```
###############################################################
#                                                             #
#      $$$$$$\   $$$$$$\  $$$$$$$\ $$$$$$$$\                   #
#     $$  __$$\ $$  __$$\ $$  __$$\\__$$  __|                  #
#     $$ /  $$ |\__/  $$ |$$ |  $$ |  $$ | $$$$$$\   $$$$$$\   #
#     $$ |  $$ | $$$$$$  |$$ |  $$ |  $$ |$$  __$$\ $$  __$$\  #
#     $$ |  $$ |$$  ____/ $$ |  $$ |  $$ |$$ /  $$ |$$ /  \__| #
#     $$ $$\$$ |$$ |      $$ |  $$ |  $$ |$$ |  $$ |$$ |       #
#     \$$$$$$ / $$$$$$$$\ $$$$$$$  |  $$ |\$$$$$$  |$$ |       #
#      \___$$$\ _____|_____/   \__| _____/ \__|      #
#          \___|                                              #
#                                                             #
###############################################################
```

Q2DTor v2020.1

Quantum Two-Dimensional Torsions

(version 2020.1)

David Ferro-Costas[1]

M. Natália D. S. Cordeiro[2]

Donald G. Truhlar[3]

Antonio Fernández-Ramos[1]

[1] Universidade de Santiago de Compostela (SPAIN)

[2] Universidade de Porto (Portugal)

[3] University of Minnesota (USA)

The Cathedral Package

About Q2DTor License

MIT/X11 License

Copyright (c) 2020 David Ferro Costas (david.ferro@usc.es) and Antonio Fernández Ramos (qf.ramos@usc.es)

February 2020

Contents

The `Q2DTor` **software**

1.1   About `Q2DTor`

`Q2DTor` (Quantum Two-Dimensional Torsions) is a program designed to calculate partition functions and thermodynamic properties of molecular systems with two or more torsional modes. It allows calculating rotational-vibrational (or rovibrational) partition functions and thermodynamic functions by the multistructural harmonic oscillator (MS-HO) [1, 2] and the Extended Two-Dimensional Torsion (E2DT) [3] methods. If the molecule has more than two torsions, the program treats only two of them as anharmonic, with the rest of them as harmonic.

`Q2DTor` has to be executed in several steps, each one of them performing a different task. With this "step-by-step" procedure the user can check the output file after a given task (and before proceeding to the next one).

1.2   How to cite `Q2DTor`

D. Ferro-Costas, M. N. D. S. Cordeiro, D. G. Truhlar, A. Fernández-Ramos, *Comput. Phys. Commun.* **232**, 190-205 (2018).

1.3   The Cathedral Package

`Q2DTor` v2020.1 belongs to a bigger repository named **The Cathedral Package**:

   https://github.com/cathedralpkg/

The following codes are part of this repository:

- *Pilgrim*

- `Q2DTor`

- *TorsiFlex* (to be released)

1.4   How to obtain `Q2DTor`?

`Q2DTor` can be downloaded from the `cathedralpkg` GitHut webpage:

> https://github.com/cathedralpkg/q2dtor/releases

1.5   Terms of use

`Q2DTor` is free software under MIT/X11 license. We refer to the GitHub webpage for more details about the license:

> https://github.com/cathedralpkg/q2dtor

1.6   Python modules

The source code of `Q2DTor` is written in Python 3, and it uses the following modules:

(a) argparse, cmath, math, os, random, shutil, sys, time, and warnings

(b) matplotlib, numpy, pylab, and scipy

Whereas the (a) modules are generally distributed with the standard Python package, the (b) modules may not be included and need to be installed.

1.7   Software contents

The `q2dtor-2020.1.tar.gz` file contains the README.md file and three folders (`src/`, `tests/`, and `docs/`).

The `src/` folder contains all the Python files needed to run the program, i.e., the `q2dtor.py` file and the directory `modq2dtor/`. A brief description of them is given below:

- q2dtor.py is the main file.

- Files inside modq2dtor/:

    - classes.py includes different Python classes.

    - constants.py contains physical constants, conversion factors, and Python dictionaries with the atomic masses and symbols and covalent radii.

    - gtsfile.py contains functions that read and write a gts file (the format used by `Q2DTor`).

- `helpfns.py` contains helper functions for specific tasks.

- `mesc_X.py` files, which control the calls to the electronic structure software (X = 'gaussian' or 'orca').

- `quotes.py` contains some quotes, which are printed at the end of the output file.

- `tesselation.py` carries out interpolations using Delaunay triangles.

The `tests/` folder contains the output files of the tests set and an script to run them. The `docs/` folder contains this manual.

1.8   The electronic structure software

`Q2DTor` uses the electronic structure software (ESSO) to calculate the two-dimensional potential energy surface (2D-PES) associated with the target torsions, and to optimize the geometries of the 2D-PES stationary points. Currently, `Q2DTor` can handle the following packages:

- *Gaussian* (versions 03, [4] 09 [5] and 16 [6]), whose interface is implemented in the `mesc_gaussian.py` module.

- *Orca* [7] (versions 3.x and 4.x), implemented in the `mesc_orca.py` module.

In order to interact with *Gaussian* and/or Orca, `Q2DTor` needs to know the location of some executable files. Such information is obtained from the following environment variables, which have to be defined and exported by the user in their *.bashrc* file:

- Environment variables for Gaussian software:

  – `GauExe`: with the path to the Gaussian executable

  – `GauFchk`: with the path to the formchk tool

- Environment variable for Orca software:

  – `OrcaExe`: with the path to the Orca executable

For example:

```
export GauExe="/home/programs/g09/g09"
export GauFchk="/home/programs/g09/formchk"
export OrcaExe="/home/programs/orca_4_0_1_2/orca"
```

Different ESSOs could be implemented in a straghtforward way by a Python programmer.

1.9   How to run the test set

The test set contains 20 molecular systems, formed by the same molecules studied in Ref. [3]. In the `tests/` directory there is a script that facilitates running the tests set (`Q2DTorTesting.py`). The script runs under Python version 3.x. and it is executed by typing:

```
python3.6 Q2DTorTesting.py
```

This action opens the following interactive menu:

```
|   << Test creator for Q2DTor >>
|
|      Select case:
|       (1) HF/sto-3g (GAUSSIAN)
|       (2) HF/sto-3g (ORCA)
|
|      your choice:
```

where the user should choose case (1) to run the tests with *Gaussian* or (2) to run them with *Orca*. Once the software is selected, the menu displays:

```
|      Select action:
|       (1) Create input files
|       (2) Check number of Stationary Points
|       (3) Check results against refdata file
|       ..  to exit
```

Selecting action (1), the corresponding folder (`GAUSSIAN/` or `ORCA/`) containing the 20 test folders (`SXX/` with `XX` = 01,...,20) is created. In each `SXX/` directory, the user will find two files: the `xyz` file with a reference geometry in Cartesian coordinates and the corresponding `Q2DTor` input file.

After executing `q2dtor.py` for a given system (or for all of them), the user can check the results against the reference file using action (2) and (3).

1.9.1   The stationary points of the test set

It is possible that some critical points are not found when executing with the `--findsp` argument because: (i) the search algorithm implemented in `Q2DTor` may fail or (ii) the `tolerance` keyword is not small enough. For comparison, we list the stationary points of the test set in Table 1.1.

Table 1.1: Stationary point (SP) torsional angles (in degrees) of each molecule in the test set. The list is reduced according to the symmetry of the system.

System	ϕ_1	ϕ_2	Type of SP
S01	180.00	180.00	minimum
	0.00	180.00	minimum
	180.00	0.00	minimum
	92.38	179.83	saddle
	173.98	90.17	saddle
	0.00	0.00	minimum
	6.67	72.99	saddle
	82.33	13.33	saddle
	88.04	82.07	maximum
	270.81	77.40	maximum
S02	3.31	214.80	minimum
	0.00	180.00	saddle
	149.01	156.24	minimum
	180.00	180.00	saddle
	74.30	173.55	saddle
	0.00	0.00	saddle
	114.45	7.27	saddle
	320.32	19.22	maximum
	85.14	5.96	maximum
	180.00	0.00	maximum
S03	0.00	0.00	minimum
	180.00	180.00	minimum
	197.44	81.21	minimum
	187.43	125.18	saddle
	281.41	64.52	saddle
	180.00	0.00	saddle
	0.00	180.00	minimum
	6.11	222.40	saddle
	60.27	191.24	saddle
	68.76	66.22	saddle
	61.99	219.37	maximum
	78.98	13.29	maximum
	60.02	124.43	maximum
S04	62.63	55.04	minimum
	180.00	59.36	minimum
	121.59	55.83	saddle
	0.00	60.14	saddle
	180.00	0.00	saddle
	292.50	4.73	saddle
	238.28	3.48	maximum
	0.00	0.00	maximum

S05	0.00	0.00	minimum
	180.00	0.00	minimum
	0.00	180.00	minimum
	92.48	0.21	saddle
	159.48	159.20	minimum
	180.00	180.00	saddle
	169.40	107.10	saddle
	3.77	258.14	saddle
	90.15	182.63	saddle
	260.31	95.60	maximum
	90.47	97.09	maximum
S06	0.00	0.00	minimum
	180.00	180.00	minimum
	180.00	0.00	minimum
	182.82	74.45	saddle
	0.00	180.00	minimum
	11.16	246.82	saddle
	261.79	9.93	saddle
	87.84	201.77	saddle
	266.31	84.39	maximum
	88.96	77.11	maximum
S07	0.00	0.00	minimum
	199.72	151.42	minimum
	180.00	180.00	saddle
	7.11	240.72	minimum
	2.53	285.16	saddle
	180.00	0.00	saddle
	73.64	236.77	saddle
	0.00	180.00	saddle
	242.04	64.51	saddle
	271.25	67.19	maximum
	85.79	125.48	saddle
	100.79	73.42	maximum
	76.15	171.69	maximum
S08	0.00	60.81	minimum
	0.00	0.00	saddle
	149.74	54.23	minimum
	180.00	61.04	saddle
	180.00	0.00	saddle
	75.11	63.02	saddle
	90.10	2.64	maximum
S09	11.16	67.99	minimum
	227.80	60.91	minimum
	0.00	180.00	minimum
	2.26	225.49	saddle

	139.89	175.97	minimum
	128.11	63.63	minimum
	161.87	72.10	saddle
	180.00	180.00	saddle
	118.79	14.40	saddle
	216.62	133.54	saddle
	137.99	118.58	saddle
	77.47	49.69	saddle
	174.65	126.43	maximum
	0.00	0.00	saddle
	307.73	50.83	saddle
	313.70	28.45	maximum
	69.09	178.96	saddle
	180.00	0.00	maximum
	65.81	127.82	maximum
	69.00	235.75	maximum
S10	0.00	0.00	minimum
	0.00	60.17	saddle
	180.00	0.00	minimum
	88.62	1.23	saddle
	166.58	60.60	saddle
	180.00	60.02	maximum
	89.68	62.15	maximum
S11	0.00	0.00	minimum
	0.00	60.73	saddle
	208.97	3.71	minimum
	180.00	0.00	saddle
	77.11	8.29	saddle
	180.00	59.49	saddle
	81.37	67.24	maximum
S12	180.00	0.00	minimum
	180.00	60.20	saddle
	0.00	0.00	minimum
	0.00	60.18	saddle
	91.77	2.27	saddle
	93.13	61.67	maximum
S13	180.00	0.00	minimum
	43.50	8.86	minimum
	179.58	165.01	minimum
	180.00	180.00	saddle
	185.40	74.20	saddle
	249.24	7.93	saddle
	0.00	180.00	minimum
	0.00	0.00	saddle
	14.33	95.01	saddle
	90.23	188.07	saddle

	84.43	258.98	maximum
	97.89	91.25	maximum
S14	180.00	0.00	minimum
	167.48	129.99	minimum
	175.57	86.42	saddle
	28.56	1.56	minimum
	0.00	0.00	saddle
	180.00	180.00	saddle
	40.81	212.75	minimum
	29.85	281.08	saddle
	0.00	180.00	saddle
	100.49	0.71	saddle
	100.13	185.06	saddle
	99.32	81.47	maximum
	254.26	80.68	maximum
S15	0.00	0.00	minimum
	0.00	58.76	saddle
	180.00	0.00	minimum
	180.00	60.40	saddle
	265.73	1.21	saddle
	92.88	58.96	maximum
S16	116.65	58.76	minimum
	0.00	60.28	minimum
	180.00	59.98	saddle
	50.90	58.63	saddle
	119.35	358.63	saddle
	0.00	0.00	saddle
	180.00	0.00	maximum
	303.80	0.21	maximum
S17	0.00	0.00	minimum
	60.30	0.00	saddle
	59.88	59.88	maximum
S18	180.00	0.00	minimum
	180.00	180.00	minimum
	32.42	0.08	minimum
	0.00	0.00	saddle
	32.79	179.45	minimum
	0.00	180.00	saddle
	100.56	359.63	saddle
	100.57	180.05	saddle
	180.14	92.58	saddle
	38.98	92.09	saddle
	39.22	266.74	saddle
	0.96	267.32	maximum
	257.67	92.84	maximum
	102.62	92.72	maximum

S19	25.09	65.84	minimum
	0.00	180.00	minimum
	2.66	222.53	saddle
	88.19	0.00	saddle
	130.43	66.81	saddle
	88.63	180.00	saddle
	89.93	128.31	maximum
	0.00	0.00	maximum
S20	180.00	59.40	minimum
	180.00	0.00	saddle
	0.00	60.71	minimum
	92.64	49.23	saddle
	0.00	0.00	saddle
	309.03	15.06	maximum

1.10   Any question? Contact us!

We recommend to follow our Cathedral Package Facebook web page for updates, questions and other program information:

> https://www.facebook.com/cathedralpkg/

1.11   Version history

1.11.1   Version 2020.1

- Q2DTor is now part of a bigger project, **The Cathedral Package**;

- Q2DTor has been ported to Python 3;

- Now the version name is based on the release year;

- The path to the electronic structure software is now obtained from the environment variables instead of from the mesc.txt file.

1.11.2   Version 1.1

- License was changed from GPL-3.0 to MIT/X11.

- Corrected a bug in the function that reads the xyz file (problem with empty lines).

- Corrected a bug related to the calculation of partition functions in systems with the 'a' symmetry condition.

- For systems with 'b' type symmetry, odd symmetry terms of the Fourier series are automatically removed, even if they are specified by the user in the input file.

- If the total torsional symmetry number is not equal to 1, the program assumes that the system meets the 'c' type condition of symmetry even if that it is not specified in the input file.

- Corrected a bug in the calculation of S and G for MSHO.

- The version of Q2DTor is printed after the logo of the program.

- The date is added at the beginning of each execution.

- The `q2dtor.py` executable and its modules are now separated. Modules are stored in the `modq2dtor/` folder.

- If the internal symmetry number of a torsion is not 1, the corresponding Fourier terms of that dihedral are multiplies of that symmetry number. Other terms are automatically removed even if they are specified in the input file.

- By default, the scan of the PES starts at the closest geometry given in the input instead of at $(\phi_1, \phi_2) = (0,0)$.

- Option --SI is now available. The option prints the thermodynamic properties in the International System of Units.

- A new table for each thermodynamic function was added.

`Q2DTor` **input files**

`Q2DTor` needs two input files to run properly. Those files have `.inp` (the main input file) and `.xyz` (the Cartesian coordinates file) extensions. Instructions about how to generate those files are given below.

2.1   How to generate the main input file

The input file `name_system.inp`, where `name_system` is a word given by the user, is automatically generated by `Q2DTor`. For instance, in the case of the example test S19 we can use as `name_system` the word S19. Thus, by typing

```
python q2dtor.py S19
```

the program creates an input file called `S19.inp`. To run the tests provided with the program the user should skip this step because the input files should be created with the `Q2DTorTesting.py` script.

This section describes the different parts of a standard `Q2DTor` input file; the examples used to illustrate the usage are taken from the benzyl alcohol test (test molecule S19, see Figure 2.1).

2.2   The Cartesian coordinates file

Before executing `Q2DTor` for a new system the user also has to provide the Cartesian coordinates of an initial geometry, as in an standard xyz file. For instance, for S19 the starting geometry written in the `S19.xyz` file is:

```
 16
reference geometry for S19
C    -0.44444    +0.24701    +0.13754
C    +0.46516    +1.28342    -0.01406
C    +1.82115    +1.02336    -0.12141
C    +2.28065    -0.28193    -0.08887
C    +1.37751    -1.32311    +0.05344
C    +0.02381    -1.06026    +0.16805
H    +0.10950    +2.30369    -0.05152
H    +2.51721    +1.83975    -0.23979
H    +3.33624    -0.48770    -0.17833
H    +1.72959    -2.34325    +0.07756
```
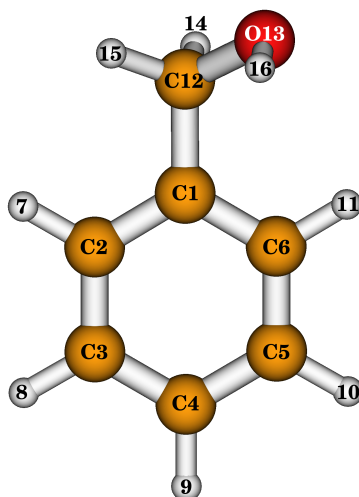
Figure 2.1: Labelling of the benzyl alcohol molecule. Carbon (orange), hydrogen (grey), and oxygen (red).

```
H      -0.68458     -1.86738     +0.28042
C      -1.91162     +0.52997     +0.29023
O      -2.73031     -0.43978     -0.30627
H      -2.18446     +0.52613     +1.34367
H      -2.13344     +1.52798     -0.09450
H      -2.52089     -0.49170     -1.23682
```

2.2.1   Isotopic substitution

A different isotope than the most abundant chemical element can be defined in the xyz file by adding the isotopic mass (in amu) after the Cartesian coordinates of the atom. For instance, the last hydrogen atom of the previous example can be turned into a deuterium atom by modifying the last line to:

```
H      -2.52089     -0.49170     -1.23682    2.01410178
```

2.3   Description of the main input file

The input file is split into different sections, each of them containing different keywords. Some general aspects related to the input file are as follows:

- A given section (scname) is initialized with start_scname. The end of the section is indicated with end_scname.

- The keywords always include recommended values and, in general, the user needs only to modify a few of them, usually just the symmetry of the torsional PES and the values between curly brackets. **The brackets should be removed** after the values have been modified.

- Comments can be added in the input file with the hashtag symbol (#).

- The xyz and input file names have to match; thus, if the input file is `S19.inp`, the reference Cartesian coordinates geometry should be provided in a file called `S19.xyz`.

- The specification of an output file is not needed. `Q2DTor` automatically generates one after the initialization step (`S19.out` in the example) and the information associated with each run is appended to this file.

The following subsections explain the keywords of the different parts of the input file.

2.3.1 `torsions` **section**

In this section the user defines the two coupled torsions to be studied. Notice that the values associated with these keywords are between curly brackets, as an indication that **they must be checked** by the user.

```
#--------------------------#
# Torsional information     #
#--------------------------#
start_torsions              #
  torsion1        {1-2-3-4} #
  torsion2        {2-3-4-5} #
  tsigma1         {1}       #
  tsigma2         {1}       #
end_torsions                #
#--------------------------#
```

Keywords `torsion1` and `torsion2` are used to define the atoms involved in the two torsions (ϕ_1 and ϕ_2), and keywords `sigma1` and `sigma2` specify their torsional symmetry numbers. Notice that the numbering of atoms starts at index 1 (and not at index 0, as might be anticipated by *Orca* users)

For S19 (Figure 2.1), the two torsions involve the atoms 13-12-1-2 and 16-13-12-1, respectively. The first torsion has a torsional symmetry number of two because a 180° internal rotation of the phenyl group is indistinguishable from the phenyl structure before rotation. Thus, for the test S19 we have:

```
#--------------------------#
# Torsional information     #
#--------------------------#
start_torsions              #
  torsion1       13-12-1-2  #
  torsion2       16-13-12-1 #
  tsigma1        2          #
  tsigma2        1          #
end_torsions                #
#--------------------------#
```

Notice that in torsions ϕ_1 and ϕ_2 the central atoms have to be 12-1 and 13-12, respectively, because they define the torsional bonds. However, the terminal atoms

could be others from the ones specified here. Thus, it is also correct to specify ϕ_1 by the atoms 13-12-1-6 and ϕ_2 by the atoms 16-13-12-14 or 16-13-12-15, although we recommend to specify heavy atoms as the terminal atoms whenever possible.

2.3.2 `calcs` **section**

This section collects some information regarding the electronic structure calculations. The level of calculation can be defined with the keyword `level` and the charge and the spin multiplicity of the system are introduced with the `charge` and `multiplicity` keywords, respectively.

```
#---------------------------#
# Calculations              #
#---------------------------#
start_calcs                  #
  level          {hf sto-3g} #
  charge              0       #
  multiplicity        1       #
end_calcs                    #
#---------------------------#
```

For the tests we have used HF/3-21G. The calculation was performed with *Gaussian 09*, and therefore:

```
  level           hf/3-21G
```

2.3.3 `pes` **section**

This section gives some specifications about how to build the 2D-PES.

```
#---------------------------#
# Torsional PES             #
#---------------------------#
start_pes                    #
  t1step          10.0        #
  t2step          10.0        #
  symmetry        none        #
end_pes                      #
#---------------------------#
```

The `t1step` and `t2step` keywords define the step (in degrees) for each of the two torsions ϕ_1 and ϕ_2, respectively, when the scan is performed. Notice that with the default values (every 10 degrees), a total of 1 296 points are calculated. Each of the points of the 2D-PES involves optimization of all the degrees of freedom except the two torsions. However, if some of the geometries of the torsional 2D-PES present molecular point group symmetry and/or symmetry under internal rotation of the two tops, this number is drastically reduced.

The symmetry conditions can be addressed by the following keywords: 'a' indicates that internal rotation of the molecule leads to a C_2 axis or/and a plane that allows the two rotors to exchange places; 'b' indicates that internal rotation of

the molecule involves a C_s plane that contains the two torsional bonds (the single bonds that twist in the rotation of the tops); and 'c' indicates that the total torsional symmetry number is greater than 1. Consequently, the symmetry keyword can be set to a, b, c, ab, ac, bc, abc or none. The default is none, i.e., no symmetry.

The example molecule fulfills both b and c conditions, which can be indicated by:

```
symmetry            bc
```

As a consequence of this symmetry, the number of points to be calculated for benzyl alcohol is reduced to 351.

2.3.4 `fourier` **section**

```
#--------------------------#
# Fitting details          #
#--------------------------#
start_fourier             #
  weight          0.9     #
  ignore          0.0     #
  # Fourier Terms (Even)   #
  cos1            1-6     #
  cos2            1-6     #
  cos1cos2        1-6 , 1-6 #
  sin1sin2        1-6 , 1-6 #
  # Fourier Terms (Odd)    #
  sin1            none    #
  sin2            none    #
  cos1sin2        none    #
  sin1cos2        none    #
end_fourier               #
#--------------------------#
```

Q2DTor fits the 2D-PES to the sum of truncated Fourier series:

$$
\begin{aligned}
V(\phi_1, \phi_2) \simeq V_0 &+ \sum_j a_j \cos(j\phi_1) + \sum_k b_k \cos(k\phi_2) + \\
&+ \sum_j c_j \sin(j\phi_1) + \sum_k d_k \sin(k\phi_2) + \\
&+ \sum_{jk} e_{jk}^{(cc)} \cos(j\phi_1)\cos(k\phi_2) + \sum_{jk} e_{jk}^{(cs)} \cos(j\phi_1)\sin(k\phi_2) + \\
&+ \sum_{jk} e_{jk}^{(sc)} \sin(j\phi_1)\cos(k\phi_2) + \sum_{jk} e_{jk}^{(ss)} \sin(j\phi_1)\sin(k\phi_2)
\end{aligned}
$$

In order to define the 1D-series, *i.e.* those with terms $\cos(j\phi_1)$, $\cos(k\phi_2)$, $\sin(j\phi_1)$, and $\sin(k\phi_2)$, the keywords cos1, cos2, sin1 and sin2 keywords should be used. The line

```
cos1    1-6
```

which is equivalent to

```
cos1     1 2 3 4 5 6
```

indicates that the terms $\cos(\phi_1)$ to $\cos(6\phi_1)$ are part of the Fourier series. The line

```
sin1      none
```

indicates that the $\sum \sin(j\phi_1)$ series is not needed.

The two-dimensional terms are defined in a similar fashion. For example, to define the series with $\cos(j\phi_1)\cos(k\phi_2)$ terms, the `cos1cos2` keyword is used. A line like:

```
cos1cos2      1-6 , 1-3
```

indicates that the fitting considers the $\cos(j\phi_1)\cos(k\phi_2)$ terms with the pairs $(j,k) = (1,1),(1,2),(1,3),(2,1)\ldots(6,1),(6,2),$ and $(6,3)$.

To select the terms, the user should take into account that:

- The odd terms (keywords `sin1`, `sin2`, `sin2cos1` and `sin1cos2`) should not be included when the 2D-PES fulfills symmetry condition (b).

- If the torsional symmetry number of one rotor is larger than one, then only multiples of that symmetry number should be included in the terms involving that torsion.

This section also contains two more keywords. On the one hand, `weight` can be used to perform a weighted fitting, which is useful to better fit the low-energy PES points. An unweighted fitting can be performed by setting this keyword to 0.0, but we recommend to use the default value (i.e. 0.9). In addition, the `ignore` keyword can be used to remove those Fourier terms whose coefficient absolute value is smaller than the defined value (in cm^{-1}).

The Fourier terms defined for fitting the 2D-PES are also used in the fitting of the kinetic energy terms.

2.3.5 `statpoint` **section**

```
#---------------------------#
# Searches for stat. points  #
#---------------------------#
start_statpoint               #
  tolerance        2.0        #
  freqscal         1.000      #
end_statpoint                 #
#---------------------------#
```

Once the 2D-PES is represented as a sum of Fourier terms, the location of stationary points can be carried out. The `tolerance` keyword defines the searching step (in degrees) and a value of two degrees (the default) is recommended. The `freqscal` defines a scaling factor for the harmonic normal-mode frequencies. [8] When one uses a scaling factor other than one, all calculated vibrational frequencies are mutiplied by the scaling factor to account for anharmonicity and systematic errors of the model chemisty; when a non-unity scaling factor is used, the

harmonic approximation should be called the quasiharmonic approximation.

2.3.6 `tor2dns` section

```
#--------------------------#
# 2D-NS Hamiltonian        #
#--------------------------#
start_tor2dns             #
  dijvar          yes     #
  kmax            100     #
  maxeigen        1e4     #
end_tor2dns               #
#--------------------------#
```

This section collects the keywords associated with the use of the two-dimensional non-separable (2D-NS) Hamiltonian by the variational method and with the parameters of the variational calculation. [9, 10] The first keyword is related to the kinetic energy operator. The user can either include the full coupling in the kinetic energy using (`dijvar yes`) or neglect the kinetic energy coupling (`dijvar no`). For the latter, Q2DTor assumes that the reduced moments of inertia do not change with the torsions and that they are given by their values at the global minimum.

Keyword `kmax` specifies the value of K in the trial variational function, which is given by:

$$\Phi(\phi_1, \phi_2) = \frac{1}{2\pi} \sum_{k_1=-K}^{K} \sum_{k_2=-K}^{K} c_{k_1,k_2} e^{ik_1\phi_2} e^{ik_2\phi_2}$$

The `maxeigen` keyword is needed in the diagonalization process of the 2D-NS Hamiltonian. This task is carried out by the `eigsh` function of the `scipy` library. Additionally, as most of the Hamiltonian matrix elements are zero, the Python classes `csc_matrix` and `csr_matrix` of the `scipy` library for sparse matrices are used. During the diagonalization process, `eigsh` generates a limited number of eigenvalues each time it is called, and therefore it works in a loop until the largest eigenvalue specified by `maxeigen` (in cm^{-1}) is reached. The recommended value is the default one, i.e., $10\,000$ cm^{-1}.

2.3.7 `rovibpf` section

```
#--------------------------#
# Partition functions      #
#--------------------------#
start_rovibpf             #
  interpolation    fourier  #
  integrationstep  1.0    #
end_rovibpf               #
#--------------------------#
```

The E2DT rovibrational partition function involves the evaluation of a double integral, which is calculated numerically using the trapezoidal rule. The keywords

of this section allow the user to specify: the integration step (in degrees) with
`intergrationstep` and the interpolation method `interpolation` used to eval-
uate the determinants of the **D** and **S** matrices at points at which the geometries
are unavailable. The default for the latter is to use Fourier interpolation, although
the use of splines is also possible. For example, the use of cubic splines can be
indicated by:

```
interpolation      3
```

2.3.8 `temperatures` **section**

This is the last section in the input file. The working temperatures (in K) are
specified here.

```
#---------------------------#
# Working temperatures      #
#---------------------------#
start_temperatures         #
  100.0 300.0 500.0        #
  700.0 900.0              #
end_temperatures           #
#---------------------------#
```

`Q2DTor` **in-line arguments**

In this chapter we describe the steps (see Figure 3.1) needed to calculate the partition functions and thermodynamic functions. Each step can be carried out by executing `Q2DTor` with a specific argument. The list of the available arguments can be obtained by executing:

```
python q2dtor.py --help
```
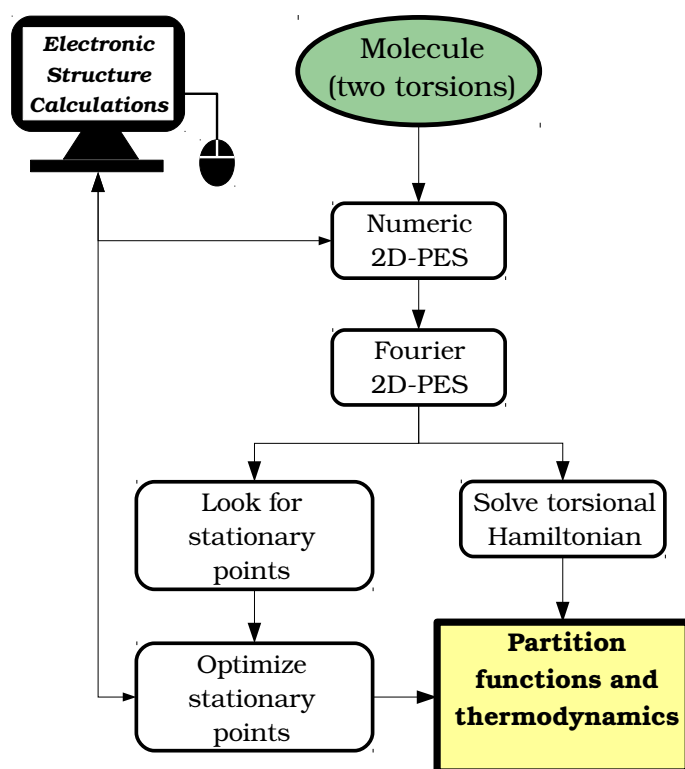
or just

```
python q2dtor.py -h
```


Figure 3.1: Schematic flux diagram of `Q2DTor`.

The results commented in this chapter correspond to the example system S19 (i.e., benzyl alcohol). The input file for this molecule is:

```
#--------------------------#        sin1              none       #
# Torsional information    #        sin2              none       #
#--------------------------#        cos1sin2          none       #
start_torsions                      sin1cos2          none       #
  torsion1     13-12-1-2    #     end_fourier
  torsion2     16-13-12-1   #     #--------------------------#
  tsigma1      2            #     # Searches for stat. points #
  tsigma2      1            #     #--------------------------#
end_torsions                      start_statpoint
#--------------------------#        tolerance        1.0         #
# Calculations             #        freqscal         1.000       #
#--------------------------#      end_statpoint
start_calcs                       #--------------------------#
  level         hf 3-21G    #     # 2D-NS Hamiltonian         #
  charge        0           #     #--------------------------#
  multiplicity  1           #     start_tor2dns
end_calcs                           dijvar           yes         #
#--------------------------#        kmax             100         #
# Torsional PES            #        maxeigen         1e4         #
#--------------------------#      end_tor2dns
start_pes                         #--------------------------#
  t1step          10.0      #     # Partition functions       #
  t2step          10.0      #     #--------------------------#
  symmetry        bc        #     start_rovibpf
end_pes                             interpolation   fourier     #
#--------------------------#        integrationstep 1.0         #
# Fitting details          #     end_rovibpf
#--------------------------#      #--------------------------#
start_fourier                     # Working temperatures      #
  weight          0.9       #     #--------------------------#
  ignore          0.0       #     start_temperatures          #
  # Fourier Terms (Even)    #        100.0   150.0   200.0     #
  cos1      2 4 6 8         #        250.0   300.0   400.0     #
  cos2      1-9             #        500.0   700.0  1000.0     #
  cos1cos2  2 4 6  , 1-7    #       1500.0  2000.0  2500.0     #
  sin1sin2  2 4 6  , 1-7    #     end_temperatures            #
  # Fourier Terms (Odd)     #     #--------------------------#
```

After the user has created the input files (`inp` and `xyz` files), `Q2DTor` has to be executed sequentially, and in each run it generates output files that are read when executing the program with the subsequent argument in the list. Figure 3.2 indicates the order of the arguments.

Before entering into details, we emphasize that some tasks may require quite a long time. For those steps, a useful option is to execute `Q2DTor` in the background, using for instance the `nohup` command. Thus, for a given argument `arg`, this implies:

```
nohup python q2dtor.py S19 --arg &
```

instead of:
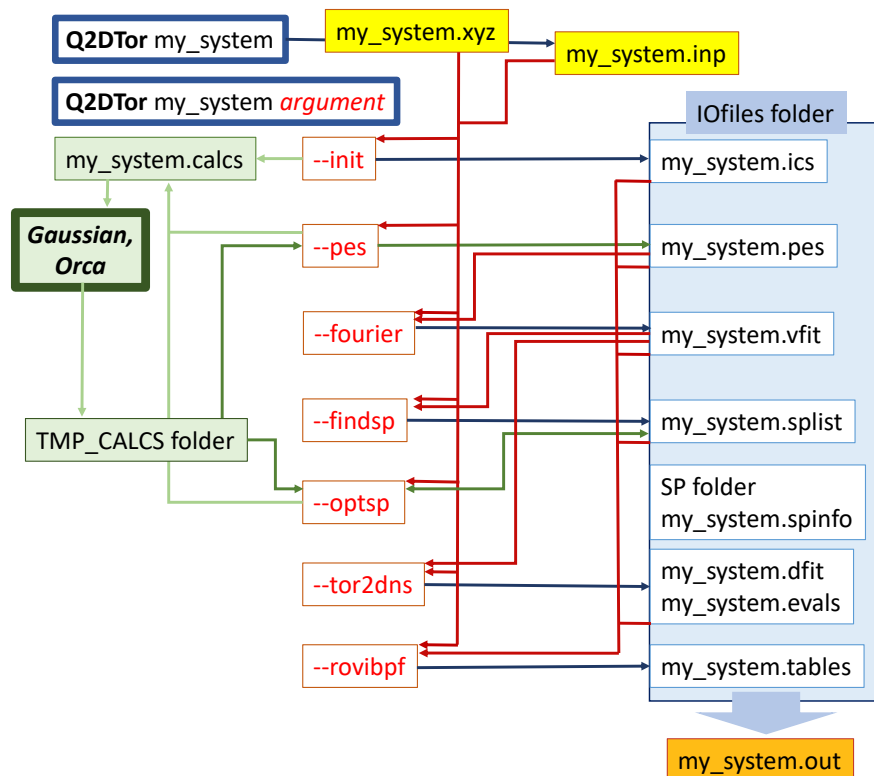
```
python q2dtor.py S19 --arg
```

Figure 3.2: Flow diagram indicating the available task arguments and the files used or created by Q2DTor. Files provided by the user are in yellow boxes; files read by the program are pointed to by red arrows; files related to electronic structure calculations are pointed to by green arrows; the main output file is an orange box; and other output files are pointed to by blue arrows.

In general, the executions that we recommend to use with `nohup` are the ones with `pes`, `optsp`, `tor2dns` and `rovibpf` arguments.

3.1 `--init`: initialization

At this initial stage the user indicates the electronic structure program to be used. The two options are `gaussian`

```
python q2dtor.py S19 --init gaussian
```

and `orca`.

```
python q2dtor.py S19 --init orca
```

Q2DTor also reads the `xyz` file and creates the `IOfiles` folder and the `S19.ics` and `S19.calcs` files. The `S19.ics` file contains the connectivity of the system, as well as a set of redundant internal coordinates:

```
start_connectivity
  1-2        1-6        1-12       2-3        2-7
  3-4        3-8        4-5        4-9        5-6
  5-10       6-11       12-13      12-14      12-15
  13-16
end_connectivity

start_ics
 #--------
 # Stretches (16):
 #--------
 1-2            1-6            1-12           2-3            2-7
 3-4            3-8            4-5            4-9            5-6
 5-10           6-11           12-13          12-14          12-15
 13-16
 #--------
 # Bent bond angles (18):
 #--------
 1-2-7          1-6-11         1-12-14        1-12-15        2-1-12
 2-3-8          3-2-7          3-4-9          4-3-8          4-5-10
 5-4-9          5-6-11         6-1-12         6-5-10         12-13-16
 13-12-14    13-12-15    14-12-15
 #--------
 # Linear bond angles (2x0):
 #--------
 #--------
 # Improper torsions & Torsions (15):
 #--------
 1-2-3-8    1-3-7-2    1-5-11-6   1-12-13-16   1-13-15-12
 2-1-6-11   2-1-12-13  2-3-4-9    2-4-8-3      2-6-12-1
 3-2-1-12   3-5-9-4    4-6-10-5   9-4-5-10     10-5-6-11
end_ics
```

The `connectivity` section contains the pairs of atoms that are considered to be bonded. This information is used to generate the geometries that build the 2D-PES. The internal coordinates are defined in the `ics` section. Notice that the '-' symbol is used to separate the label of each atom involved, with the exception of linear bond angles that uses '='. Thus, for example, a linear bond angle involving atoms 1, 2, and 3 (with 2 being the central atom) would be defined by 1=2=3. Although these internal coordinates are automatically generated, the user can modify this file to define an alternative set of coordinates.

The `S19.calcs` file contains four sections:

- The `scangeom` section includes the input file that is read by the ESSO. `Q2DTor` calls the ESSO program as many times as needed to scan the whole torsional PES. The ESSO partially optimizes each geometry, i.e, optimizes all the degrees of freeedom less the two torsions.

```
#----------------------------#
start_scangeom gaussian
%nproc=2
%mem=2GB
%chk=[Q2DTor_name].chk
```

```
#p hf/3-21G
   scf=tight NoSymmetry [Q2DTor_MOread]
   opt=(tight,modredundant)

Scan and partial geometry opt. with 2 frozen torsions

0 1
[Q2DTor_geometry]

2 1 12 13  F
1 12 13 16  F

end_scangeom
#---------------------------#
```

The sections below are used by the ESSO to locate stationary points, once that the construction of the torsional PES is completed.

- The sp0 section contains the reference input file for the optimization of the minima.

```
#---------------------------#
start_sp0 gaussian
%nproc=2
%mem=2GB
%chk=[Q2DTor_name].chk
#p hf/3-21G
   scf=verytight
   opt=tight

Optimization of a minimum

0 1
[Q2DTor_geometry]

--Link1--
%nproc=2
%mem=2GB
%chk=[Q2DTor_name].chk
#p hf/3-21G
   scf=verytight freq=noraman geom=allcheck

end_sp0
#---------------------------#
```

- The sp1 section contains the reference input file for the optimization of first order saddle points. These stationary points correspond to transition states associated with the internal rotation about ϕ_1 or ϕ_2.

```
#---------------------------#
start_sp1 gaussian
%nproc=2
%mem=2GB
%chk=[Q2DTor_name].chk
#p hf/3-21G
   scf=verytight
```

```
    opt =( tight ,ts , calcfc , noeigentest )

Optimization  of  a  first  order  saddle  point

0 1
[Q2DTor_geometry]

-- Link1 --
%nproc =2
%mem =2 GB
%chk =[Q2DTor_name].chk
#p hf/3-21G
    scf = verytight  freq = noraman  geom = allcheck

end_sp1
#----------------------------#
```

- The `sp2` section contains the reference input file for the optimization of second order saddle points, i.e., structures which are a maximum with respect to the two dihedral angles and a minimum with respect to all the other degrees of freedom. Therefore, these structures will be maxima in the 2D-PES.

```
#----------------------------#
start_sp2 gaussian
%nproc =2
%mem =2 GB
%chk =[Q2DTor_name].chk
#p hf/3-21G
    scf = verytight
    opt =( tight , saddle =2 , calcfc , noeigentest )

Optimization  of  a  second  order  saddle  point

0 1
[Q2DTor_geometry]

-- Link1 --
%nproc =2
%mem =2 GB
%chk =[Q2DTor_name].chk
#p hf /3 -21 G
    scf = verytight  freq = noraman  geom = allcheck

end_sp2
#----------------------------#
```

Notice that the information obtained from the `S19.inp` file is colored in red. Additionally, the commands in brackets (in blue) are not *Gaussian* commands, but are indicators to the `Q2DTor` program to perform different actions when the ESSO is working on a given geometry:

[**Q2DTor_name**]: is substituted by the name of the file generated by `Q2DTor`.

[**Q2DTor_geometry**]: is substituted by the Cartesian coordinates of the structure.

[**Q2DTor_MOread**]: copies the chk file of the previous calculation (if it exists) and adds the `guess=read` command in the case of *Gaussian* or `MORead` in the case of *Orca*. This action can speed up the calculation substantially.

Finally, we note that additional commands can be introduced in the command lines in the same way as for the *Gaussian* or *Orca* input files. For example, in this case the line with the level of calculation can be modified to add additional *Gaussian* commands. For instance, we can add more cycles to the SCF procedure (magenta color):

```
scf=(verytight,maxcycle=200)
```

3.2 `--pes`: getting the numeric 2D-PES

The construction of the 2D-PES is carried out by typing:

```
python q2dtor.py S19 --pes
```

The ESSO input and output files will be stored inside `TMP_CALCS/`, so the user can check these files if something goes wrong. At the end of the calculation Q2DTor makes a summary of the ESSO output and all the information needed for a restart calculation is stored in the `S19.pes` file. The format is the following:

```
16
Geometry    -342.66210965       0.000     0.000  S19_000_000   YES
C        -0.415300      +0.281062       +0.141041
C        +0.514020      +1.293214       -0.002960
C        +1.865267      +1.001512       -0.122217
C        +2.300468      -0.307385       -0.098502
C        +1.374685      -1.329641       +0.045684
C        +0.032399      -1.035301       +0.163904
H        +0.217037      +2.322587       -0.026092
H        +2.571022      +1.800803       -0.233146
H        +3.344032      -0.532627       -0.190597
H        +1.700839      -2.350617       +0.065500
H        -0.678877      -1.832158       +0.275256
C        -1.915863      +0.519021       +0.277847
O        -2.362814      +1.885142       +0.252383
H        -2.416436      -0.005593       -0.526591
H        -2.243676      +0.083387       +1.213618
H        -1.645649      +2.515366       +0.148968
16
Geometry    -342.66252611       0.000    10.000  S19_000_010   YES
C        -0.415292      +0.279969       +0.151264
C        +0.512725      +1.293148       +0.005793
C        +1.863341      +1.002340       -0.121897
C        +2.299028      -0.306553       -0.104581
C        +1.374593      -1.329609       +0.042724
C        +0.032603      -1.036328       +0.167445
H        +0.215279      +2.322593       +0.002865
H        +2.568500      +1.802251       -0.232200
H        +3.342150      -0.531178       -0.203062
H        +1.701419      -2.350448       +0.058344
```

```
 H          -0.678118        -1.833911         +0.277170
 C          -1.916429        +0.518962         +0.276083
 O          -2.359125        +1.886059         +0.236601
 H          -2.409239        -0.023555         -0.522332
 H          -2.254165        +0.104989         +1.217220
 H          -1.663922        +2.498133         -0.018238
 ...
```

This file can be visualized with the Molden [11] program, and it contains the
energy (in hartrees), the dihedral angle associated with each targeted torsion, the
name of the electronic structure file, and the Cartesian coordinates of every of the
calculated geometries by the ESSO.

3.2.1   How to proceed if an electronic structure calculation fails?

If the electronic structure calculation of one of the points of the PES fails, the
program does not store the geometry. Instead, it prints an 'XX' with the word
FAILED in the line reserved for comments:

```
 1
 FAILED     0.0000000    S19_000_010
 XX         0.000000        0.000000          0.000000
```

In such a case, the user can execute again the program to recalculate that point.
The converged geometries are stored in the .pes file and only the points which
have failed are recalculated. If preferred, the user can introduce directly the infor-
mation about a given point in the .pes file. Before proceeding to the next step, all
the required points to build the torsional PES should have 'YES' in the comment
line. This guarantees that the geometries were properly optimized.

3.3   --fourier: getting the Fourier 2D-PES

The fitting to the Fourier series potential is carried out by executing:

```
python q2dtor.py S19 --fourier
```

The fitting coefficients are stored in the S19.vfit file, which also contains infor-
mation about the quality of the fit:

```
#-------------------------------------------#
# fitting correlation: (1.0 - r^2) = 1.6e-04   #
# average abs. errors:                      #
#    3.8e+00                                 #
#    1.5e+00 (for points below mean value)    #
# elapsed time:  34.0 seconds                #
#-------------------------------------------#
const       -      -     +1248.9444
cos         02     -      -376.1206
cos         04     -       +34.6231
cos         06     -       +13.6779
cos         08     -        -2.7950
```

```
cos            -      01        -73.3360
cos            -      02       +159.3618
cos            -      03       +256.3117
cos            -      04         -7.2031
cos            -      05         -1.7663
cos            -      06         +1.0919
cos            -      07         +0.0047
cos            -      08         +0.2710
cos            -      09         -0.1369
coscos        02      01       +492.7750
coscos        04      01        +51.9601
coscos        06      01         -0.2149
coscos        02      02       +303.9126
coscos        04      02       +100.0706
coscos        06      02         +3.6327
coscos        02      03        +38.1390
coscos        04      03        +32.0073
coscos        06      03         +9.6760
...
```

In general, the fitting can be considered satisfactory when the average absolute error is smaller than 10 cm^{-1}.

3.4 `--findsp`: finding the stationary points

The search for stationary points across the 2D Fourier series potential is accomplish by typing:

```
python q2dtor.py S19 --findsp
```

The stationary points that are successfully located are stored in the `S19.splist` file:

```
#   Type      Phi1      Phi2        Energy        opt OK?        SP name
# ------ | ------ | ------ | ------------ | ---------- | --------------
     0        25.08     65.82        -17.22          NO         S19_025_066
     0         0.00    180.00       +697.38          NO         S19_000_180
     1         3.00    223.00       +805.31          NO         S19_003_223
     1        90.00      0.00      +1303.25          NO         S19_090_000
     1        90.00    180.00      +1862.54          NO         S19_090_180
     2        89.85    128.44      +2203.17          NO         S19_090_128
     2         0.00      0.00      +2315.82          NO         S19_000_000

# Number of stationary points: 7
```

At this stage, we recommend to visualize the position of those stationary points in the 2D-PES. A pdf file (`S19.pdf`) containing the plot with the PES and the location of each stationary point can be generated with the `--pdf` tool (see Chapter 4), i.e., by executing:

```
python q2dtor.py S19 --pdf
```

3.4.1   How to proceed when a stationary point is missing?

In our case, we observe that the loose saddle point about $(\phi_1, \phi_2) = (130, 67)$ was not found (see Figure 3.3). To include it, the user only has to modify the S19.splist, adding the following line:

```
    1        130.00    67.00              -              NO            S19_130_067
```

In the Type column, 0 is added for points that are minima in the 2D-PES, 1 for saddle points and 2 for maxima. Notice that --findsp locates the stationary points in the 2D Fourier series potential, and in general the values of the dihedral angles ϕ_1 and ϕ_2 will be very close to those obtained by optimization of the whole structure. However, we still need to calculate by the ESSO the geometry of the stationary points, together with the Hessian, in order to be able to evaluate the vibrational partition functions.

3.5   --optsp: optimizing the stationary points

The optimization of the stationary points listed in S19.splist is carried out by:

```
python q2dtor.py S19 --optsp
```

The program reads the sp0, sp1 and sp2 section of the S19.calcs file and it launches the ESSO, which is now responsable for the optimization of the stationary points. For each stationary point, Q2DTor generates two files: a file containing the geometry and the Hessian matrix (with the gts extension) and a file (with the molden extension) for the Molden program, so the user can visualize the normal mode vibrations.

Once all the stationary points are optimized (see Figure 3.4), Q2DTor updates the S19.splist file:

```
#  Type      Phi1      Phi2          Energy       opt OK?        SP name
# ------  |  ------  |  ------  |  ------------  |  ----------  |  --------------
    0        25.09     65.84         +0.00           YES          S19_025_066
    0         0.00    180.00       +713.29           YES          S19_000_180
    1         2.66    222.53       +821.27           YES          S19_003_223
    1        88.19      0.00      +1321.90           YES          S19_090_000
    1       130.43     66.81      +1584.83           YES          S19_130_067
    1        88.63    180.00      +1871.24           YES          S19_090_180
    2        89.93    128.31      +2225.03           YES          S19_090_128
    2         0.00      0.00      +2331.92           YES          S19_000_000

# Number of stationary points: 8
```

The resulting stationary points are listed in the S19.splist file as indicated above. As previously, Type 0 corresponds to minima, Type 1 to first order saddle points (one imaginary frequency associated with one of the torsions), and Type 2 to second order saddle points (maxima in the 2D-PES and with two imaginary frequencies corresponding to the motion of the two torsions). All of the stationary points have been sucessfully optimized because all the structures associated with the 'opt OK?' column are 'YES'. In some cases Q2DTor may complain about a given stationary point with a warning message. This may happen when:

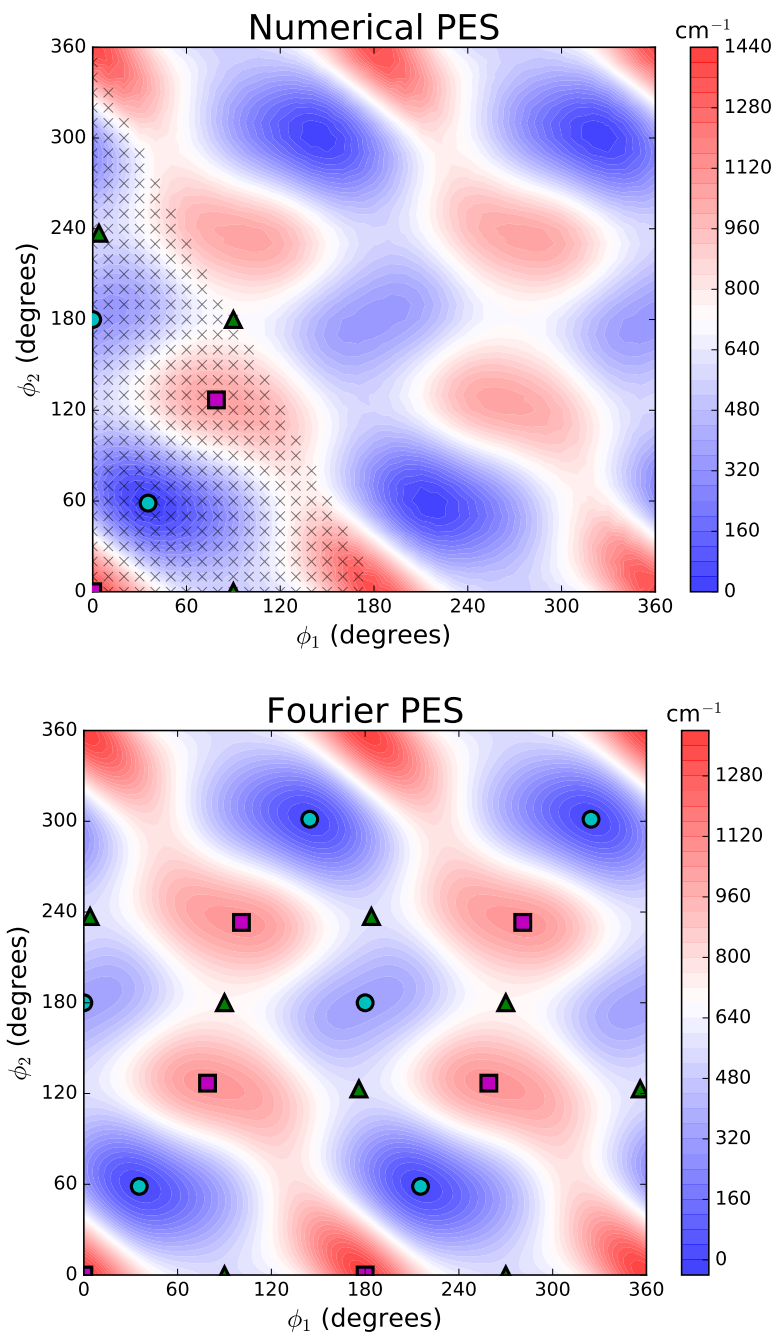Figure 3.3: Contour-line representation of the numerical 2D-PES for S19 and the position of the stationary points obtained with the --findsp option. Cyan circles (⊙) are minima; green triangles (▲) are saddle points; and pink squares (▪) are maxima. In the upper figure, each **x** symbol represents a calculated point. In the lower one, redundant stationary points were generated by applying the symmetry conditions.
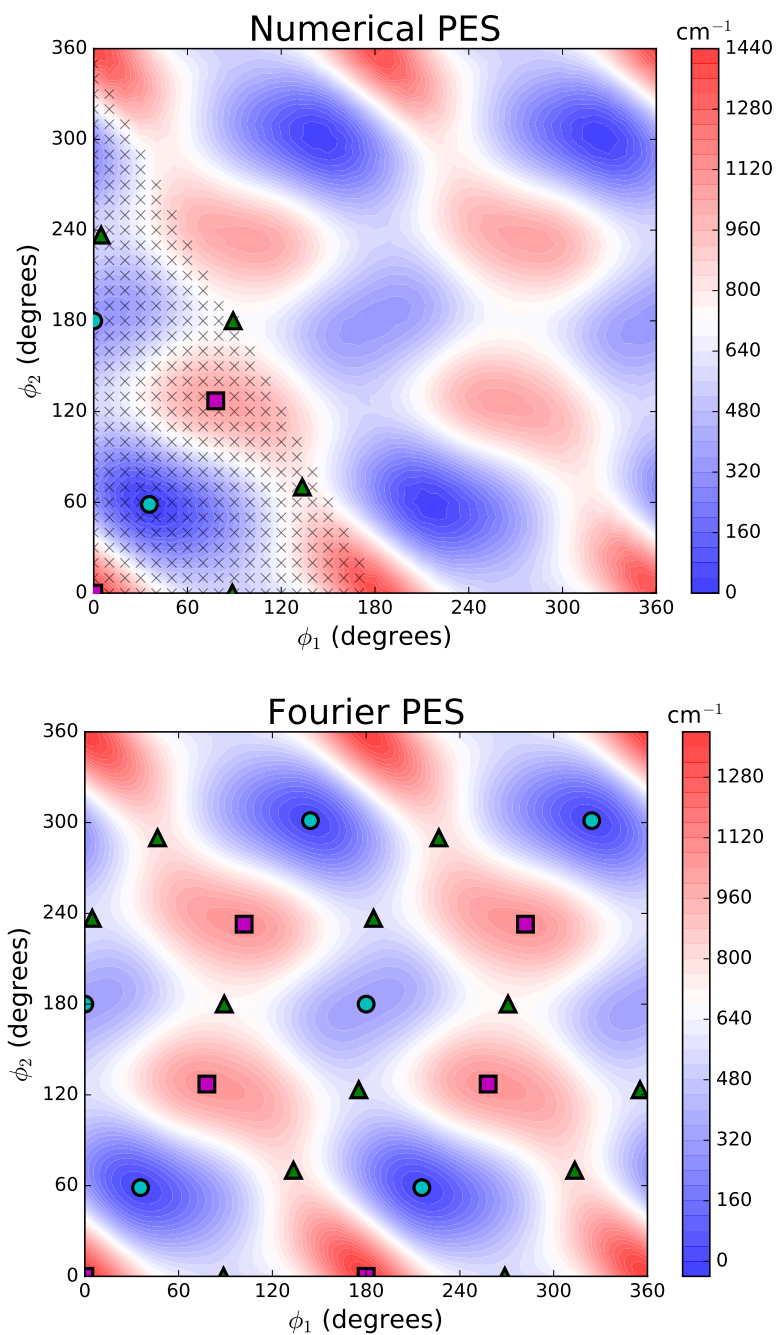
Figure 3.4: Same as Figure 3.3, but after executing `Q2DTor` with the --optsp option.

- There is a problem in the optimization of the stationary point

- The number of imaginary frequencies does not correspond to the specified type in the S19.splist file.

- The dihedral angles differ significantly from those obtained in the search.

In any of these situations, the stationary point appears with a NO in the "opt OK?" column of the S19.splist file.

The program also refines the set of internal coordinates, reducing it to a set of non-redundant coordinates. Therefore, at this stage the S19.ics file is updated with the new set and the initial set is stored in S19.ics_original. After the optimization step, Q2DTor prints the following information about each stationary point:

- The energy relative to the global minimum;

- The (scaled) Cartesian normal-mode frequencies and the zero-point energy;

- The **D** matrix, its inverse, the reduced moments of inertia, and the coupling between them.

- The (scaled) projected non-torsional normal-mode frequencies and the zero-point energy.

3.6   --tor2dns: getting the 2D-NS energy levels

In this section the torsional two-dimensional non-separable (2D-NS) partition functions [9, 10] are calculated. To build and diagonalize the 2D-NS Hamiltonian matrix, the user should type:

```
python q2dtor.py S19 --tor2dns
```

The corresponding eigenvalues are stored in the S19.evals file:

```
kmax    100
dijvar  yes
    0   241.97006
    1   241.97006
    2   241.97006
    3   241.97006
    4   329.04317
    5   329.04317
    6   329.04318
    7   329.04318
        ...
```

At this stage Q2DTor also generates the S19.dfit file, which contains the fitting to Fourier series of the d_{ij} elements. The program prints the 2D-NS torsional partition functions in the output file.

3.7 `--rovibpf`: partition functions and thermodynamic functions

Finally, the calculation of the partition functions is accomplish by typing:

```
python q2dtor.py S19 --rovibpf
```

In addition to the output file, Q2DTor also saves the partition functions in the S19.tables file. There, the user can find a list of the partition functions at different temperatures wih the zero of energy at the lowest zero point level of the torsional PES:

```
Energy of the lowest zero point level of the torsional PES:
      * 1WHO =>   90.248 kcal/mol
      * MSHO =>   90.248 kcal/mol
      * 2DNS =>    0.692 kcal/mol
      * EHR  =>   89.537 kcal/mol
      * E2DT =>   90.229 kcal/mol

(a) Rovibrational partition functions using as zero of energy
    the lowest zero point level of the torsional PES

           T (K) |  rv(1WHO) |  rv(MSHO) |  rv(E2DT) | E2DT/MSHO
          -------------------------------------------------------
          100.00 |  8.504E+04 |  1.701E+05 |  1.758E+05 |  1.03342
          150.00 |  2.401E+05 |  4.822E+05 |  5.062E+05 |  1.04979
          200.00 |  6.095E+05 |  1.240E+06 |  1.336E+06 |  1.07741
          ...
         2000.00 |  3.716E+18 |  1.585E+19 |  1.671E+19 |  1.05443
         2500.00 |  1.789E+21 |  8.053E+21 |  7.567E+21 |  0.93971

    Components of E2DT

           T (K) |    2DNS    |  TorsClas  |    EHR
          -------------------------------------------------
          100.00 |  2.830E+00 |  2.690E-01 |  1.671E+04
          150.00 |  3.706E+00 |  6.202E-01 |  8.472E+04
          200.00 |  4.830E+00 |  1.141E+00 |  3.156E+05
          ...
         2000.00 |  2.403E+02 |  2.019E+02 |  1.404E+19
         2500.00 |  3.385E+02 |  2.945E+02 |  6.584E+21
```

Additionally, it also lists the the rovibrational partition functions calculated from the bottom of the torsional PES,

```
(b) Rovibrational partition functions using as zero of energy
    the bottom of the torsional PES

           T (K) |  rv(1WHO) |  rv(MSHO) |  rv(E2DT) | E2DT/MSHO
          -------------------------------------------------------
          100.00 | 4.975E-193 | 9.953E-193 | 1.135E-192 |  1.14018
          150.00 | 7.797E-127 | 1.566E-126 | 1.755E-126 |  1.12090
          200.00 |  1.474E-93 |  2.999E-93 |  3.394E-93 |  1.13170
          ...
         2000.00 |  5.111E+08 |  2.179E+09 |  2.309E+09 |  1.05963
         2500.00 |  2.308E+13 |  1.039E+14 |  9.802E+13 |  0.94341


    Components of E2DT

           T (K) |    2DNS    |  TorsClas  |    EHR     | Fq(2DNS)
          -------------------------------------------------------------
          100.00 |  8.705E-02 |  2.690E-01 | 3.507E-192 |  0.32359
          150.00 |  3.638E-01 |  6.202E-01 | 2.991E-126 |  0.58660
          200.00 |  8.472E-01 |  1.141E+00 |  4.572E-93 |  0.74231
```

```
             ...
     2000.00 |  2.019E+02 |  2.019E+02 |  2.309E+09 | 1.00000
     2500.00 |  2.945E+02 |  2.945E+02 |  9.802E+13 | 1.00000
```

as well as the translational, electronic and total partition functions:

```
Translational and electronic partition functions

          T (K) |   Qtrans   |   Qelec
        -----------------------------------
         100.00 |  2.915E+06 |  1.000E+00
         150.00 |  8.031E+06 |  1.000E+00
         200.00 |  1.649E+07 |  1.000E+00
         ...
        2000.00 |  5.214E+09 |  1.000E+00
        2500.00 |  9.108E+09 |  1.000E+00


Total partition functions, from rovibrational partition functions in (b)

          T (K) |     1WHO    |     MSHO    |     E2DT
        --------------------------------------------------
         100.00 | 1.450E-186 | 2.901E-186 | 3.307E-186
         150.00 | 6.262E-120 | 1.257E-119 | 1.409E-119
         200.00 |  2.431E-86 |  4.944E-86 |  5.595E-86
         ...
        2000.00 |  2.664E+18 |  1.136E+19 |  1.204E+19
        2500.00 |  2.102E+23 |  9.464E+23 |  8.928E+23
```

If the user is also interested in the calculation of the corresponding thermodynamic functions, the keyword `thermo` should be specified:

```
python q2dtor.py S19 --rovibpf thermo
```

and their values will be stored in the `S19.tables` and `S19.out` files:

```
#-------------------------#
# Thermodynamic Functions #
#-------------------------#

    Note: All thermodynamics functions are calculated using as
    zero of energy the lowest zero point level of the torsional PES

    Units:
       * U, H, G => kcal/mol
       * S, Cp   =>  cal/mol/K

Thermodynamic functions for the 1WHO partition function

  T (K) |    U^o    |    H^o    |    S^o    |    G^o     |    Cp
  -----------------------------------------------------------------
  100.00 |     0.749 |     0.948 |    61.617 |    -5.214 |    11.840
  150.00 |     1.313 |     1.611 |    66.956 |    -8.432 |    14.759
  200.00 |     2.033 |     2.430 |    71.646 |   -11.899 |    18.074
    ...
 2000.00 |   110.678 |   114.653 |   186.761 |  -258.869 |    81.719
 2500.00 |   151.380 |   156.348 |   205.356 |  -357.042 |    84.786


Thermodynamic functions for the MSHO partition function

  T (K) |    U^o    |    H^o    |    S^o    |    G^o     |    Cp
  -----------------------------------------------------------------
  100.00 |     0.750 |     0.948 |    62.999 |    -5.351 |    11.873
  150.00 |     1.320 |     1.618 |    68.387 |    -8.640 |    15.024
  200.00 |     2.063 |     2.460 |    73.205 |   -12.181 |    18.731
  ...
 2000.00 |   111.731 |   115.705 |   190.169 |  -264.632 |    81.856
 2500.00 |   152.487 |   157.455 |   208.789 |  -364.517 |    84.874

     Thermodynamic functions for the E2DT partition function
```

```
T (K)  |    U^o    |    H^o    |    S^o    |    G^o    |    Cp
------------------------------------------------------------------
100.00 |     0.754 |     0.952 |    63.102 |    -5.358 |    12.036
150.00 |     1.340 |     1.638 |    68.619 |    -8.654 |    15.500
200.00 |     2.112 |     2.510 |    73.602 |   -12.211 |    19.390
 ...
2000.00 |  109.864 |   113.838 |   189.341 |  -264.843 |    80.054
2500.00 |  149.701 |   154.669 |   207.551 |  -364.208 |    83.004
```

The program also lists together the 1WHO, MSHO and E2DT thermodynamics properties to facilitate the comparison between the different approximations. For instance:

```
     Entropy (S^o)

T (K)  |    1WHO   |    MSHO   |    E2DT
-------------------------------------------------
100.00 |    61.617 |    62.999 |    63.102
150.00 |    66.956 |    68.387 |    68.619
200.00 |    71.646 |    73.205 |    73.602
 ...
2000.00 |  186.761 |   190.169 |   189.341
2500.00 |  205.356 |   208.789 |   207.551
```

Finally, together with the option `thermo`, it is possible to use option `--SI` which prints the thermodynamics properties in the International System of Units:

```
python q2dtor.py S19 --rovibpf thermo --SI
```

`Q2DTor` **tools**

`Q2DTor` also contains three tools that may be useful in some cases. In this chapter we explain how to use them.

4.1 `--pdf`

This tool generates a pdf file that contains a plot specifying the position of each stationary point in the 2D-PES. For system S19, this is done by executing:

```
python q2dtor.py S19 --pdf
```

In fact, this tool can be used before locating the stationary points, or even before fitting the 2D-PES to the Fourier series. The plots are generated according to the available information.

4.2 `--gts`

It may occur that the user prefers to optimize a given (or all) the stationary points without using `Q2DTor`. The information about those stationary points can be directly provided to `Q2DTor` if the corresponding `gts` file is placed in the SP folder. This tool can be used to generate a `gts` file from either *Gaussian* or *Orca* output files.

In the case of *Gaussian*, by executing

```
python q2dtor.py statpoint --gts gaussian
```

Q2DTor seeks for the `statpoint.out` and `statpoint.fchk` Gaussian files and generates the `statpoint.gts` file.

In the same way, for *Orca* users, by executing

```
python q2dtor.py statpoint --gts orca
```

Q2DTor seeks for the `statpoint.out`, `statpoint.engrad` and `statpoint.hess` files to generate the `statpoint.gts` file.

4.3 --icoords

This tool can be used to check if a given set of internal coordinates is able to reproduce the normal-mode frequencies for the same geometry when the Hessian was diagonalized in Cartesian coordinates. We reproduce the information in the help menu, which shows how to use this tool:

```
Basic usage: python q2dtor.py system --icoords mode icsfile

mode = sp
----------
Command: python q2dtor.py system --icoords sp

Use this mode to check the internal coordinates of all the
stationary points listed in the Q2DTor_files_system/system.splist file

If icsfile is not defined, the internal coordinates will be
 read from file Q2DTor_files_system/system.ics

To use the internal coordinates stored in the 'file.ics' file
just execute:
    python q2dtor.py system --icoords sp file.ics

mode = gts
----------
Command: python q2dtor.py system --icoords gts file.ics

Use this option to check the internal coordinates of the 'file.ics'
file by using the stationary point of the 'system.gts' file
```

Bibliography

[1] J. Zheng, T. Yu, E. Papajak, I. M. Alecu, S. L. Mielke, D. G. Truhlar, Phys. Chem. Chem. Phys. 13 (2011) 10885.

[2] R. Meana-Pañeda, A. Fernández-Ramos, J. Am. Chem. Soc. 134 (2012) 346; (E) 134 (2012) 7193.

[3] L. Simón-Carballido, J. L. Bao, T. V. Alves, R. Meana-Pañeda, D. G. Truhlar, A. Fernández Ramos, J. Chem. Theory Comput. 13 (2017) 3478.

[4] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, J. A. Montgomery, Jr., T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, 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, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, S. Dapprich, A. D. Daniels, M. C. Strain, O. Farkas, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, J. A. Pople, *Gaussian* 03, Revision C.02 Gaussian, Inc., Wallingford CT, (2004).

[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. 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. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, T. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. 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, D. J. Fox, *Gaussian* 09, Revision B.01, Gaussian, Inc., Wallingford CT, (2010).

[6] 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, D. J. Fox, *Gaussian* 16, Revision A-03, Gaussian, Inc., Wallingford CT, (2016).

[7] F. Neese Wiley Interdiscip. Rev. Comput. Mol. Sci. 2 (2012) 73.

[8] I. M. Alecu, J. Zheng, Y. Zhao, D. G. Truhlar, J. Chem. Theory and Comput. 6, (2010) 2872.

[9] A. Fernández-Ramos, J. Chem. Phys. 138 (2013) 134112.

[10] L. Simón-Carballido, A. Fernández-Ramos, J. Mol. Model. 20 (2014) 2190.

[11] G. Schaftenaar, Molden a pre- and post processing program of molecular and electronic structure, http://www.cmbi.ru.nl/molden/molden.html (January, 2018)