TorsiFlex v2022.1

A program for the conformational search in flexible molecules

David Ferro-Costas

Antonio Fernández-Ramos

Universidade de Santiago de Compostela (SPAIN)

July 12, 2022

About `TorsiFlex` License

July 2022

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 Software requirements 6
1.6 Electronic structure software 6
1.7 How to cite 6
1.8 Version history 8
1.8.1 List of changes in version 2022.1 8
1.8.2 List of changes in version 2021.3 8
1.8.3 List of changes in version 2021.2 8

2 The algorithm in a nutshell **9**

2.1 The low-level search 10
2.2 The high-level reoptimization 10
2.3 Validation tests 10

3 TorsiFlex inline options **13**

3.1 List of inline options 13
3.2 Low-level search with --prec and --stoc 14
3.2.1 Speeding up the search with --prec 14
3.2.2 Searching for a given conformer with --stoc 14
3.3 High-level optimization with --hlopt 14
3.3.1 Speeding up the HL search 15
3.4 Partition functions with --msho and --mstor 16
3.4.1 MS-HO partition functions 16
3.4.2 MS-T(C) partition functions 16

4 TorsiFlex input files **19**

4.1 Building the reference geometry file 20
4.2 The `torsiflex.inp` input file 22
4.2.1 Keywords 22
4.3 *Gaussian* templates 28
4.3.1 Modifying the *Gaussian* templates 28
4.4 The file with preconditioned angles: `pcfile` 30

5 Some tips **31**
5.1 Focusing on low energy conformers 31
5.2 Dealing with similar molecules 31

6 Worked examples **33**
6.1 **WE1**: conformers of *n*-butanol 36
6.2 **WE2**: conformers of L-alanine 38
6.3 **WE3**: conformers of proline 40
6.4 **WE4**: conformers of a transition state 42
6.5 Data for comparison 44
6.5.1 Number of conformers and partition functions 44
6.5.2 List of conformers for **WE1** 45
6.5.3 List of conformers for **WE2** 45
6.5.4 List of conformers for **WE3** 46
6.5.5 List of conformers for **WE4** 47

7 Example files **49**

Bibliography **58**

1. TorsiFlex software

This document is a guide to TorsiFlex, a program designed to find all the conformers of flexible molecules by applying a combination of preconditioned and stochastic algorithms.

Notice that the example files mentioned in this manual are listed in Chapter 7.

1.1 About TorsiFlex

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

TorsiFlex is a user-friendly code written in Python 3 and designed to search for all the conformers of a given molecule by adopting a combined low-level/high-level (LL/HL) strategy.[1,2] TorsiFlex is part of a bigger project that include other programs of interest: the Cathedral package. We encourage the user to visit our GitHub webpage to learn more about the software we are developing in our research 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 python3:

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

where path_to_torsiflex refers to the TorsiFlex directory. For the sake of simplicity, we will omit path_to_torsiflex/ hereinafter.

TorsiFlex can be directly executed provided that execution permissions are granted:

```
>> chmod u+x 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 in torsiflex.py to define the path of the Python interpreter:

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

1.5 Software requirements

Before running TorsiFlex the user should have installed the following Python libraries:

numpy	rdkit	scipy

We highlight that rdkit is only mandatory to execute TorsiFlex with the --smiles option.

1.6 Electronic structure software

At this moment, TorsiFlex can only perform electronic structure calculations using *Gaussian* (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"
```

1.7 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].

1.8 Version history

1.8.1 List of changes in version 2022.1

Important! This new version is incompatible with the previous ones.

- New execution option, `--smiles`, is now available.
- New execution option, `--cartesian`, is now available.
- New execution option, `--torsions`, is now available.
- A specific point of the torsional space can be calculated with `--stoc`.
- The information printed with `--msho` has been slightly modified.
- The `--msho` execution is no longer mandatory before `--hlopt`.
- *Gaussian* templates are now stored in a single file.
- Conformer files of the system (inside folders defined by `dirll` and `dirhl`) are no longer stored as *Gaussian* output files.
- Keyword `tempGibbs` has been disabled.
- Keyword `tempsPF` has been renamed as `temps`.
- Other minor changes.

1.8.2 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.

1.8.3 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.
- Other minor changes.

2. The algorithm in a nutshell

This Chapter briefly describes the algorithm implemented in `TorsiFlex`, 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).

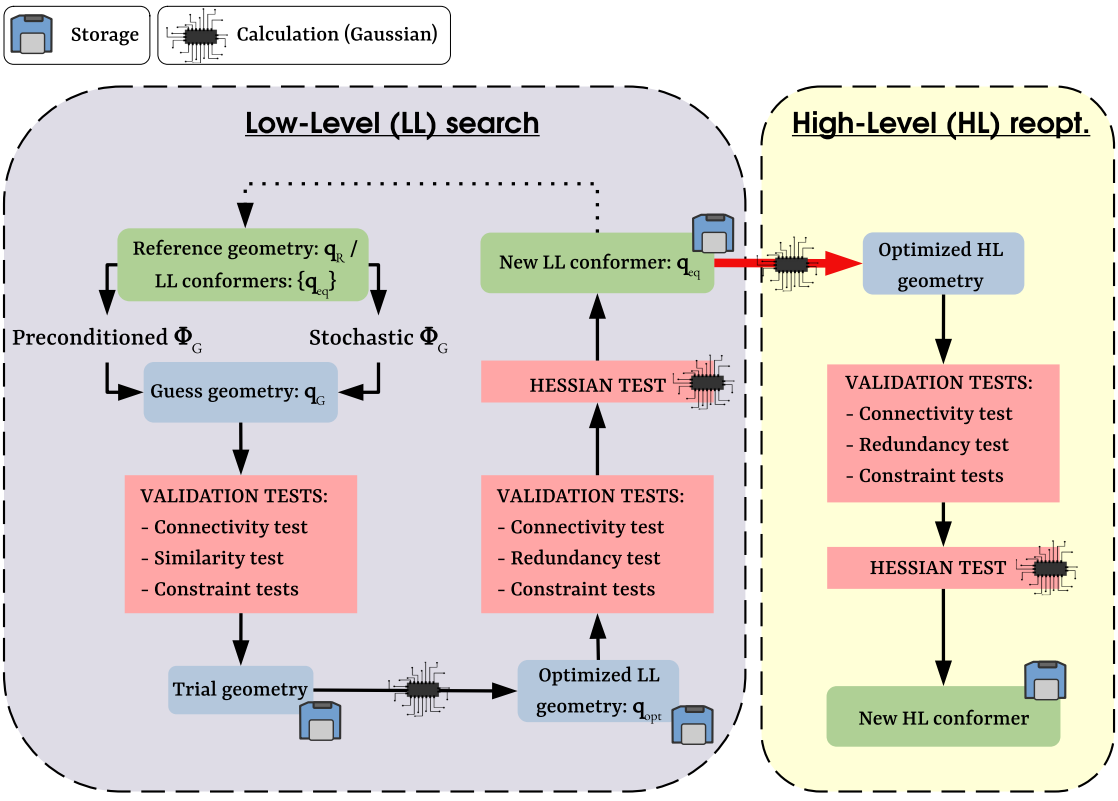


Figure 2.1: Flowchart of the search and reoptimization processes implemented in `TorsiFlex`.

2.1 The low-level search

TorsiFlex requires a reference geometry, $\mathbf{q}_R$, 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, $\mathbf{\Phi}^R$. The replacement of $\mathbf{\Phi}^R$ by a guess vector, $\mathbf{\Phi}^G$, renders a new configuration or guess geometry, $\mathbf{q}_G$.* For the generation of the reference Z-Matrix, see Section 4.1.

There are two strategies for the generation of $\mathbf{\Phi}^G$: 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°.

Each generated Z-Matrix, $\mathbf{q}_G$, is validated through a series of tests. The geometries that fail the tests are discarded, whereas those that pass them turn into trial structures that will be optimized geometrically. The low-level (LL) electronic structure optimizations allow performing inexpensive electronic structure calculations and the tests avoid doing unnecessary calculations, thus accelerating the search process.

The resulting optimized geometry, $\mathbf{q}_{opt}$, is validated through a new set of tests and, if all of them are passed, the Hessian matrix is calculated (Hessian test) to confirm that we are dealing with a conformer of the system. We refer to our previous works [1,2] for a more detailed explanation.

2.2 The high-level reoptimization

The LL search described in the previous section is intended for inexpensive electronic structure levels of calculation to reduce the computational cost, which speeds up the location of new conformers.

TorsiFlex can employ 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. After all tests are passed, the Hessian test is carried out to check that the HL optimized geometry corresponds to a equilibrium structure.

2.3 Validation tests

The following tests are implemented in TorsiFlex:

- **The connectivity test**: positive (passed) if the connectivity of the guess/optimized structure coincides with that of the reference Z-Matrix (i.e. this

*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 $\mathbf{\Phi}^G$ instead of resorting to the reference Z-Matrix, speeding up the geometry optimization.

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

test asserts that we are dealing with the same constitutional isomer).

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

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

  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, $\mathbf{\Phi}^{\mathrm{opt}}$, against the pool of stored conformers, $\{\mathbf{\Phi}^{\mathrm{eq}}\}$. If $\mathbf{\Phi}^{\mathrm{opt}}$ was not previously stored in $\{\mathbf{\Phi}^{\mathrm{eq}}\}$, *i.e.* $\mathbf{\Phi}^{\mathrm{opt}} \notin \{\mathbf{\Phi}^{\mathrm{eq}}\}$, the test results positive. From a practical point of view this test is positive if:

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

  where ϵ is a threshold (usually not greater than 2 degrees to account 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 distances between pairs of atoms, angle between triads of atoms and dihedral angles between four atoms. Specifically, `TorsiFlex` distinguishes between two types of constraints:

  - *hard* constraints; the test is positive if all the requirements for the constraints are fulfilled.
  - *soft* constraints; the test is positive if one or more of the requirements for the constraints are fulfilled.

- **The Hessian test**: performs a test on the Hessian matrix of the optimized geometry. If all the vibrational frequencies are real, the test is positive. When dealing with the conformers of a transition state, the test is passed when all the vibrational frequencies are real but one. Keywords `ifqrangeLL` and `ifqrangeHL` can be used to define a minimum value for the imaginary frequency at the LL and HL, respectively, for the geometry to be considered a valid transition state. Notice that the calculation of the Hessian matrix could be computer-time demanding.

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

3. TorsiFlex inline options

3.1 List of inline options

TorsiFlex accepts different *inline* commands through the syntax:

```
>> torsiflex.py [command]
```

where [command] refers to one of the following options:

- --smiles: generates the Z-Matrix from the SMILES code;
- --cartesian: builds the Z-Matrix from Cartesian coordinates;
- --input: generates the TorsiFlex standard input file, torsiflex.inp, and the file that interfaces with *Gaussian*, TemplatesGAUSSIAN.txt;
- --prec: performs the LL search by the preconditioned algorithm;
- --stoc: performs the LL search by the stochastic algorithm;
- --hlopt: 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 functions;
- --mstor: generates the MsTor input files;[6,7]
- --regen: regenerates the domains.txt file from scratch files;
- --torsions: allows modifying the selected torsions;
- --help: displays an *online* explanation of the options;
- --version: displays the version of TorsiFlex.

An extense explanation of the options in red can be found in this chapter, whereas options in blue will be explored in Chapter 4.

3.2 Low-level search with --prec and --stoc

TorsiFlex only searches for new conformers at the low level, and it can carry out the preconditioned conformer localization when executed together with --prec:

```
>> torsiflex.py --prec
```

or the stochastic search when executed with --stoc:

```
>> torsiflex.py --stoc
```

The difference between the two procedures lies on the generation of the starting structures (see section 2.1). Optimum performance is achieved when the preconditioned generation precedes the stochastic one. There are two reasons for this choice:

- geometry optimizations of preconditioned guesses rarely fail;
- two preconditioned guesses infrequently lead to the same conformer.

3.2.1 Speeding up the search with --prec

It is possible to speed up the preconditioned search by splitting the set of preconditioned guesses into G groups and by dealing with each individual group separately. Thus, TorsiFlex manages the g-*th* group (g $\leq$ G) with:

```
>> 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:

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

In this manner, the user can send several jobs at the same time.

3.2.2 Searching for a given conformer with --stoc

The search for a specific point in the torsional space can be achieved by defining it in the --stoc execution. For example, in a system with three torsions, the search for a geometry near (30°,40°,50°) can be carried out employing:

```
>> torsiflex.py --stoc 30_40_50
```

3.3 High-level optimization with --hlopt

Once the LL conformers were located, the HL conformers can be obtained with the --hlopt option:

```
>> torsiflex.py --hlopt
```

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

3.3.1 Speeding up the HL search

It is possible to execute `TorsiFlex` "in parallel" by creating groups of LL conformers that will be further optimized at the HL. The n1-*th* LL conformer (sorted by increasing total energy) can be re-optimized at HL with

```
>> torsiflex.py --hlopt n1
```

whereas

```
>> torsiflex.py --hlopt n1 n2
```

instructs `TorsiFlex` to re-optimize the n1-*th* to n2-*th* LL conformers, both included. The indices and relative energies of all LL conformers can be obtained by executing

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

Thus, the re-optimization of the five most stable LL conformers can be carried out with:

```
>> torsiflex.py --hlopt 1 5
```

Sometimes, it may be useful to optimize many HL at the same time. In such cases, we recommend to execute `TorsiFlex` as follows:

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

This will generate the *Gaussian* input files for the HL re-optimization of the LL conformers. In this manner, the user can optimize just the desired geometries. In this case, the calculations are managed by the user and not by `TorsiFlex`. Once the optimizations are complete, the corresponding *Gaussian* input files for frequency calculations can be also generated by executing `TorsiFlex` again:

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

When all optimization and frequency calculations are finished, `TorsiFlex` can be executed a third time to automatically read the generated output files an identify the conformers of the system:

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

3.4 Partition functions with --msho and --mstor

One of the main goals of TorsiFlex is the accurate evaluation of partition functions, because they are the gate toward thermodynamic functions. The program allows calculating:

- Multistructural harmonic-oscillator (MS-HO) partition functions
- Multistructural partition functions with coupled torsional anharmonicity [MS-T(C)]

3.4.1 MS-HO partition functions

The MS-HO partition function is just a sum over all harmonic-oscillator partition functions of the conformers weighted by the energy of the conformer.[10,11] The inline command to perform such calculation is:

```
>> torsiflex.py --msho
```

In this execution, TorsiFlex sorts the located conformers by increasing energy and prints diverse information about each conformer (like the corresponding rotational constants), as well as about the whole system. Gibbs free energies for each working temperature are also printed. This option also looks for possible redundant conformers to avoid repetitions.

TorsiFlex only evaluates the LL/HL MS-HO partition function if ll/hl is specified after --msho:

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

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

3.4.2 MS-T(C) partition functions

A more accurate method than MS-HO is MS-T(C), which incorporates couplings in the kinetic and potential energies between the torsions and between the torsions and the non-torsional degrees of freedom.[12–14] 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:

```
>> 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 program increases N_{tot} until all the standard errors of each M_j (σ_{M_j}):

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

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`.

`TorsiFlex` only generates LL/HL `MsTor` input files if `ll/hl` is specified after `--mstor`:

```
>> torsiflex.py --mstor ll
```

```
>> torsiflex.py --mstor hl
```

4. TorsiFlex input files

TorsiFlex needs the following files to work:

- a file with the reference geometry to initiate the searching procedure;
- the file containing the keywords that control the program (`torsiflex.inp`);
- a file that includes the *Gaussian* commands for carrying out the electronic structure calculations (`TemplatesGAUSSIAN.txt`);
- a file (optional) with torsional angles, which can be used instead the preconditional angles automatically defined inside `torsiflex.inp`.

This Chapter shows how to create these files.

4.1 Building the reference geometry file

The user must provide a file containing the Z-Matrix of the system with the reference geometry. There are three ways of creating this file:

- **From SMILES**

  The Z-Matrix file can be generated automatically from a SMILES code by executing:

  ```
  >> torsiflex.py --smiles "smiles_code" zmfile.zmat
  ```

  where `smiles_code` is the SMILES code of the system* and `zmfile.zmat` is the name of the file where the Z-Matrix will be stored. For example, for the n-butanol molecule, we can execute:

  ```
  >> torsiflex.py --smiles "CCCCO" buoh.zmat
  ```

  → `RdKit` Python library is required for this option to work.

- **From Cartesian coordinates**

  Similarly, the Z-Matrix file can be created automatically by `TorsiFlex` from a file containing the Cartesian coordinates with:

  ```
  >> torsiflex.py --cartesian ccfile.xyz zmfile.zmat
  ```

  where `ccfile.xyz` is the file where the Cartesian coordinates are stored. We highlight that the order of the atoms between the initial Cartesian coordinates and the final Z-Matrix will differ.

- **From scratch**

  The user can always build its own Z-Matrix. This will be the preferred possibility when dealing with transition state conformations. There are many programs that can help with this task but we recommend `Molden`.[8,9] This program allows building up the molecule and save its geometry in the Z-Matrix format by selecting the "`Gaussian`" option and by clicking on the "`Write Z-Matrix`" button (see Figure 4.1).

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

*Preferably given between quotation marks; this is mandatory if the SMILES code presents special characters like, for instance, brackets.

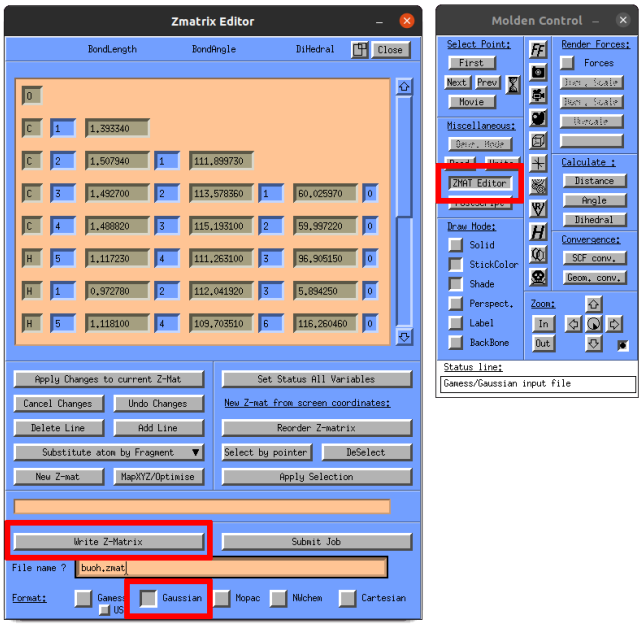


Figure 4.1: The Molden interface.

4.2 The `torsiflex.inp` input file

The input file for TorsiFlex is called `torsiflex.inp` and contains the keywords that control the program. This file is generated by executing:

```
>> torsiflex.py --input zmfile.zmat
```

where `zmfile.zmat` is the name of the Z-Matrix file. During the execution, TorsiFlex will ask for some basic information about the system:

- the type of stationary point (*i.e.*, a minimum or a transition state);
- the total charge;
- the spin multiplicity;
- the presence of torsional enantiomerism.

If the user skips the questions, the following default values will be assumed: minimum, no charge, singlet, and no torsional enantiomerism.

4.2.1 Keywords

The TorsiFlex input file (`torsiflex.inp`) contains different keywords, whose detailed description can be found below. These keywords are also listed in Table 4.1. The default input file can be found in Chapter 7.

System-specific keywords

- `zmatfile`: the file containing the Z-Matrix of the system. See section 4.1.
- `charge`: the total charge of the system.
  → Replaces the `[charge]` string in the *Gaussian* template.
- `multipl`: the spin multiplicity of the system.
  → Replaces the `[multipl]` string in the *Gaussian* template.
- `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 `skipcon` lines can be used and several pairs of atoms can be defined in each line. For instance:

```
skipcon (1,2) (4,5)
skipcon (5,9)
```

Keywords associated with the storage of conformers

- `dirll`: the folder where the LL conformers will be stored.
- `dirhl`: the folder where the HL conformers will be stored.
- `tmpll`: the folder where the temporal LL *Gaussian* files will be stored. By default this folder is set to `/scratch/user/LL_zmfile/` where `user` is the name of the user and `zmfile` is the name of the Z-Matrix file (without extension).
- `tmphl`: same as `tmpll` but for temporal HL *Gaussian* files.

Keywords associated with the targeted torsions

In the keywords below, `X` must be replaced by an integer (starting with 1).

- `torsionX`: the name of the internal coordinate in the Z-Matrix file associated with the `X`-*th* target torsion. For instance, if the first torsion of interest is labeled `dih8` in the Z-Matrix file, then we write:

  ```
  torsion1 dih8
  ```

- `precondX`: defines the preconditioned angles of `torsionX`. By default, these values are 60, 180 and 300 degrees.
- `tdomainX`: defines the allowed domain of `torsionX`; by default, [0,360).

  A domain can be defined by an interval `(a,b)`, with `a<b`. The union of several intervals can be specified by using a capital `U`. For example:

  ```
  tdomain1 (60,120)U(240,300)
  ```

  indicates that `torsion1` is restricted to the union of two intervals: (60,120) and (240,300).
- `tsigmaX`: the torsional symmetry number for `torsionX`. By default it is the unity.
- `pcfile`: a file containing preconditioned angles. It overrides `precondX`. See Section 4.4.

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).

Keywords associated with the HL re-optimization

- `hlcutoff`: a cutoff for HL calculations. Only LL conformers with relative total energy smaller than `hlcutoff` (in kcal/mol) are re-optimized at HL. If this keyword is deactivated, all LL conformers are re-optimized.

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 by its name in the Z-Matrix or can be defined by listing the involved atoms. As an example, for the latter, `1-2-3` would refer to the angle defined by the atoms labeled as 1, 2 and 3.

Domains are specified as indicated in tdomainX. Notice that here domains may refer to distances (in Angstrom) or to angles (in degrees).

We highlight 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: this keyword is exclusive of transition states. It restricts the imaginary frequency (LL conformers of transition states) to a defined interval (in cm^{-1}). When the imaginary frequency falls outside this interval, the structure is discarded. For example, with:

```
ifqrangeLL (700,1500)
```

we restrict to the conformers with imaginary frequencies between $700i$ and $1500i$ cm^{-1}.

- ifqrangeHL: same as ifqrangeLL but for HL conformers.

Keywords associated with the calculations with *Gaussian*

- optmode: defines the arguments of the opt keyword in the *Gaussian* input file. It can be:
  - 0: opt(z-matrix)
  - 1: opt(modreduntant).

  → Replaces the [optmode] string in the *Gaussian* template.

- 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).

  → Also replaces the [fccards] string by the LL Hessian matrix in the
  → *Gaussian* template for HL optimization.

- lowlevel: the low-level to be inserted in the *Gaussian* template.
  → Replaces the [level] string in the corresponding *Gaussian* template.

- highlevel: same as lowlevel but to define the level of calculation in the HL templates.
  → Replaces the [level] string in the corresponding *Gaussian* template.

- nproc: number of threads for *Gaussian* calculations.
  → Replaces the [nproc] string in the corresponding *Gaussian* template.
- procll: same as proc but it only applies to LL *Gaussian* templates.
  → Replaces the [nproc] string in the corresponding *Gaussian* template.
- prochl: same as proc but it only applies to HL *Gaussian* templates.
  → Replaces the [nproc] string in the corresponding *Gaussian* template.
- mem: dynamic memory for *Gaussian* calculations.
  → Replaces the [mem] string in the corresponding *Gaussian* template.
- memll: same as mem but it only applies to LL *Gaussian* templates.
  → Replaces the [mem] string in the corresponding *Gaussian* template.
- memhl: same as mem but it only applies to HL *Gaussian* templates.
  → Replaces the [mem] string in the corresponding *Gaussian* template.

Keywords associated with the partition functions

- temps: 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 unscaled (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 unscaled (i.e. freqscalHL is set to 1).
- sigmamj: the standard error in the calculation of the local periodicity parameters for MsTor (see Section 3.4.2). This value must be ≥ 0.01.

Table 4.1: Available keywords in the TorsiFlex input file. In the Table, 'X' must be replaced by an integer, starting with number one.

Keyword	(Type of) value	Description
zmatfile	string	path and name of the Z-Matrix file
charge	integer (in a.u.)	charge of the system
multipl	integer	spin multiplicity of the system
enantio	yes or no	are there torsional enantiomers?
ts	yes or no	is the structure a transition state?
cfactor	float	controls system connectivity
skipcon	pair(s) of atoms (a,b)	excludes pairs of atoms in connectivity test
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
torsionX	string	the label of the X-th torsion in the Z-Matrix file
precondX	a list of integers	preconditioned angles for torsionX e.g. precond1 0 180
tdomainX	a domain	domain of torsionX, e.g. tdomain2 (-60,60)
tsigmaX	integer	torsional symmetry number of torsionX
pcfile	string	path to the file with the preconditioned angles
ncycles	integer	number of cycles in stochastic search
hlcutoff	float (kcal/mol)	total energy cutoff for HL re-optimizations
testsG	four integers (0's or 1's)	tests for the guess geometries (0, off; 1, on)
testsO	four integers (0's or 1's)	tests for the optimized geometries (0, off; 1, on)
dist1D	integer (in degrees)	$\geq$ 2.0; domain about an angle
epsdeg	integer (in degrees)	$\leq$ 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 or 1	defines the opt *Gaussian* keyword
fccards	yes or no	use LL Hessian in HL optimization
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
temps	list of floats (Kelvin)	temperatures for the partition functions
freqscalLL	float	scaling factor for frequencies (LL conformers)
freqscalHL	float	scaling factor for frequencies (HL conformers)
sigmamj	float	$\geq$ 0.01; with the --mstor option

4.3 *Gaussian* templates

TorsiFlex also creates the file TemplatesGAUSSIAN.txt when executed inline with the --input option. This file interfaces *Gaussian* with TorsiFlex to carry out geometry optimizations and frequency calculations (OPT and FRQ) at low and high levels of calculation (LL and HL) for the target system, which may be a minimum or a saddle point structure in the potential energy surface (MIN or TS).

Inside TemplatesGAUSSIAN.txt there are eight templates. Each template can be found between the start_KEY and end_KEY indicators, KEY indicating the type of calculation:

- KEY=MINOPTLL for LL optimization of a minimum
- KEY=MINOPTHL for HL optimization of a minimum
- KEY=MINFRQLL for LL frequency calculation of a minimum
- KEY=MINFRQHL for HL frequency calculation of a minimum
- KEY=TSOPTLL for LL optimization of a transition state (TS)
- KEY=TSOPTHL for HL optimization of a TS
- KEY=TSFRQLL for LL frequency calculation of a TS
- KEY=TSFRQHL for HL frequency calculation of a TS

4.3.1 Modifying the *Gaussian* templates

These templates can be edited to include new *Gaussian* keywords and/or modify the current ones. However, **the keywords inside the square brackets**, which are for the internal use of TorsiFlex, should not be modified. For instance, the [zmat] indicator is automatically replaced by TorsiFlex with a guess Z-Matrix and **must not** be removed. All the indicators, listed below, can be controlled with the keywords in the TorsiFlex input file (Section 4.2):

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

Using external basis sets

An external basis set can be specified within the corresponding template by using the Gen *Gaussian* keyword as basis set in the torsiflex.inp file:

```
lowlevel    HF Gen
```

Additionally, the non-standard basis set should be indicated at the end of the corresponding *Gaussian* template. For example, the following input uses the 6-31G(d,p) basis set for C and H and the 6-31G‡ basis set for F and places an extra function only on center number 1:

***Gaussian* template with external basis set.**

```
...

start_MINOPTLL
%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]
C H 0
6-31G(d,p)
****
F 0
6-31G(d',p')
****
1 0
SP   1 1.00
 0.4380000000D-01  0.1000000000D+01  0.1000000000D+01
****

end_MINOPTLL

...
```

Speeding up Hartree-Fock calculations

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 to speed up the calculations:

```
scf=(tight,incore)
```

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

4.4 The file with preconditioned angles: pcfile

The preconditioned angles can be also introduced as plain text using the pcfile keyword. The first line in this file consists of the selected 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 remaining torsions, if any, will be defined according to the default preconditioned angles, *i.e.* the (60,180,300) triad, or according to precondX. In this manner, if the system contains three torsions (torsion1,torsion2,torsion3), the previous lines would lead to the following points:

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

5. Some tips

5.1 Focusing on low energy conformers

The user may also be interested into re-optimizing only the most stable conformers. In such cases, we recommend to use the `hlcutoff` keyword. Only those LL conformers with total energy below the value given to `hlcutoff` are optimized at HL.

For example, to optimize at HL the conformers that at LL have energies smaller or equal to 3 kcal/mol with respect to the conformer with the lowest energy, we need to include the following line in `torsiflex.inp`:

```
hlcutoff     3.0
```

5.2 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 to have similar conformations of the M moiety in A and B. 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 preconditioned angles through `pcfile` instead of through 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 *.zmat | 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 *.zmat | 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 BM torsional angles `torsion3` and `torsion4` are set to those in AM, whereas starting values for `torsion1` and `torsion2` are defined by the default triad, (60,180,300) or by `precondX`

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

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

6. Worked examples

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

- **WE1**: *n*-butanol
- **WE2**: L-alanine
- **WE3**: L-proline
- **WE4**: H abstraction transition state (*n*-BuOH + H)

Before proceeding, make sure that the `GauExe` variable with the path to your *Gaussian* version is exported (for example in your `.bashrc` file).

Information about the level of calculation

In the four examples, the low-level search procedure is carried out at the HF/3-21G level, whereas B3LYP/6-31G is employed as the accurate high-level (HL) method for geometry re-optimizations. These levels can be specified in the input file with:

```
lowlevel   HF     3-21G
highlevel  B3LYP  6-31G  int=ultrafine
```

Notice that, for the HL calculations, the `int=ultrafine` *Gaussian* keyword is included.

We also highlight that the LL calculations can be speed up by including the `incore` argument of the `scf` *Gaussian* option in `TemplatesGAUSSIAN.txt`:*

```
scf=(tight,incore)
```

*In the `MINOPTLL` and `MINFRQLL` sections.

Information about the reference geometry files

The Z-Matrix of the reference geometries can be created (i) from the corresponding SMILES code , (ii) from Cartesian coordinates files, or (iii) directly with `Molden`. For the former, we provide the SMILES codes for worked examples **WE1** to **WE3**:[†]

- **WE1**: `CCCCO`
- **WE2**: `C[C@@H](C(=O)O)N`
- **WE3**: `C1C[C@H](NC1)C(=O)O`

However, the instructions and results presented in this Chapter assume that the Z-Matrix files that can be found in the `tests/zmatrices/` folder were used. If these files are created from scratch, the name of the target torsions and the corresponding involved atoms may differ (so, please, take this issue into account).

Information about the stochastic search

The stochastic search tries, by default, 200 random points in the torsional space. This number could be reduced by modifying the `ncycles` keyword. For example:

```
ncycles     50
```

Obviously, this will decrease the chances of locating all the system conformers. However, as the located conformers are listed for each worked example (Section 6.5), it is possible to search near the specific missed points when executing `TorsiFlex` with the `--stoc` option. For example, in **WE1**, the conformer associated with the 305_086_086 torsional vector (or to its torsional enantiomer) may be missed along the stochastic search. In such a case, the user could run again the stochastic search until this conformer is located or, directly, execute:

```
>> torsiflex.py --stoc 305_086_086
```

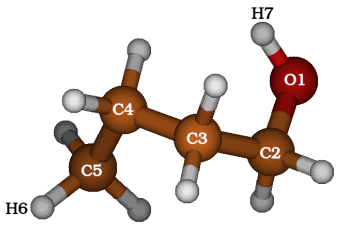
[†]Transition state structures cannot be generated from SMILES codes to our knowledge

Information about the systems of interest

The target torsions for each worked example can be found below (check also the atom labelling in the corresponding figure). The name of each torsion (and the corresponding involved atoms) agrees with those defined in the Z-Matrix files contained inside the `tests/zmatrices/` folder. If the user generates the Z-Matrix files from scratch, **the label of the target torsions and the atoms involved may differ.** In any case, the final number of conformers and their energies should be the same independently of the initial definition.

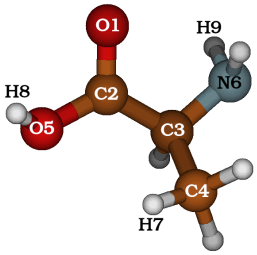
Torsions of interest for WE1

- `ptor04` (O1-C2-C3-C4)
- `ptor05` (C2-C3-C4-C5)
- `ptor07` (C3-C2-O1-H7)



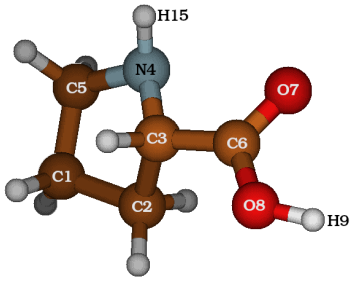
Torsions of interest for WE2

- `ptor04` (O1-C2-C3-C4)
- `ptor08` (O1-C2-O5-H8)
- `ptor09` (C2-C3-N6-H9)



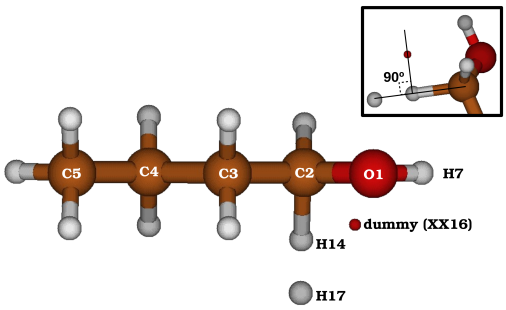
Torsions of interest for WE3

- `rtor04` (C1-C2-C3-N4)
- `rtor05` (C2-C3-N4-C5)
- `ptor07` (C2-C3-C6-O7)
- `ptor09` (C3-C6-O8-H9)
- `itor15` (C5-C3-N4-H15)



Torsions of interest for WE4

- `ptor04` (O1-C2-C3-C4)
- `ptor05` (C2-C3-C4-C5)
- `ptor07` (C3-C2-O1-H7)



6.1 WE1: conformers of *n*-butanol

In this worked example, we will find the conformers of *n*-butanol.

How to proceed

- Create the `WE1/` folder and enter it.

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

- Copy the corresponding Z-Matrix file (`we01.zmat`, see Chapter 7) from the `tests/` folder to the current directory (or create one).

- Create the input file and the templates for the *Gaussian* calculations:

```
>> torsiflex.py --inp we1.zmat
```

  In the options, the user should select minimum (`0`), total charge equal to `0`, spin multiplicity equal to `1`, and `yes` for torsional enantiomerism.

- Confirm the value of the following keywords in the input file:

```
zmatfile    we01.zmat
enantio     yes
torsion1    ptor04
torsion2    ptor05
torsion3    ptor07
```

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

```
>> torsiflex.py --prec
```

  In a few minutes, the program finds the conformers at the HF/3-21G level.

- Consider random angles to find new conformers with `--stoc`:

```
>> torsiflex.py --stoc
```

- List the LL conformers and calculate the corresponding MS-HO partition functions with the `--msho` option:

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

  Notice that a total of 15 LL conformers should be listed. We highlight that there is an enantiomer for any conformer, except for the one with Cs symmetry. Therefore, *n*-butanol presents 29 conformers ($14 \cdot 2 + 1$).

- Carry out the HL optimizations having as base the LL structures:

```
>> 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:

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

6.2 WE2: conformers of L-alanine

In this example, we will find the conformers of L-alanine.

How to proceed

- Create the folder `WE2/` and enter it.

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

- Copy the corresponding Z-Matrix file (`we02.zmat`, see Chapter 7) from the `tests/` folder to the current directory (or create one).
- Create the input file and the templates for the *Gaussian* calculations:

```
>> torsiflex.py --inp we2.zmat
```

  In the options, the user should select the minimum (`0`), total charge equal to `0`, spin multiplicity equal to `1`, and `no` to torsional enantiomerism.

- Check the following keywords in the input file:

```
zmatfile    we02.zmat
enantio     no
torsion1    ptor04
torsion2    ptor08
torsion3    ptor09
```

- Set the preconditioned angles for the carboxyl group torsions to 0 and 180 degrees:

```
precond2    0 180
```

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

```
>> torsiflex.py --prec
```

  In a few minutes, the program should find the conformers at the HF/3-21G level.

- Consider random angles to find new conformers with `--stoc`:

```
>> torsiflex.py --stoc
```

- List your LL conformers (a total of 11 conformers should be found along the whole LL search) and calculate the corresponding MS-HO partition functions with the `--msho` option:

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

- Carry out the HL optimizations based on the LL structures:

```
>> torsiflex.py --hlopt
```

This execution may take a few minutes.

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

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

The number of conformers should have been reduced to 10.

6.3 WE3: conformers of proline

In this example, we find the conformers of L-proline.

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. The ring configurations arise from two proper torsions, (i) C_1-C_2-C_3-N_4 and (ii) C_2-C_3-N_4-C_5, and from one improper torsion, C_5-C_3-N_4-H_{15}. In order to avoid huge distortions in the five-member ring, the domains of these torsions will be 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

- Create the folder `WE3/` and enter it.

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

- Copy the corresponding Z-Matrix file (`we03.zmat`, see Chapter 7) from the `tests/` folder to the current directory (or create one).

- Create the input file and the templates for the *Gaussian* calculations:

```
>> torsiflex.py --inp we3.zmat
```

  In the options, the user should select minimum (`0`), total charge equal to `0`, spin multiplicity equal to `1`, and `no` to torsional enantiomerism.

- Check the following keywords in the input file:

```
zmatfile    we03.zmat
enantio     no
torsion1    rtor04
torsion2    rtor05
torsion3    ptor07
torsion4    ptor09
```

- Modify the following keywords in the input file:

```
precond1     -30   +30
precond2     -30   +30
precond4       0   180
tdomain1    (-50,50)
tdomain2    (-50,50)
testsG       0 1 1 1
```

  Notice that we have reduced the interval for `torsion1` and `torsion2` to (-50,50), in order to generate geometries where the structure of the five-member ring is preserved. As in **WE2**, `precond4` accounts for the two expected conformations of the carboxylic acid group. Finally, the first argument

of `testsG` is set to `0` in order to deactivate the connectivity test on the guess structures.

- The internal coordinate associated with the NH_2 inversion was not detected by `TorsiFlex` because this motion is not represented by a proper torsion. Include this motion as the fifth target torsion:

```
torsion5    itor15
tdomain5    (-160,-110)U(110,160)
precond5    -120 +120
```

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

```
>> torsiflex.py --prec
```

  In a few minutes, the program will find the conformers at the HF/3-21G level from the preconditioned guesses.

- Consider random angles to find new conformers with `--stoc`:

```
>> torsiflex.py --stoc
```

- List your LL conformers (a total of 20 conformers should be found along the whole LL search) and calculate the corresponding MS-HO partition functions using `--msho`:

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

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

```
>> 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:

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

  The number of conformers should have been reduced to 14.

6.4 WE4: conformers of a transition state

In this example, we find the conformers of the transition state associated with the H abstraction by H in the α position of *n*-BuOH.

How to proceed

- Create the folder `WE4/` and enter it.

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

- Copy the corresponding Z-Matrix file (`we04.zmat`, see Chapter 7) from the `tests/` folder to the current directory (or create one).

- Create the input file and the templates for the *Gaussian* calculations:

```
>> torsiflex.py --inp we4.zmat
```

  In the options, select transition state (1), total charge equal to 0, spin multiplicity equal to 2, and no to torsional enantiomerism.

- Check the following keywords in the input file:

```
zmatfile    we04.zmat
multipl     2
ts          yes
enantio     no
torsion1    ptor04
torsion2    ptor05
torsion3    ptor07
```

- Add the following keywords in the input file:

```
skipcon      (2,14) (14,17)
ifqrangeLL   (500,3000)
```

  These two optional extra keywords are included to solve two specific problems that may arise. It may occur that the connectivity between H14 and H17 differs from one conformer to another due to a small variation in the distance between them. The `skipcon` keyword is employed to omit the two distances in the C···H···H moiety when carrying out the connectivity test.[‡] Additionally, the system may converge to an internal rotation transition state.[§] The `ifqrangeLL` keyword defines an expected range of values for the imaginary frequency, which in this case is between 500*i* and 3000*i* cm^{-1} for a H-abstraction transition-state at the HF/3-21G level.

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

```
>> torsiflex.py --prec
```

[‡]This could also be solved by modifying the `cfactor` keyword.
[§]For example, if the H radical goes away from butanol during the optimization.

In a few minutes, the program finds the conformers at the HF/3-21G level from the preconditioned guesses.

- Consider random angles to find new conformers with `--stoc`:

```
>> torsiflex.py --stoc
```

- List your LL conformers (a total of 25 conformers should be found along the whole LL search) and calculate the corresponding MS-HO partition functions using `--msho`:

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

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

```
>> 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:

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

The number of conformers should have been reduced to 19.

6.5 Data for comparison

In this section, several results for the worked examples are included. We encourage the user to compare the results with the ones presented in this section in order to check that `TorsiFlex` is working properly.

6.5.1 Number of conformers and partition functions

Table 6.1 lists the following data:

- n_{LL}: the number of conformers located in the LL search; it is split into those found by the preconditioned (n_{LL}^{prec}) and the stochastic searchings (n_{LL}^{stoc});
- n_{HL}: the final number of conformers after the HL re-optimization;
- Q^{MSHO}: the total MSHO rovibrational partition function calculated using the HL conformers at 100 and 2500 K.

Table 6.1: Selected data listed to check the performance of `TorsiFlex`. The list of conformers is given in the main text.

System	n_{LL}^{prec}	n_{LL}^{stoc}	n_{LL}	n_{HL}	Q^{MSHO}(100 K)	Q^{MSHO}(2500 K)
WE1	14	1	15	15	$1.6 \cdot 10^{5}$	$3.2 \cdot 10^{21}$
WE2	9	2	11	10	$5.6 \cdot 10^{4}$	$3.6 \cdot 10^{20}$
WE3	16	4	20	14	$1.5 \cdot 10^{5}$	$1.2 \cdot 10^{25}$
WE4	20	5	25	19	$1.3 \cdot 10^{5}$	$1.1 \cdot 10^{23}$

6.5.2 List of conformers for WE1

Low-level conformers

1. (056,184,065)
2. (060,183,184)
3. (054,070,064)
4. (064,287,063)
5. (177,180,064)
6. (058,069,183)
7. (062,179,285)
8. (180,180,180)
9. (067,282,182)
10. (181,069,297)
11. (174,069,065)
12. (058,065,283)
13. (177,068,179)
14. (305,086,086)
15. (084,299,292)

High-level conformers

1. (059,183,063)
2. (177,180,062)
3. (065,179,291)
4. (064,181,184)
5. (180,180,180)
6. (057,067,062)
7. (061,066,183)
8. (181,067,300)
9. (065,287,060)
10. (063,065,291)
11. (174,067,063)
12. (177,066,178)
13. (069,284,181)
14. (081,298,294)
15. (297,086,076)

6.5.3 List of conformers for WE2

Low-level conformers

1. (117,000,294)
2. (325,175,088)
3. (340,000,304)
4. (257,000,288)
5. (108,000,033)
6. (129,359,196)
7. (350,003,174)
8. (251,357,059)
9. (112,177,290)
10. (106,174,023)
11. (004,192,174)

High-level conformers

1. (317,178,097)
2. (114,000,295)
3. (020,000,295)
4. (256,000,290)
5. (107,000,027)
6. (130,000,196)
7. (344,002,175)
8. (258,357,058)
9. (112,179,294)
10. (109,177,016)

6.5.4 List of conformers for WE3

Low-level conformers

1. (028,354,302,358,130)
2. (035,342,114,180,230)
3. (010,326,119,181,232)
4. (010,330,318,354,122)
5. (038,335,339,179,224)
6. (013,325,336,179,231)
7. (022,003,008,186,139)
8. (325,016,281,007,129)
9. (022,002,222,176,137)
10. (324,021,003,184,138)
11. (351,342,256,182,226)
12. (356,340,026,186,125)
13. (353,343,231,175,125)
14. (323,024,120,180,154)
15. (331,041,338,355,243)
16. (034,345,110,354,231)
17. (004,330,117,357,233)
18. (353,340,006,012,228)
19. (038,327,098,352,128)
20. (025,320,214,352,224)

High-level conformers

1. (025,357,298,359,130)
2. (352,346,307,357,124)
3. (032,345,116,181,227)
4. (345,350,118,181,226)
5. (034,339,324,177,222)
6. (009,328,328,178,228)
7. (018,006,010,185,144)
8. (332,009,016,185,141)
9. (021,000,217,176,144)
10. (348,350,229,175,130)
11. (330,040,335,356,243)
12. (030,349,113,359,229)
13. (345,351,118,001,226)
14. (353,341,022,009,224)

6.5.5 List of conformers for WE4

Low-level conformers

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

High-level conformers

1. (063,182,051)
2. (066,181,186)
3. (178,180,049)
4. (302,179,183)
5. (180,180,184)
6. (300,182,052)
7. (060,066,050)
8. (069,290,049)
9. (064,066,186)
10. (180,293,048)
11. (074,287,185)
12. (183,293,184)
13. (311,295,184)
14. (308,296,052)
15. (298,077,184)
16. (170,069,050)
17. (170,068,181)
18. (282,061,048)
19. (306,088,057)

7. 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.

```
#--------------#
start_MINOPTLL
%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]

end_MINOPTLL
#~~~~~~~~~~~~~~#
```

Gaussian template for LL frequency calculation of a minimum.

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

--Frequency calculation--

[charge],[multipl]
[zmat]

end_MINFRQLL
#~~~~~~~~~~~~~~#
```

Gaussian template for HL optimization of a minimum.

```
#--------------#
start_MINOPTHL
%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]

end_MINOPTHL
#~~~~~~~~~~~~~~#
```

Gaussian template for HL frequency calculation of a minimum.

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

--Frequency calculation--

[charge],[multipl]
[zmat]

end_MINFRQHL
#~~~~~~~~~~~~~~#
```

Gaussian template for LL optimization of a transition state.

```
#--------------#
start_TSOPTLL
%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]

end_TSOPTLL
#~~~~~~~~~~~~~~#
```

Gaussian template for HL optimization of a transition state.

```
#--------------#
start_TSOPTHL
%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]

end_TSOPTHL
#~~~~~~~~~~~~~~#
```

Gaussian template for LL frequency calculation of a transition state.

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

--Frequency calculation--

[charge],[multipl]
[zmat]

end_TSFRQLL
#~~~~~~~~~~~~~~#
```

Gaussian template for HL frequency calculation of a transition state.

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

--Frequency calculation--

[charge],[multipl]
[zmat]

end_TSFRQHL
#~~~~~~~~~~~~~~#
```

The default `TorsiFlex` input file

```
# This is a torsiflex input file

#------------------------#
#         System         #
#------------------------#
zmatfile     zmfile.zmat # Z-matrix file
charge       0           # charge of the system
multipl      1           # multiplicity of the system
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


#------------------------#
#         Storage        #
#------------------------#
dirll        files_LL/   # folder to store LL conformers
dirhl        files_HL/   # folder to store HL conformers
tmpll        /scratch/user/LL_zmfile/ # folder for LL temporal files
tmphl        /scratch/user/HL_zmfile/ # folder for HL temporal files


#------------------------#
#    Target torsions     #
#------------------------#
torsion1    --           # name of target torsion number 1 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)


#------------------------#
#   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       7           # 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)
lowlevel     HF    3-21G               # low-level of calculation
highlevel    B3LYP 6-31G int=ultrafine # 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   #
#------------------------#
temps      100.00 200.00 # temperatures (K) for part. functions
temps      298.15 300.00 # temperatures (K) for part. functions
temps      500.00 750.00 # temperatures (K) for part. functions
temps            1000.00 # temperatures (K) for part. functions
temps            2000.00 # temperatures (K) for part. functions
temps            2500.00 # temperatures (K) for part. functions
freqscalLL   1.000       # freq. scaling factor (LL)
freqscalHL   1.000       # freq. scaling factor (HL)
sigmamj      0.02        # max value for sigma(Mj); >= 0.01
```

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

```
 O
 C     1    dist02
 C     2    dist03    1    angl03
 C     3    dist04    2    angl04    1    ptor04
 C     4    dist05    3    angl05    2    ptor05
 H     5    dist06    4    angl06    3    ptor06
 H     1    dist07    2    angl07    3    ptor07
 H     5    dist08    4    angl08    6    itor08
 H     5    dist09    4    angl09    6    itor09
 H     4    dist10    3    angl10    5    itor10
 H     4    dist11    3    angl11    5    itor11
 H     3    dist12    2    angl12    4    itor12
 H     3    dist13    2    angl13    4    itor13
 H     2    dist14    1    angl14    3    itor14
 H     2    dist15    1    angl15    3    itor15

dist02       1.39334
dist03       1.50794
angl03     111.89973
dist04       1.49270
angl04     113.57836
ptor04      60.02597
dist05       1.48882
angl05     115.19310
ptor05      59.99722
dist06       1.11723
angl06     111.26310
ptor06      96.90515
dist07       0.97278
angl07     112.04192
ptor07       5.89425
dist08       1.11810
angl08     109.70351
itor08     116.26046
dist09       1.09806
angl09     115.22337
itor09    -121.88996
dist10       1.10953
angl10     107.54313
itor10     120.65829
dist11       1.11452
angl11     107.39874
itor11    -121.41645
dist12       1.12433
angl12     105.58783
itor12    -115.97036
dist13       1.10922
angl13     111.04757
itor13     130.28396
dist14       1.11703
angl14     109.21413
itor14    -117.61743
dist15       1.08847
angl15     107.72248
itor15     125.98384
```

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

```
 O
 C     1    dist02
 C     2    dist03    1    angl03
 C     3    dist04    2    angl04    1    ptor04
 O     2    dist05    1    angl05    3    itor05
 N     3    dist06    2    angl06    4    itor06
 H     4    dist07    3    angl07    2    ptor07
 H     5    dist08    2    angl08    1    ptor08
 H     6    dist09    3    angl09    2    ptor09
 H     4    dist10    3    angl10    7    itor10
 H     4    dist11    3    angl11    7    itor11
 H     3    dist12    2    angl12    4    itor12
 H     6    dist13    3    angl13    9    itor13

dist02       1.25669
dist03       1.46513
angl03     119.17335
dist04       1.50890
angl04     113.16170
ptor04     125.64760
dist05       1.38725
angl05     116.91927
itor05     179.99916
dist06       1.43372
angl06     110.90762
itor06    -125.65195
dist07       1.11313
angl07     111.66282
ptor07      13.15326
dist08       1.01115
angl08     117.10283
ptor08     -45.79430
dist09       1.05936
angl09     107.84616
ptor09     -64.43562
dist10       1.11175
angl10     107.03225
itor10    -115.25979
dist11       1.11505
angl11     110.52000
itor11     124.14704
dist12       1.11771
angl12     107.28476
itor12     117.76441
dist13       1.03906
angl13     108.66622
itor13     122.22573
```

Reference Z-Matrix file for proline (WE3).

```
 C
 C     1    dist02
 C     2    dist03    1    angl03
 N     3    dist04    2    angl04    1    rtor04
 C     4    dist05    3    angl05    2    rtor05
 C     3    dist06    2    angl06    4    itor06
 O     6    dist07    3    angl07    2    ptor07
 O     6    dist08    3    angl08    7    itor08
 H     8    dist09    6    angl09    3    ptor09
 H     1    dist10    2    angl10    5    itor10
 H     1    dist11    2    angl11    5    itor11
 H     2    dist12    1    angl12    3    itor12
 H     2    dist13    1    angl13    3    itor13
 H     3    dist14    2    angl14    4    itor14
 H     4    dist15    3    angl15    5    itor15
 H     5    dist16    4    angl16    1    itor16
 H     5    dist17    4    angl17    1    itor17

dist02       1.53610
dist03       1.53270
angl03      98.21009
dist04       1.45095
angl04      99.08367
rtor04     -52.81725
dist05       1.44026
angl05     105.03107
rtor05      44.90310
dist06       1.45545
angl06     112.82258
itor06    -120.47697
dist07       1.25589
angl07     120.38173
ptor07     111.76265
dist08       1.38349
angl08     120.30303
itor08    -179.99788
dist09       1.01368
angl09     122.71892
ptor09     163.98786
dist10       1.12971
angl10     105.37559
itor10    -115.40032
dist11       1.09104
angl11     112.23178
itor11     121.80855
dist12       1.10777
angl12     107.46667
itor12    -113.31882
dist13       1.08457
angl13     112.42331
itor13     119.02609
dist14       1.10951
angl14     106.61485
itor14     113.38453
dist15       1.03622
angl15     113.22849
itor15     124.26138
dist16       1.11402
angl16     109.38802
itor16     122.61645
dist17       1.11653
angl17     108.53316
itor17    -121.16555
```

Reference Z-Matrix file for the H_α-abstraction by H in n-BuOH (WE4).

```
 O
 C     1    dist02
 C     2    dist03    1    angl03
 C     3    dist04    2    angl04    1    ptor04
 C     4    dist05    3    angl05    2    ptor05
 H     5    dist06    4    angl06    3    ptor06
 H     1    dist07    2    angl07    3    ptor07
 H     5    dist08    4    angl08    6    itor08
 H     5    dist09    4    angl09    6    itor09
 H     4    dist10    3    angl10    5    itor10
 H     4    dist11    3    angl11    5    itor11
 H     3    dist12    2    angl12    4    itor12
 H     3    dist13    2    angl13    4    itor13
 H     2    dist14    1    angl14    3    itor14
 H     2    dist15    1    angl15    3    itor15
 XX   14    dist16    2    angl16    1    ptor16
 H    14    dist17   16    angl17    2    itor17

dist02       1.39334
dist03       1.50794
angl03     111.89973
dist04       1.49270
angl04     113.57836
ptor04      60.02597
dist05       1.48882
angl05     115.19310
ptor05      59.99722
dist06       1.11723
angl06     111.26310
ptor06      96.90515
dist07       0.97278
angl07     112.04192
ptor07       5.89425
dist08       1.11810
angl08     109.70351
itor08     116.26046
dist09       1.09806
angl09     115.22337
itor09    -121.88996
dist10       1.10953
angl10     107.54313
itor10     120.65829
dist11       1.11452
angl11     107.39874
itor11    -121.41645
dist12       1.12433
angl12     105.58783
itor12    -115.97036
dist13       1.10922
angl13     111.04757
itor13     130.28396
dist14       1.28800
angl14     109.21413
itor14    -117.61743
dist15       1.08847
angl15     107.72248
itor15     125.98384
dist16       1.00000
angl16      90.00000
ptor16       0.00000
dist17       0.99300
angl17      90.00000
itor17     180.00000
```

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*, 2021, **13**, 100.

[3] D. Ferro-Costas and A. Fernández-Ramos, *The Cathedral Package*, https://github.com/cathedralpkg (accessed June 28, 2022), 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] G. Schaftenaar and J. Noordik, *J. Comput. Aided Mol. Des.*, 2000, **14**, 123–134.

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

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

[11] 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.

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

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

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