chemically-intuitive conformers.
    After it, the stochastic algorithm should be used.
```

Notice that `TorsiFlex` accepts several options. Of them, we highlight the following:

> `--inp`: generates the `TorsiFlex` standard input file;
>
> `--prec`: performs the LL-searching using the preconditioned algorithm;
>
> `--stoc`: performs the LL-searching using the stochastic algorithm;
>
> `--hltop`: carries out the HL-refinement of the LL-conformers;
>
> `--msho`: checks the list of conformers and calculates the corresponding multi-structural harmonic-oscillator (MS-HO) partition function.
>
> `--mstor`: generates the `MsTor` input files.[6,7]

3.1 Input generation and Gaussian templates

`TorsiFlex` generates a default input file by executing:

```
  >> path_to_torsiflex/torsiflex.py --inp
```

This file, named `torsiflex.inp` (see Chapter 8), contains keywords that control the program. Their detailed description can be found in Chapter 5.

This execution also generates the following eight templates for the calculations with Gaussian inside the `GauTemplates/` folder:

- min_optLL: LL optimization of a minimum.
- min_optHL: HL optimization of a minimum.
- min_frqLL: LL frequency calculation of a minimum.
- min_frqHL: HL frequency calculation of a minimum.
- ts_optLL : LL optimization of a transition state (TS).
- ts_optHL : HL optimization of a TS.
- ts_frqLL : LL frequency calculation of a TS.
- ts_frqHL : HL frequency calculation of a TS.

Although the above files are templates for the calculations with Gaussian, they contain indications for TorsiFlex (between square brackets). The [zmat] indicator is automatically replaced with a guess Z-matrix and must not be removed under any circumstance. The rest of them, listed below, can be controlled with the keywords in the TorsiFlex input file (Chapter 8):

- [nproc]
- [mem]
- [level]
- [optmode]
- [charge]
- [multipl]
- [modred]
- [fccards]

3.2 Low-level search

TorsiFlex will carry out the preconditioned conformer localization when executed together with --prec:

```
 >> path_to_torsiflex/torsiflex.py --prec
```

whereas the stochastic search is carried out when executed with --stoc:

```
 >> path_to_torsiflex/torsiflex.py --stoc
```

The difference between the two procedures lies on the generation of the geometry guesses (see section 2.1). Optimum performance is achieved when the preconditioned generation precedes the stochastic one. There are two reasons for this choice. Firstly, the geometry optimizations of preconditioned guesses rarely fail. Secondly, two preconditioned guesses infrequently lead to the same conformer.

In order to speed up the preconditioned search, TorsiFlex can split the set of preconditioned guesses into G groups and deal with each individual group separately. Thus, TorsiFlex manages the g-th group with (g ≤ G):

```
 >> python3 torsiflex.py --prec G g
```

For example, to split the guess vectors into G=10 groups and take care of the second group (g=2) we use:

```
 >> python3 torsiflex.py --prec 10 2
```

3.3 High-level re-optimization

When the LL conformers are obtained, HL refinements can be carried out by using the `--hlopt` option:

```
  >> path_to_torsiflex/torsiflex.py --hlopt
```

This execution may computationally expensive. For this reason we recommend to run it in the background using either a queue system or the well-known `nohup` command.

3.4 Conformer analysis

Once all conformers (LL and/or HL) are obtained, the multi-structural harmonic-oscillator (MS-HO) partition function[8,9] can be calculated by:

```
  >> path_to_torsiflex/torsiflex.py --msho
```

This option also classifies the located conformers by increasing energy and lists their total and Gibbs free energies. Other properties, as dipole moments and rotational constants are also listed.

3.5 Torsional anhamonicity

A more accurate method than MS-HO to evaluate the partition function can be obtained by means of the coupled torsional anharmonic approximation, MS-T(C), which incorporates couplings in the kinetic and potential energies between the torsions.[10–12] This partition function can be calculated using the `MsTor` software.[6,7]

`TorsiFlex` is able to generate the standard `MsTor` input files when executed with the `--mstor` option:

```
  >> path_to_torsiflex/torsiflex.py --mstor
```

Firstly, `TorsiFlex` calculates the local periodicity parameter for each conformer (M_j for the j-th conformer) adopting a Monte Carlo approach:

$$M_j = \left(\frac{N_{\text{tot}}}{N_j}\right)^{1/K} \tag{3.1}$$

In this equation, N_{tot} is the total number of random points in the torsional space and N_j is the number of samples assigned to structure j. The number of torsions is represented by K.

The standard error of each M_j is given by:

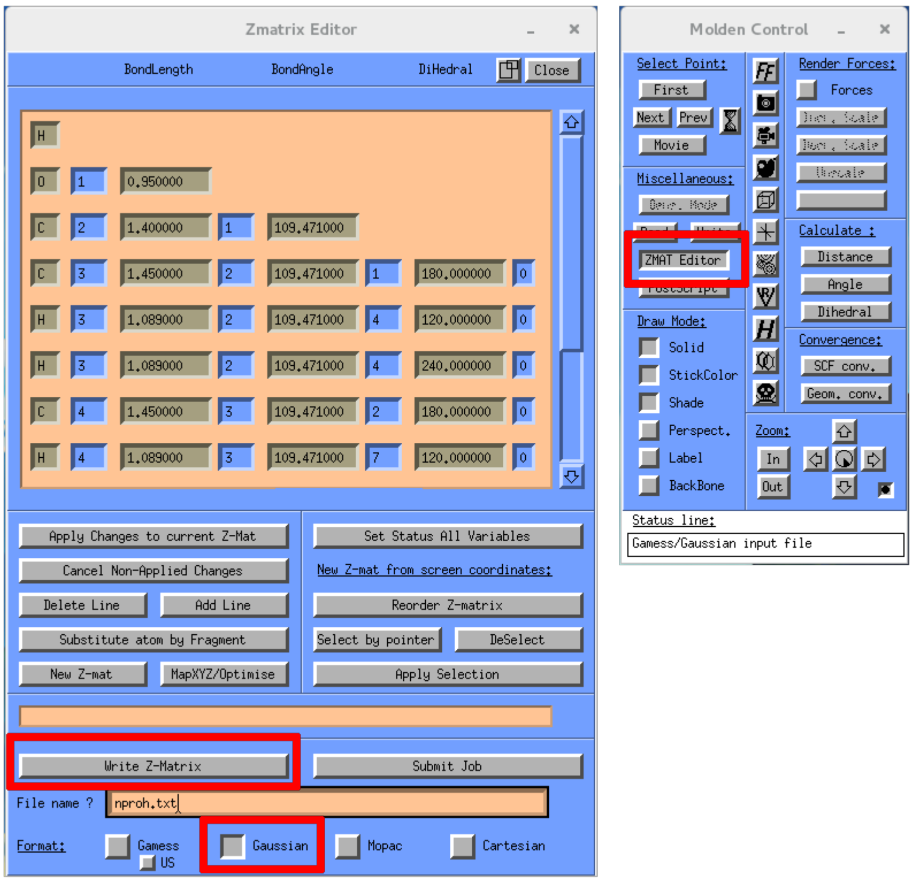
$$\sigma_{M_j} = \frac{M_j}{K\sqrt{N_j}} \tag{3.2}$$

The program increases N_{tot} until all σ_{M_j} are smaller (or equal) than the value associated with the `sigmamj` keyword.

Once these M_j parameters are calculated, `TorsiFlex` generates the two `MsTor` input files: `mstor.dat` and `hess.dat`.

4. The reference Z-matrix

The user has to provide `TorsiFlex` with a file containing the Z-matrix of the system. We highly recommend `Molden`[13,14] to build up the molecule. `Molden` saves the Z-matrix in the correct format by selecting the “Gaussian” option and by clicking on the “Write Z-Matrix” button (see Figure 4.1).



		BondLength		BondAngle		DiHedral	
H							
O	1	0.950000					
C	2	1.400000	1	109.471000			
C	3	1.450000	2	109.471000	1	180.000000	0
H	3	1.089000	2	109.471000	4	120.000000	0
H	3	1.089000	2	109.471000	4	240.000000	0
C	4	1.450000	3	109.471000	2	180.000000	0
H	4	1.089000	3	109.471000	7	120.000000	0

File name ? nproh.txt

Gamess/Gaussian input file

Figure 4.1: The Molden interface.

We highlight that the torsions of interest should be defined univocally: the file must contain **only one proper torsion about each torsional bond**; otherwise the `TorsiFlex` algorithm will fail.

5. The `TorsiFlex` input file

5.1 Keywords

An example of the `TorsiFlex` input file, `torsiflex.inp`, can be found in Chapter 8. This file contains different keywords, which are listed in Table 5.1. A detailed description of their meaning can be found in the following sections.

5.1.1 System-specific keywords

- `zmatfile`: the file containing the Z-matrix of the system. This file can be created with `Molden`.[13,14] See Chapter 4.
- `enantio`: `yes` if torsional enantiomers are generated upon internal rotations, `no` otherwise.
- `ts`: `yes` if a transition state (TS) structure is being studied, `no` otherwise.
- `cfactor`: a factor that control the distance criterium needed to obtain the connectivity graph of the system. For a pair of atoms, the higher the value of `cfactor`, the easiest is for the two atoms to be 'connected'. We recommend a value of 1.3 for a proper connectivity.
- `skipcon`: accepts pair(s) of atoms, (a,b), a and b being the numerical label of each atom. The connectivity between the selected pair(s) of atoms will be omitted from the connectivity test. Several pairs of atoms can be defined in each line, for instance:

  ```
  skipcon (1,2) (4,5)
  ```

  Several `skipcon` lines can be used.

5.1.2 Keywords associated with the target torsions

In the keywords, `X` must be replaced by the corresponding integer.

- `torsionX`: the name of the target torsion in the Z-matrix file (`zmatfile` variable).

- `precondX`: defines the preconditioned angles of `torsionX`. By default, these values are 60, 180 and 300 degrees.
- `tdomainX`: defines the allowed domain of `torsionX`. See section 5.3 for more information on the definition of domains. By default the domain is [0,360).
- `tsigmaX`: the torsional symmetry number for `torsionX`. By default, it is set to 1.
- `pcfile`: a file containing preconditioned angles. It overrides `precondX`. See section 5.2.

5.1.3 Keywords associated with the search procedure

- `ncycles`: the number of steps in the random search of conformers (active when `TorsiFlex` is executed with the `--stoc` option).

5.1.4 Keywords associated with the HL re-optimization

- `hlcutoff`: a cutoff for HL calculations. LL conformers with relative Gibbs free energy smaller than `hlcutoff` (in kcal/mol) are re-optimized at HL. If this keyword is deactivated, all LL conformers are re-optimized.
- `tempGibbs`: temperature, in Kelvin, for the calculation of the Gibbs free energy.

5.1.5 Keywords associated with the validation tests

- `testsG`: controls which tests are carried out on a **Guess geometry**. This keyword must be followed by four integers, each one controlling one test. These tests are the connectivity, similarity, hard-constraint and soft-constraint tests (given in this order). Use 1 to activate a test and 0 to switch it off. By default, its value is:

  ```
  testsG 1 1 1 1
  ```

  For example, the connectivity test for the guess geometry is deactivated with the following line:

  ```
  testsG 0 1 1 1
  ```

- `testsO`: controls which tests are carried out on a **Optimized geometry**. This keyword must be followed by four integers, each one controlling one test. These tests are the connectivity, redundancy, hard-constraint and soft-constraint tests (given in this order). Use 1 to activate a test and 0 to switch it off. By default, its value is:

  ```
  testsO 1 1 1 1
  ```

- dist1D: the domain about a torsional angle (in degrees). An angle ϕ_i belongs to the domain of another angle ϕ_j if their difference is within dist1D (d):

  $$\phi_i \in \mathrm{Dom}(\phi_j) \text{ if } |\phi_i - \phi_j| < d$$

  This variable defines the threshold in the **similarity test**.

- epsdeg: a criterium to decide whether two angles are equal or not (in degrees). Two angles, ϕ_i and ϕ_j, are considered identical if their difference is within epsdeg (ϵ):

  $$\phi_i = \phi_j \text{ if } |\phi_i - \phi_j| < \epsilon$$

  This variable defines the threshold in the **redundancy test**.

- hconstr: defines a *hard* constraint (one per line). The argument of a hconstr line consists of an internal coordinate (a distance, an angle or a torsional angle) and a domain:

  ```
  hconstr icoord domain
  ```

  The internal coordinate can be either defined in the Z-matrix or can be defined by listing the involved atoms. As an example, for the latter, 1-2-3 refers to the angle defined by the atoms labelled as 1, 2 and 3. The definition of domains can be found below, in section 5.3. We notice that all hard-constraints are mandatory and a guess/optimized geometry is **discarded** if **a single** hard constraint is **not fulfilled**.

- sconstr: defines a *soft* constraint (one per line) using the same format as in hconstr:

  ```
  sconstr icoord domain
  ```

  In this case, a guess/optimized geometry is **discarded** if **all** the soft constraints are **not fulfilled**.

- ifqrangeLL: restricts the imaginary frequency to the defined interval (for LL conformers of saddle point structures). When the imaginary frequency falls outside this interval, the structure is discarded.

- ifqrangeHL: same as ifqrangeLL but for HL conformers.

5.1.6 Keywords associated with the calculations with Gaussian

- optmode: defines the arguments of the opt keyword in the Gaussian input file (it replaces the [optmode] string in the template). It can be:
  - 0: to use opt(z-matrix)
  - 1: to use opt(modreduntant).

- fccards: when set to yes, the LL Hessian matrix is used in the HL optimization. This keyword will also modify the [optmode] string in the template by adding the fccards argument:

  - `opt(z-matrix,fccards)`
  - `opt(modreduntant,fccards)`.

- `charge`: the charge of the system (it replaces the `[charge]` string in the Gaussian templates).

- `multipl`: the spin multiplicity of the system (it replaces the `[multipl]` string in the Gaussian templates).

- `lowlevel`: the low-level to be inserted in the Gaussian template. This keyword replaces the `[level]` string in the Gaussian template.

- `highlevel`: same as `lowlevel` but to define the level of calculation in the HL templates.

- `nproc`: number of threads for Gaussian calculations. This keyword replaces the `[nproc]` string in the Gaussian template.

- `procll`: same as `proc` but it only applies to LL Gaussian templates.

- `prochl`: same as `proc` but it only applies to HL Gaussian templates.

- `mem`: dynamic memory for Gaussian calculations. This keyword replaces the `[mem]` string in the Gaussian template.

- `memll`: same as `mem` but it only applies to LL Gaussian templates.

- `memhl`: same as `mem` but it only applies to HL Gaussian templates.

5.1.7 Keywords associated with the partition functions

- `tempsPF`: temperatures, in Kelvin, for the calculation of partition functions. Each line can contain several temperatures. More than one line can be used.

- `freqscalLL`: a scaling factor for the harmonic frequencies of the LL conformers, if needed. By default, frequencies are not scaled (i.e. `freqscalLL` is set to 1).

- `freqscalHL`: a scaling factor for the harmonic frequencies of the HL conformers, if needed. By default, frequencies are not scaled (i.e. `freqscalHL` is set to 1).

- `sigmamj`: the standard error in the calculation of the local periodicity parameters for MsTor (see section 3.5). This value must be $\geq 10^{-2}$.

5.1.8 Keywords associated with the storage of conformers

- `dirll`: the folder where the LL conformers are stored.
- `dirhl`: the folder where the HL conformers are stored.
- `tmpll`: the folder where the temporal LL Gaussian files are stored. By default this folder is set to `/scratch/user/LL_xxxxx/` where `user` is the name of the user and `xxxxx` is a random sequence of characters. The folder `LL_xxxxx/` is automatically created by the program when executed. If the creation fails, the program will let the user know that the folder must be created by themselves.
- `tmphl`: the folder where the temporal HL Gaussian files are stored. See also `tmpll`.

Table 5.1: Available keywords in the TorsiFlex input file.

Keyword	(Type of) value	More info
`zmatfile`	string	path to Z-matrix file
`enantio`	`yes`, `no`	are there torsional enantiomers?
`ts`	`yes`, `no`	is the system a transition state?
`cfactor`	float	controls system connectivity
`skipcon`	pair(s) of atoms (a,b)	excludes pairs of atoms in connectivity test
`torsionX`	string	the coordinate name of the `X`-th torsion in the Z-matrix file; replace `X` with integer;
`precondX`	a list of integers	preconditioned angles for `torsionX`; replace `X` with integer; e.g. `precond1 0 180`
`tdomainX`	a domain	domain for `torsionX`; replace `X` with integer; e.g. `tdomain1 (-60,60)`
`tsigmaX`	integer	torsional symmetry number of `torsionX`; replace `X` with integer
`pcfile`	string	path to file with preconditioned angles
`ncycles`	integer	number of cycles in stochastic search; for `--stoc`
`hlcutoff`	float (kcal/mol)	Gibbs free energy cutoff for HL re-optimizations
`tempGibbs`	float (Kelvin)	temperature for the calculation of Gibbs free energy
`testsG`	four integers (0's or 1's)	selected validation tests for Guess geometries
`testsO`	four integers (0's or 1's)	selected validation tests for Optimized geometries
`dist1D`	integer (in degrees)	≥ 2.0; domain about an angle
`epsdeg`	integer (in degrees)	≤ 2.0; max. difference between equal angles
`hconstr`	internal coord. and domain	one line per hard constraint
`sconstr`	internal coord. and domain	one line per soft constraint
`ifqrangeLL`	domain	restricts values for the imaginary frequency (LL)
`ifqrangeHL`	domain	restricts values for the imaginary frequency (HL)
`optmode`	`0,1`	defines the argument of the `opt` Gaussian keyword
`fccards`	`yes,no`	use LL Hessian in HL optimization
`charge`	integer (in a.u.)	charge of the system
`multipl`	integer	spin multiplicity of the system
`lowlevel`	string	low-level methodology
`highlevel`	string	high-level methodology
`nproc`	integer	number of threads in Gaussian template (e.g. 2)
`nprocll`	integer	same as `nproc` but only for LL
`nprochl`	integer	same as `nproc` but only for HL
`mem`	string	dynamic memory in Gaussian template (e.g. `2GB`)
`memll`	string	same as `mem` but only for LL
`memhl`	string	same as `mem` but only for HL
`tempsPF`	list of floats (Kelvin)	temperatures for the partition functions
`freqscalLL`	float	scale factor for harmonic frequencies (LL conformers)
`freqscalHL`	float	scale factor for harmonic frequencies (HL conformers)
`sigmamj`	float	≥ 0.01; for `--mstor`
`dirll`	string	folder for storing LL conformers
`dirhl`	string	folder for storing HL conformers
`tmpll`	string	folder for LL Gaussian calculations
`tmphl`	string	folder for HL Gaussian calculations

5.2 The file with preconditioned angles: `pcfile`

The preconditioned angles can be also introduced as plain text using the `pcfile` keyword. Different combinations are possible for the selected torsions.

The first line in this file consists of the target torsions, whereas the rest of the lines define their corresponding values (separated by blank spaces or by an underscore). For example, the next file:

```
 1   3
100_100
200_300
```

defines two combinations:

- (`torsion1`,`torsion3`) = (100, 100)
- (`torsion1`,`torsion3`) = (200, 300)

The value of the disregarded torsions, if any, will be defined according to the default angles, *i.e.* the (60,180,300) triad or those defined through `precondX`. In this manner, if the system contains three torsions (`torsion1`,`torsion2`,`torsion3`), the previous lines would lead to the next points:

- (100,60,100)
- (100,180,100)
- (100,300,100)
- (200,60,300)
- (200,180,300)
- (200,300,300)

5.3 Specifying a domain

Keywords `hconstr`, `sconstr` and `tdomainX` require the definition of a domain. An interval can be defined with the format `(a,b)`, where `a` and `b` are the limits of the interval (`a<b`). Several intervals can be used to define a given domain through their union (denoted by a capital `U`). For example, `(a,b)U(c,d)`, where `a<b<c<d`.

In the case of `tdomainX`, the domain corresponds to a torsion angle (in degrees). However, `hconstr` or `sconstr` can be specified to restrict the domain of a distance (in Angstrom) or of an angle (in degrees).

6. Some tips

6.1 Low-level calculations at the HF level

If Hartree-Fock (HF) is the LL method of choice for the searching algorithm, we highly recommend to modify the Gaussian template and include the argument `incore` in the `scf` Gaussian keyword:

```
scf=(verytight,incore)
```

In this manner, the full integral list will be stored in memory along the self-consistent field method (SCF). This should speed up the calculation. Obviously, if `incore` is used, enough memory should be available.

6.2 Systems with many LL conformers

The HL re-optimizations are carried out sequentially, which may not be convenient for systems with a big number of conformers. In such a cases, we recommend to execute `TorsiFlex` as follows:

```
  >> path_to_torsiflex/torsiflex.py --hlopt nocalc
```

This will generate the Gaussian input files for the HL re-optimization of all the LL conformers. In this manner, the user can execute all Gaussian jobs at the same time if desired (i.e. the calculations are managed by the user and not by `TorsiFlex`). Once the optimizations are complete, the corresponding Gaussian input files for the frequency calculations can be also generated by executing `TorsiFlex` again:

```
  >> path_to_torsiflex/torsiflex.py --hlopt nocalc
```

When all optimization and frequency calculations are finished, `TorsiFlex` can be executed normally and will automatically read the generated output files:

```
  >> path_to_torsiflex/torsiflex.py --hlopt
```

6.3 Focusing on low energy conformers

The user may also be interested into re-optimize only the most stable conformers. In such cases, we recommend to use the `hlcutoff` keyword. Only those LL conformers whose Gibbs free energy is below the value associated to `hlcutoff` are optimized at HL.

For example, in order to optimize at HL those conformers whose relative Gibbs free energy is smaller than 3 kcal/mol at 300 K, we need the following line in the input file:

```
hlcutoff    3.0 300.0
```

The first argument of `hlcutoff` defines the cutoff for the Gibbs free energy, whereas the second argument defines the temperature for the calculation of such an energy. If not indicated, the default temperature is 300 K.

6.4 Dealing with similar molecules

When performing conformational studies, it may occur that we have to deal with two molecules with common fragments. For the sake of simplicity, let us name these two molecules as AM and BM, both containing the M moiety. In such cases, it is very likely for the conformations of the M moiety in A and in B to be similar. If we have performed a full study on AM, it would be very convenient to use the conformations of its M moiety as a guess for BM. This may be achieved by defining the pre-conditioned angles through `pcfile` instead of with the `precondX` keyword.

For instance, let us assume that molecule AM consists of three torsions, `torsion1` to `torsion3`, the last two torsions being located in the M fragment, whereas BM consists of four torsions, `torsion1` to `torsion4`, the last two corresponding to M. `TorsiFlex` can initialize `torsion3` and `torsion4` of BM according to the values of `torsion2` and `torsion3` of AM. The procedure is as follows:

(a) Go to the working directory of AM and enter in the folder where conformers are stored.

(b) List the angles using:

```
  >> ls *log | awk -F. '{print $2}'
```

The output would look like:

```
069_284_314
071_172_060
071_182_301
072_268_179
073_080_304
075_093_192
075_280_051
```

(c) As we are only interested in the second and third columns of this list (`torsion2` and `torsion3`), the first column should be eliminated. This can be achieved with a slight modification of the command:

```
>> ls *log | awk -F. '{print $2}' | \
   awk -F"_" '{print $2, $3}'
```

leading to:

```
284 314
172 060
182 301
268 179
080 304
093 192
280 051
```

(d) Save these lines in a new file and use it for BM through the `pcfile` keyword. A first line, indicating the target torsions, must be added to this file:

```
 3   4
284 314
172 060
182 301
268 179
080 304
093 192
280 051
```

(e) Now, the initial angles for torsions `torsion3` and `torsion4` of BM are set to those in AM, whereas starting values for `torsion1` and `torsion2` are defined by the default triad, (60,180,300).

An example of this situation may be found when studying the amino acids. For example, alanine and valine, or cysteine and methionine, et cetera.

Finally, we notice that (i) `precondX` defines preconditioned values for individual torsions whereas (ii) `pcfile` defines preconditioned (sub)vectors in the torsional space.

7. Worked examples

This Chapter contains the following step-by-steps examples:

- WE1: n-butanol
- WE2: L-alanine
- WE3: L-proline
- WE4: Transition state for the H abstraction from n-butanol at the C—α atom by the H radical

The level of calculation for the search procedure is set to HF/3-21G (low-level, LL), whereas B3LYP/6-31G is employed as the accurate high-level (HL) for the re-optimizations.

Torsions of interest

Torsions for WE1:

- `dih4` (H1-O2-C3-C4)
- `dih5` (O2-C3-C4-C5)
- `dih6` (C3-C4-C5-H6)

Torsions for WE2:

- `dih4` (H1-O2-C3-C4)
- `dih5` (O2-C3-C4-C5)
- `dih12` (C3-C4-N8-H12)

Torsions for WE3:

- `dih4` (C1-C2-C3-C4)
- `dih5` (C2-C3-C4-N5)
- `dih17` (C3-C4-N5-H17)
- `dih7` (C2-C1-C6-O7)
- `dih8` (C1-C6-O7-H8)

Torsions for WE4:

- `dih4` (H1-O2-C3-C4)
- `dih5` (O2-C3-C4-C5)
- `dih6` (C3-C4-C5-C6)

7.1 WE1: conformers of n-butanol

In this worked example, we find the conformers of n-butanol (Figure 7.1)

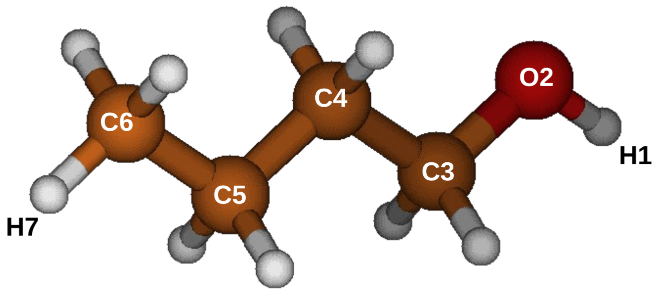


Figure 7.1: Labeling for n-butanol.

How to proceed

- Assert that the GauExe variable with the path to your Gaussian version is exported in the .bashrc file.
- Create the folder WE1/ and enter to it.

```
>> mkdir WE1/
>> cd WE1/
```

- Copy the corresponding Z-matrix file (we01.zmat, see Chapter 8) from the tests/ folder to the current directory (or create it with the help of Molden).
- Create the input file and the templates for the Gaussian calculations:

```
>> path_to_torsiflex/torsiflex.py --inp
```

- Modify/add the following keywords in the input file:

```
zmatfile    we01.zmat
enantio     yes
ncycles     20
lowlevel    hf/3-21g
```

```
torsion1    dih4
torsion2    dih5
torsion3    dih6
```

  Keyword tmpll may be also modified.

- In principle, it is not necessary to modify the templates for Gaussian. However, the incore argument could be included in the scf Gaussian option to speed-up the LL calculations (min_optLL file inside GauTemplates/).
- Execute TorsiFlex with the --prec option:

```
>> path_to_torsiflex/torsiflex.py --prec
```

In a few minutes, the program finds the conformers from the preconditioned guesses at the HF/3-21g level. A total of 14 conformers should be found.

- Execute `TorsiFlex` with the `--stoc` option:

```
  >> path_to_torsiflex/torsiflex.py --stoc
```

In this case, `TorsiFlex` considers random angles to find new conformers. Execute `TorsiFlex` as many times as needed. A new conformer should be found in this random search (086_305_086 or its torsional enantiomer, 274_055_274).

- A total of 15 LL conformers should be found (14+1). Notice that there exists an enantiomer for any of the conformers, but the one with Cs symmetry. Consequently, there are 29 conformers for n-butanol $(14 \cdot 2 + 1)$.
- List your LL conformers and calculate the corresponding MS-HO partition functions using `--msho`:

```
  >> path_to_torsiflex/torsiflex.py --msho ll
```

- Modify again the input file to indicate the high-level of calculation:

```
highlevel  b3lyp/6-31g
```

Keyword `tmphl` may be also modified.

- Execute `TorsiFlex` to carry out the HL optimizations based on the LL structures:

```
  >> path_to_torsiflex/torsiflex.py --hlopt
```

This execution may take a few minutes.

- List your HL conformers and calculate the corresponding MS-HO partition function using the `--msho` option:

```
  >> path_to_torsiflex/torsiflex.py --msho hl
```

List of low-level conformers

1. (295,304,176)
2. (176,300,177)
3. (295,306,290)
4. (297,296,073)
5. (296,183,180)
6. (177,302,291)
7. (285,062,179)
8. (180,180,180)
9. (178,293,078)
10. (297,181,069)
11. (295,186,291)
12. (283,058,065)
13. (181,183,292)
14. (274,055,274)
15. (292,084,299)

List of high-level conformers

1. (297,301,177)
2. (298,183,180)
3. (291,065,179)
4. (176,297,179)
5. (180,180,180)
6. (298,303,293)
7. (177,299,294)
8. (300,181,066)
9. (300,296,073)
10. (292,064,066)
11. (297,186,293)
12. (182,183,294)
13. (179,290,075)
14. (293,081,298)
15. (283,063,274)

7.2 WE2: conformers of L-alanine

In this example, we find the conformers of L-alanine (see Figure 7.2).

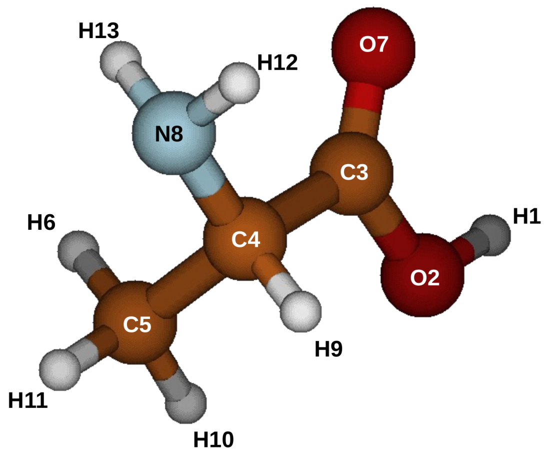


Figure 7.2: Labeling for Alanine.

How to proceed

- Assert that the GauExe variable with the path to your Gaussian version is exported in the .bashrc file.
- Create the folder WE2/ and enter to it.

```
>> mkdir WE2/
>> cd WE2/
```

- Copy the corresponding Z-matrix file (we02.zmat, see Chapter 8) from the tests/ folder to the current directory (or create it with the help of Molden).
- Create the input file and the templates for the Gaussian calculations:

```
>> path_to_torsiflex/torsiflex.py --inp
```

- Modify/add the following keywords in the input file:

```
zmatfile    we02.zmat
ncycles     20
lowlevel    hf/3-21g
precond1    0 180
```

```
torsion1    dih4
torsion2    dih5
torsion3    dih12
```

Notice that keyword precond1 is used to set the preconditioned angles for the carboxyl group to 0 and 180 degrees. Keywords tmpll and tmphl may be also modified.

- In principle, it is not necessary to modify the templates for Gaussian. However, the `incore` argument could be included in the `scf` Gaussian option to speed-up the LL calculations (`min_optLL` file inside `GauTemplates/`).
- Execute `TorsiFlex` with the `--prec` option:

  ```
   >> path_to_torsiflex/torsiflex.py --prec
  ```

  In a few minutes, the program finds the conformers at the HF/3-21g level from the preconditioned guesses. A total of 11 conformers should be found.
- Execute `TorsiFlex` with the `--stoc` option:

  ```
   >> path_to_torsiflex/torsiflex.py --stoc
  ```

  In this case, `TorsiFlex` considers random angles to find new conformers. This execution may take some time, depending on the value of the `ncycles` keyword. No new conformers are found for this system.
- List your LL conformers and calculate the corresponding MS-HO partition function using `--msho`:

  ```
   >> path_to_torsiflex/torsiflex.py --msho ll
  ```

- Modify again the input file to perform the HL calculations:

  ```
  highlevel  b3lyp/6-31g
  ```

- Execute `TorsiFlex` to carry out the HL optimizations based on the LL structures:

  ```
   >> path_to_torsiflex/torsiflex.py --hltopt
  ```

  This execution may take a few minutes.
- List your HL conformers and calculate the corresponding MS-HO partition function using the `--msho` option:

  ```
   >> path_to_torsiflex/torsiflex.py --msho hl
  ```

  The number of conformers should have been reduced to 10.

List of low-level conformers

1. (179,298,294)
2. (354,147,088)
3. (178,162,304)
4. (182,075,288)
5. (178,291,033)
6. (181,307,197)
7. (182,171,175)
8. (179,069,059)
9. (355,294,290)
10. (351,289,023)
11. (013,183,174)

List of high-level conformers

1. (356,139,097)
2. (179,294,295)
3. (184,197,295)
4. (181,077,290)
5. (178,289,027)
6. (181,309,196)
7. (180,166,175)
8. (180,076,058)
9. (358,293,294)
10. (355,291,016)

7.3 WE3: conformers of proline

In this example, we find the conformers of L-proline (see Figure 7.4).

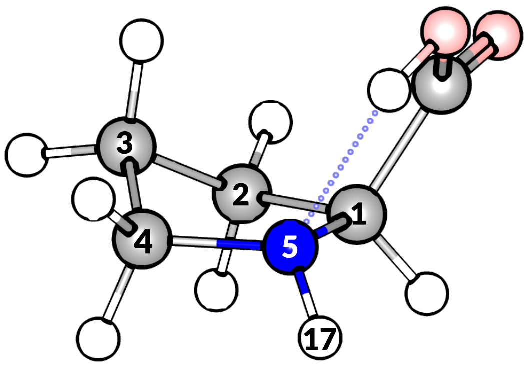


Figure 7.3: Cycle numeration in Pro. C in grey, O in red, N in blue and H in white.

We notice that L-proline is the only proteinogenic amino acid containing a flexible ring in its structure. `TorsiFlex` is not specially optimized for dealing with ring puckering, but it can still be used to find the corresponding conformers. If we consider the numbering shown in Figure 7.3, the ring configurations arise from the following torsions: (i) C_1-C_2-C_3-C_4, (ii) C_2-C_3-C_4-N_5 and (iii) C_3-C_4-N_5-H_{17}. In order to avoid huge distortions in the five-member ring, the domains of these torsions are limited to a small interval. Moreover, since the connectivity of the ring may be compromised in the guess structure, the connectivity test for guess structures should be deactivated.

How to proceed

- Assert that the `GauExe` variable with the path to your Gaussian version is exported in the `.bashrc` file.
- Create the folder `WE2/` and enter to it.

```
>> mkdir WE3/
>> cd WE3/
```

- Copy the corresponding Z-matrix file (`we03.zmat`, see Chapter 8) from the `tests/` folder to the current directory (or create it with the help of `Molden`).
- Create the input file and the templates for the Gaussian calculations:

```
>> path_to_torsiflex/torsiflex.py --inp
```

- Modify/add the following keywords in the input file:

```
zmatfile    we03.zmat
testsG      0 1 1 1
lowlevel    hf/3-21g
torsion1    dih4
torsion2    dih5
torsion3    dih17
torsion4    dih7
torsion5    dih8
```

```
tdomain1    (-50,50)
tdomain2    (-50,50)
tdomain3    (-160,-80)U(80,160)
precond1     -30  +30
precond2     -30  +30
precond3    -120 +120
precond5       0  180
```

Notice that we have reduced the interval for dih4 and dih5 to (-50,50), in order to generate structures where the five-member ring is “conserved”. Similarly, the interval for dih17 should account for the two arrangements in the umbrella inversion (in this case, this conformational effect leads to different conformers). The preconditioned angles for these torsions were also modified, in order to fit the previous defined intervals. As in WE2, `precond5` account for the two expected conformations of the carboxylic acid group. The first argument of `testsG` is set to `0` to deactivate the connectivity test on the guess structures.

- Execute `TorsiFlex` with the `--prec` option:

```
>> path_to_torsiflex/torsiflex.py --prec
```

In a few minutes, the program finds the conformers at the HF/3-21g level from the preconditioned guesses. A total of 18 conformers should be found.

- Execute `TorsiFlex` with the `--stoc` option:

```
>> path_to_torsiflex/torsiflex.py --stoc
```

In this case, `TorsiFlex` considers random angles to find new conformers. This execution may take some time, depending on the value of the `ncycles` keyword. A total of 2 new conformers should be found.

- List your LL conformers and calculate the corresponding MS-HO partition function using `--msho`:

```
>> path_to_torsiflex/torsiflex.py --msho ll
```

- Modify again the input file to perform the HL calculations:

```
highlevel   b3lyp/6-31g
```

- Execute `TorsiFlex` to carry out the HL optimizations based on the LL structures:

```
>> path_to_torsiflex/torsiflex.py --hltopt
```

This execution may take a few minutes.

- List your HL conformers and calculate the corresponding MS-HO partition function using the `--msho` option:

```
>> path_to_torsiflex/torsiflex.py --msho hl
```

The number of conformers should have been reduced to 14.

List of low-level conformers

1. (320,037,211,121,358)
2. (321,029,123,294,180)
3. (016,325,171,298,181)
4. (012,330,274,139,354)
5. (323,022,136,159,179)
6. (013,327,171,157,179)
7. (322,039,195,186,186)
8. (040,330,239,098,007)
9. (323,038,198,043,176)
10. (037,335,225,181,184)
11. (031,319,171,074,182)
12. (025,322,272,204,186)
13. (027,322,271,052,175)
14. (037,337,206,299,180)
15. (007,017,081,158,355)
16. (320,031,117,291,354)
17. (022,321,170,298,357)
18. (029,319,169,182,012)
19. (331,010,246,283,352)
20. (356,342,172,038,352)

List of high-level conformers

1. (323,036,209,117,359)
2. (027,324,266,127,357)
3. (324,028,123,294,181)
4. (033,322,163,296,181)
5. (325,023,136,145,177)
6. (017,326,173,151,178)
7. (326,036,190,188,185)
8. (035,330,234,195,185)
9. (326,034,195,040,176)
10. (031,325,255,049,175)
11. (007,017,082,156,356)
12. (033,322,162,297,001)
13. (324,030,118,292,359)
14. (029,321,171,196,009)

7.4 WE4: conformers of a transition state

In this example, we find the conformers of the transition state associated to the H abstraction by H in the α position of n-BuOH (see Figure 7.4).

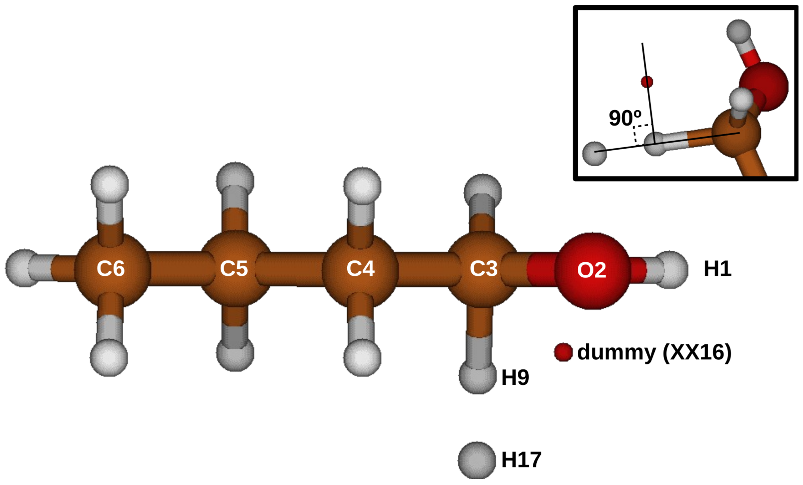


Figure 7.4: Labeling for the H-abstraction transition state. The definition of the dummy atom is also shown from another point of view.

How to proceed

- Assert that the GauExe variable with the path to your Gaussian version is exported in the .bashrc file.
- Create the folder WE4/ and enter to it.

```
>> mkdir WE4/
>> cd WE4/
```

- Copy the corresponding Z-matrix file (we04.zmat, see Chapter 8) from the tests/ folder to the current directory (or create it with the help of Molden).
- Create the input file and the templates for the Gaussian calculations:

```
>> path_to_torsiflex/torsiflex.py --inp
```

- Modify/add the following keywords in the input file:

```
zmatfile      we04.zmat
ts            yes
multipl       2
lowlevel      hf/3-21g
skipcon       (3,9) (9,17)
```

```
ifqrangeLL   (1500,2500)
torsion1     dih4
torsion2     dih5
torsion3     dih6
```

Notice that setting ts to yes indicates that we are dealing with a transition state (different Gaussian templates are selected), whereas multipl defines the spin multiplicity of the system. Two extra keywords (skipcon and ifqrangeLL) are included to consider two specific problems that may arise.

On the one side, it may occur that the connectivity between H9 and H17 differs from one conformer to another due to a small variation in the distance between them. The skipcon keyword can be used to omit the two distances in the C···H···H moiety when carrying out the connectivity test.[*]

On the other side, the system could end up in an internal rotation transition state.[†] The ifqrangeLL keyword can be used to define an expected range of values for the imaginary frequency, which should be betweem 1500i and 2500i cm^{-1} for a H-abstraction transition-state at the HF/3-21G level.

We notice that, for this system, these two last keywords are not really required, but we recommend to include them to avoid the unique situations previously described.

- Execute TorsiFlex with the --prec option:

```
 >> path_to_torsiflex/torsiflex.py --prec
```

  In a few minutes, the program finds the conformers at the HF/3-21g level from the preconditioned guesses. A total of 22 conformers should be found.

- Execute TorsiFlex with the --stoc option:

```
 >> path_to_torsiflex/torsiflex.py --stoc
```

  In this case, TorsiFlex considers random angles to find new conformers. This execution may take some time, depending on the value of the ncycles keyword. A total of 3 new conformers should be found.

- List your LL conformers and calculate the corresponding MS-HO partition function using --msho:

```
 >> path_to_torsiflex/torsiflex.py --msho ll
```

  This also lists the imaginary frequency of each transition state conformer. A total of 25 LL conformers should be listed.

- Modify again the input file to perform the HL calculations:

```
highlevel  b3lyp/6-31g
```

- Execute TorsiFlex to carry out the HL optimizations based on the LL structures:

```
 >> path_to_torsiflex/torsiflex.py --hltopt
```

  This execution may take a few minutes.

[*]This could also be solved by modifying the cfactor keyword.

[†]For example, if the H radical goes away from butanol during the optimization.

- List your HL conformers and calculate the corresponding MS-HO partition function using the `--msho` option:

  ```
  >> path_to_torsiflex/torsiflex.py --msho hl
  ```

  The number of conformers should be reduced to 19.

List of low-level conformers

1. (056,058,184)
2. (176,306,177)
3. (182,062,183)
4. (055,068,290)
5. (056,055,069)
6. (054,176,180)
7. (181,059,068)
8. (178,179,180)
9. (180,071,285)
10. (061,304,182)
11. (177,304,081)
12. (179,321,299)
13. (053,177,290)
14. (179,181,290)
15. (063,315,296)
16. (176,168,067)
17. (056,166,067)
18. (072,314,084)
19. (286,311,176)
20. (051,278,059)
21. (290,184,180)
22. (287,304,074)
23. (285,326,300)
24. (287,185,289)
25. (293,174,067)

List of high-level conformers

1. (051,063,183)
2. (187,066,181)
3. (050,177,180)
4. (183,302,179)
5. (186,180,181)
6. (052,300,182)
7. (049,060,066)
8. (049,069,290)
9. (185,064,066)
10. (049,180,293)
11. (185,074,287)
12. (184,183,293)
13. (182,311,295)
14. (047,308,296)
15. (185,298,077)
16. (051,170,070)
17. (181,171,069)
18. (048,283,061)
19. (056,306,088)

8. Example files

Example of file with preconditioned angles (`pcfile` keyword).

```
 1        3        4
068      174      063
068      183      300
071      298      299
075      291      175
172      306      296
175      300      174
179      298      070
180      176      062
180      180      180
286      307      297
292      182      182
293      301      073
299      082      295
```

Gaussian template for LL optimization of a minimum.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(tight)
iop(99/9=1,99/14=3)
opt=([optmode],tight,MaxCycles=200)

--Optimization of minimum--

[charge],[multipl]
[zmat]
[modred]
[fccards]
```

Gaussian template for HL optimization of a minimum.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(verytight)
iop(99/9=1,99/14=3)
opt=([optmode],verytight,MaxCycles=200)

--Optimization of minimum--

[charge],[multipl]
[zmat]
[modred]
[fccards]
```

Gaussian template for LL optimization of a transition state.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(tight)
iop(99/9=1,99/14=3)
opt=([optmode],tight,calcfc,ts,noeigentest,MaxCycles=200)

--Optimization of transition state--

[charge],[multipl]
[zmat]
[modred]
[fccards]
```

Gaussian template for HL optimization of a transition state.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(verytight)
iop(99/9=1,99/14=3)
opt=([optmode],verytight,calcfc,ts,noeigentest,MaxCycles=200)

--Optimization of transition state--

[charge],[multipl]
[zmat]
[modred]
[fccards]
```

Gaussian template for LL frequency calculation.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(tight)
iop(99/9=1,99/14=3)
freq=noraman

--Frequency calculation--

[charge],[multipl]
[zmat]
```

Gaussian template for HL frequency calculation.

```
%nproc=[nproc]
%mem=[mem]
#p [level]
scf=(verytight)
iop(99/9=1,99/14=3)
freq=noraman

--Frequency calculation--

[charge],[multipl]
[zmat]
```

The default TorsiFlex input file

```
# This is a torsiflex input file

#-----------------------#
#        System         #
#-----------------------#
zmatfile    zmat.txt    # Z-matrix file
enantio     no          # yes if torsional enantioners, no otherwise
ts          no          # yes if transition state, no otherwise
cfactor     1.3         # controls the connectivity criterium
#skipcon    (1,2)       # skips connectivity between pairs of atoms


#-----------------------#
#    Target torsions    #
#-----------------------#
torsion1    --          # name of 1st target torsion in the Z-matrix file
#precond1   60 180 300  # precond angles for torsion1
#tdomain1   (0,360)     # domain for torsion1
#tsigma1    1           # symmetry number for torsion1
#pcfile     precond.txt # file with precond. angles


#-----------------------#
#   Search  Procedure   #
#-----------------------#
ncycles     200         # number of steps of stochastic algorithm


#-----------------------#
#   HL reoptimization   #
#-----------------------#
#hlcutoff    5.0        # Gibbs energy cutoff (kcal/mol)
tempGibbs  298.15       # Temperature (K) for Gibbs free energy


#-----------------------#
#   Validation  tests   #
#-----------------------#
testsG      1 1 1 1     # for Guess geom (Conn, Simil, Hard, Soft)
testsO      1 1 1 1     # for Opt   geom (Conn, Redun, Hard, Soft)
dist1D      15          # domain size about each point (degrees)
epsdeg      2           # max diff between two identical angles (degrees)
#hconstr    ic domain   # hard constraint (see manual)
#sconstr    ic domain   # soft constraint (see manual)
#ifqrangeLL domain      # restricts LL imaginary-frequency interval
#ifqrangeHL domain      # restricts HL imaginary-frequency interval


#-----------------------#
# Gaussian calculations #
#-----------------------#
optmode     1           # 0:opt(z-matrix) , 1:opt(modredundant)
fccards     no          # Use LL Hessian in HL opt (yes/no)
charge      0           # charge of the system
multipl     1           # multiplicity of the system
lowlevel    hf    3-21g # low-level of calculation
highlevel   b3lyp 6-31G # high-level of calculation
nprocll     1           # Number of threads (low-level)
memll       1GB         # dynamic memory (low-level)
nprochl     1           # Number of threads (high-level)
memhl       1GB         # dynamic memory (high-level)


#-----------------------#
#  Partition functions  #
#-----------------------#
tempsPF     100   200   # temperatures (K) for part. functions
tempsPF     298.15      # temperatures (K) for part. functions
tempsPF     300   500   # temperatures (K) for part. functions
tempsPF     750  1000   # temperatures (K) for part. functions
tempsPF     2000 2500   # temperatures (K) for part. functions
freqscalLL  1.0         # freq. scaling factor (LL)
freqscalHL  1.0         # freq. scaling factor (HL)
sigmamj     0.02        # max value for sigma(Mj); >= 0.01


#-----------------------#
#        Storage        #
#-----------------------#
dirll       files_LL/   # folder to store LL conformers
dirhl       files_HL/   # folder to store HL conformers
tmpll       /scratch/david/LL_uRRXp/ # folder for LL temporal files
tmphl       /scratch/david/HL_uRRXp/ # folder for HL temporal files
```

Reference Z-matrix file for n-BuOH (WE1).

```
 h
 o     1  oh2
 c     2  co3          1  coh3
 c     3  cc4          2  cco4           1  dih4
 c     4  cc5          3  ccc5           2  dih5
 c     5  cc6          4  ccc6           3  dih6
 h     6  hc7          5  hcc7           4  dih7
 h     3  hc8          2  hco8           4  dih8
 h     3  hc9          2  hco9           4  dih9
 h     4  hc10         3  hcc10          5  dih10
 h     4  hc11         3  hcc11          5  dih11
 h     5  hc12         4  hcc12          6  dih12
 h     5  hc13         4  hcc13          6  dih13
 h     6  hc14         5  hcc14          7  dih14
 h     6  hc15         5  hcc15          7  dih15

oh2          0.950000
co3          1.380000
coh3         109.471
cc4          1.500000
cco4         109.471
dih4         180.000
cc5          1.500000
ccc5         109.471
dih5         180.000
cc6          1.500000
ccc6         109.471
dih6         180.000
hc7          1.070000
hcc7         109.471
dih7         180.000
hc8          1.070000
hco8         109.471
dih8         120.000
hc9          1.070000
hco9         109.471
dih9         240.000
hc10         1.070000
hcc10        109.471
dih10        120.000
hc11         1.070000
hcc11        109.471
dih11        240.000
hc12         1.070000
hcc12        109.471
dih12        120.000
hc13         1.070000
hcc13        109.471
dih13        240.000
hc14         1.070000
hcc14        109.471
dih14        120.000
hc15         1.070000
hcc15        109.471
dih15        240.000
```

Reference Z-matrix file for L-alanine (WE2).

```
 h
 o     1  oh2
 c     2  co3          1  coh3
 c     3  cc4          2  cco4           1  dih4
 c     4  cc5          3  ccc5           2  dih5
 h     5  hc6          4  hcc6           3  dih6
 o     3  oc7          2  oco7           4  dih7
 n     4  nc8          3  ncc8           5  dih8
 h     4  hc9          3  hcc9           5  dih9
 h     5  hc10         4  hcc10          6  dih10
 h     5  hc11         4  hcc11          6  dih11
 h     8  hn12         4  hnc12          3  dih12
 h     8  hn13         4  hnc13         12  dih13

oh2          0.950000
co3          1.380000
coh3         109.471
cc4          1.500000
cco4         109.471
dih4         180.000
cc5          1.500000
ccc5         109.471
dih5         180.000
hc6          1.070000
hcc6         109.471
dih6         180.000
oc7          1.380000
oco7         120.471
dih7         180.000
nc8          1.070000
ncc8         109.471
dih8         240.000
hc9          1.070000
hcc9         109.471
dih9         120.000
hc10         1.070000
hcc10        109.471
dih10        120.000
hc11         1.070000
hcc11        109.471
dih11        240.000
hn12         1.030000
hnc12        109.471
dih12        180.000
hn13         1.030000
hnc13        109.471
dih13        120.000
```

Reference Z-matrix file for proline (WE3).

```
 c
 c    1  cc2
 c    2  cc3         1  ccc3
 c    3  cc4         2  ccc4          1  dih4
 n    4  nc5         3  ncc5          2  dih5
 c    1  cc6         2  ccc6          5  dih6
 o    6  oc7         1  occ7          2  dih7
 h    7  ho8         6  hoc8          1  dih8
 o    6  oc9         1  occ9          7  dih9
 h    1  hc10        2  hcc10         5  dih10
 h    2  hc11        1  hcc11         3  dih11
 h    2  hc12        1  hcc12         3  dih12
 h    3  hc13        2  hcc13         4  dih13
 h    3  hc14        2  hcc14         4  dih14
 h    4  hc15        3  hcc15         5  dih15
 h    4  hc16        3  hcc16         5  dih16
 h    5  hn17        4  hnc17         1  dih17

cc2           1.500
cc3           1.500
ccc3        109.471
cc4           1.500
ccc4        109.471
dih4          0.000
nc5           1.500
ncc5        109.471
dih5          0.000
cc6           1.500
ccc6        109.471
dih6       -120.000
oc7           1.500
occ7        109.471
dih7        180.000
ho8           1.070
hoc8        109.471
dih8        180.000
oc9           1.380
occ9        109.471
dih9        180.000
hc10          1.070
hcc10       109.471
dih10       120.000
hc11          1.070
hcc11       109.471
dih11       120.000
hc12          1.070
hcc12       109.471
dih12       240.000
hc13          1.070
hcc13       109.471
dih13       120.000
hc14          1.070
hcc14       109.471
dih14       240.000
hc15          1.070
hcc15       109.471
dih15       120.000
hc16          1.070
hcc16       109.471
dih16       240.000
hn17          1.030
hnc17       109.471
dih17       240.000
```

Reference Z-matrix file for the H_{α}-abstraction by H in n-BuOH (WE4).

```
 h
 o    1  oh2
 c    2  co3        1  coh3
 c    3  cc4        2  cco4        1  dih4
 c    4  cc5        3  ccc5        2  dih5
 c    5  cc6        4  ccc6        3  dih6
 h    6  hc7        5  hcc7        4  dih7
 h    3  hc8        2  hco8        4  dih8
 h    3  hc9        2  hco9        4  dih9
 h    4  hc10       3  hcc10       5  dih10
 h    4  hc11       3  hcc11       5  dih11
 h    5  hc12       4  hcc12       6  dih12
 h    5  hc13       4  hcc13       6  dih13
 h    6  hc14       5  hcc14       7  dih14
 h    6  hc15       5  hcc15       7  dih15
xx    9  xxh16      3  xxhc16      2  dih16
 h    9  hh17      16  hhxx17      3  dih17

oh2         0.950000
co3         1.380000
coh3        109.471
cc4         1.500000
cco4        109.471
dih4        180.000
cc5         1.500000
ccc5        109.471
dih5        180.000
cc6         1.500000
ccc6        109.471
dih6        180.000
hc7         1.070000
hcc7        109.471
dih7        180.000
hc8         1.070000
hco8        109.471
dih8        120.000
hc9         1.288000
hco9        109.471
dih9        240.000
hc10        1.070000
hcc10       109.471
dih10       120.000
hc11        1.070000
hcc11       109.471
dih11       240.000
hc12        1.070000
hcc12       109.471
dih12       120.000
hc13        1.070000
hcc13       109.471
dih13       240.000
hc14        1.070000
hcc14       109.471
dih14       120.000
hc15        1.070000
hcc15       109.471
dih15       240.000
xxh16       1.000000
xxhc16       90.000
dih16         0.000
hh17        0.993000
hhxx17       90.000
dih17       180.000
```

Bibliography

[1] D. Ferro-Costas and A. Fernández-Ramos, *Front. Chem.*, 2020, **8**, 16.

[2] D. Ferro-Costas, I. Mosquera-Lois and A. Fernández-Ramos, *J. Cheminformatics*, Under review; Download from: https://www.researchsquare.com/article/rs-435544/v1.

[3] D. Ferro-Costas and A. Fernández-Ramos, *The Cathedral Package*, https://github.com/cathedralpkg (accessed November 23, 2021), 2021.

[4] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski and D. J. Fox, *Gaussian 09 Revision E.01*, 2009, Gaussian Inc. Wallingford CT.

[5] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman and D. J. Fox, *Gaussian 16 Revision B.01*, 2016, Gaussian Inc. Wallingford CT.

[6] J. Zheng, S. L. Mielke, K. L. Clarkson and D. G. Truhlar, *Comput. Phys. Commun.*, 2012, **183**, 1803–1812.

[7] J. Zheng, R. Meana-Pañeda and D. G. Truhlar, *Comput. Phys. Commun.*, 2013, **184**, 2032–2033.

[8] *Reviews in Computational Chemistry Volume 23*, ed. K. B. Lipkowitz and T. R. Cundari, Wiley-VCH, 2007.

[9] D. Ferro-Costas, E. Martínez-Núñez, J. Rodríguez-Otero, E. Cabaleiro-Lago, C. M. Estévez, B. Fernández, A. Fernández-Ramos and S. A. Vázquez, *J. Phys. Chem. A*, 2018, **122**, 4790–4800.

[10] J. Zheng, T. Yu, E. Papajak, I. M. Alecu, S. L. Mielke and D. G. Truhlar, *Phys. Chem. Chem. Phys.*, 2011, **13**, 10885–10907.

[11] J. Zheng and D. G. Truhlar, *J. Chem. Theory Comput.*, 2013, **9**, 1356–1367.

[12] J. L. Bao, R. Meana-Pañeda and D. G. Truhlar, *Chem. Sci.*, 2015, **6**, 5866–5881.

[13] G. Schaftenaar and J. Noordik, *J. Comput. Aided Mol. Des.*, 2000, **14**, 123–134.

[14] G. Schaftenaar, E. Vlieg and G. Vriend, *J. Comput. Aided Mol. Des.*, 2017, **31**, 789–800.