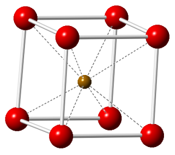
cosymlib Documentation

E. Bernuz, A. Carreras, M. Llunell
P. Alemany

Jun 05, 2023

cosymlib



Cosymlib is a python library for computing continuous symmetry & shape measures (CSMs & CShMs). Although its main aim is to provide simple and ready-to-use tools for the analysis of the symmetry & shape of molecules, many of the procedures contained in **cosymlib** can be easily applied to any finite geometrical object defined by a set of vertices or a by mass distribution function.

The basic tasks included in the current version of **cosymlib** are:

1. **Molecular structure analysis**
    - Calculation of Continuous Shape Measures (CShMs)
    - Calculation of Continuous Symmetry Measures (CSMs)
    - Calculation of Continuous Chirality Measures (CCMs)
2. **Electronic structure analysis**
    - Pseudosymmetry analysis of molecular orbitals & wavefunctions
    - Continuous Symmetry & Chirality Measures for the molecular electron density

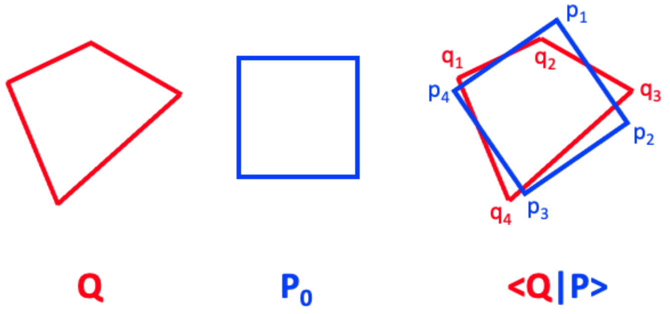
CHAPTER 1

Introduction

Cosymlib is a a python library for computing continuous symmetry & shape measures (CSMs & CShMs). Although most of the tools included in **cosymlib** have been devised especially with the purpose of analyzing the symmetry & shape of molecules as proposed initially by D. Avnir and coworkers [AVN], many of the procedures contained in **cosymlib** can be easily applied to any finite geometrical object defined by a set of vertices or a by mass distribution function.

1.1 Continuous Shape Measures (CShMs)

In a nutshell, the continuous shape measure $S_P(Q)$ of object Q with refespect to the reference shape P is an indicator of how much Q resembles another object P_0 with a given ideal shape, for instance a square as in the figure below.



Given that the shape is invariant upon translations, rotations, and scaling, the most evident way to compare the two objects is to translate, rotate and scale one of them, for instance P_0, until we maximize the overlap <Q|P> between Q and P, where P is the image of P_0 after these transformations.

If both the problem and the reference structures Q and P are defined as a set of vertices, we can define the shape measure simply as:

$$S_P(Q) = \min \frac{\sum_{i=1}^{N} \left| q_i - p_i \right|^2}{\sum_{i=1}^{N} \left| q_i - q_0 \right|^2} \times 100$$

where N is the number of vertices in the structures we are comparing, q_i and p_i are the position vectors of the vertices of Q and P, respectively, and q_0 the geometric center of the problem structure Q. The minimization in this equation refers to the relative position, orientation, and scaling that must be applied to P_0 to minimize the sum of squares of distances between their respective vertices, which is equivalent to maximizing the overlap <Q|P>. If the mismatch of the two shapes is described, as in the equation above by the distance between vertices of the two objects, a further minimization with respect to all possible ways to label the N vertices in the reference structure P_0 is also needed.

From the definition of $S_P(Q)$ it follows that if Q and P have exactly the same shape, then $S_P(Q) = 0$. Since $S_P(Q)$ is always positive, the larger its value, the less similar is Q to the ideal shape P. It can be shown that the maximum value for $S_P(Q)$ is 100, corresponding to the unphysical situation for which all vertices of Q collapse into a single point. A more detailed description of continuous shape measures and some of their applications in chemistry may be found in the following references [CShM]:

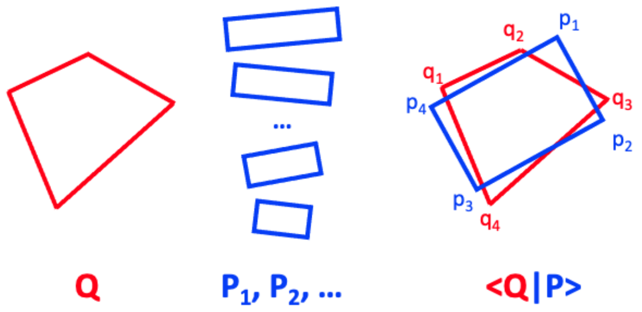
1.2 Continuous Symmetry Measures (CSMs)

To define a continuous measure for the degree of symmetry of an object one may proceed in the same way as for the definition of CShMs. The final result for the symmetry measure with respect to a given point symmetry group G, denoted as $S_G(Q)$, yields an expression totally analogous to the equation above, in which Q refers again to the problem structure, but where P is now the G-symmetric structure closest to Q:

$$S_G(Q) = \min \frac{\sum_{i=1}^{N} \left| q_i - p_i \right|^2}{\sum_{i=1}^{N} \left| q_i - q_0 \right|^2} \times 100$$

The minimization process in this case refers to the relative position of the two structures (translation), the orientation of the symmetry elements for the reference G-symmetric structure P, the scale factor, and again, the labeling of vertices of the symmetric structure. Note that although the same equation may be used both to define shape and symmetry measures, there is a fundamental difference between the two procedures: while in computing a shape measure we know in advance the reference object P_0 , in the case of symmetry measures the shape of the closest G-symmetric structure is, in principle, previously unknown and must be found in the procedure of computing $S_G(Q)$.

Consider, for instance that we would like to measure the rectangular symmetry for a given general quadrangle. Besides optimizing to seek for the translation, rotation, and scaling that leads to the optimal overlap of our quadrangle Q with a particular rectangle P as in a shape measure, we will need to consider also which is the ratio between the side lengths of the best matching rectangle and optimize also with respect to this parameter.

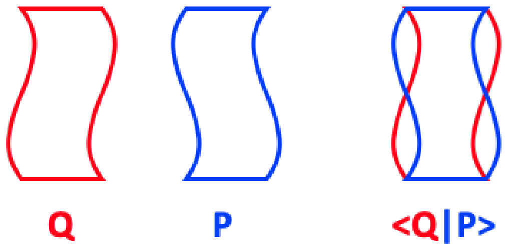


Although this additional optimization process may seem difficult to generalize for any given symmetry group, it has been shown that it is possible to do it efficiently using either the folding–unfolding algorithm or via the calculation of intermediate symmetry operation measures.

As in the case of shape measures, the values of CSMs are also limited between 0 and 100, with $S_G(Q) = 0$, meaning that Q is a G-symmetric shape. A more detailed description of continuous shape measures and some of their applications in chemistry may be found in the following references [CSM]:

1.3 Continuous Chirality Measures (CCMs)

A special mention should be made to chirality, a specific type of symmetry that has a prominent role in chemistry. A chiral object is usually described as an object that cannot be superposed with its mirror image. In this sense, we could obtain a continuous chirality measure by using the same equation as for shape measures just by replacing P by the mirror image of Q.



Technically speaking chirality is somewhat more complex since it implies the lack of any improper rotation symmetry and its CCM can be based on estimating how close a given object is from having this symmetry. Using the CSMs defined above, the continuous chirality measure can be defined as the minimal of all $S_G(Q)$ values for $S_n(Q)$ with n=1,2,4, … . In most cases it will be either for $G = S_1 = C_s$ or $G = S_2 = C_i$, whereas in a few cases we will have to look for $G = S_4$ or higher-order even improper rotation axes. Since in most cases visual inspection of the studied structure is enough in order to guess which one could be the nearest S_n group, a practical solution is just to calculate this particular $S_G(Q)$ value, or in case of doubt, a few $S_G(Q)$ values for different S_n and pick the smallest one. A more detailed description of continuous shape measures and some of their applications in chemistry may be found in the following references [CCM]:

1.4 CSMs for quantum chemical objects

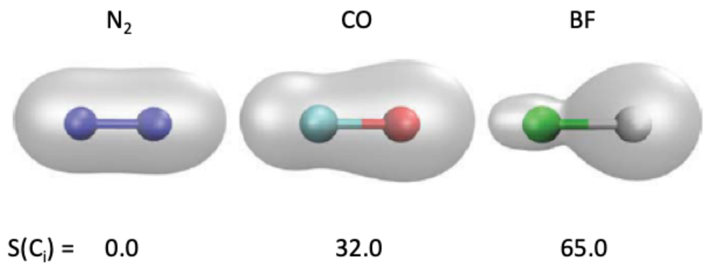
The use of the overlap <Q|P> between two general objects Q and P allows the generalization of continuous symmetry and shape measures to more complex objects that cannot be simply described by a set of vertices, such as matrices or functions. In this case the definition of the continuous symmetry measure is:

$$S_G(Q) = min\left[1 - \frac{\sum_{i=1}^{h}\langle Q|\hat{g}_i Q\rangle}{h\langle Q|Q\rangle}\right]$$

where Q is the given object and g_i the *h* symmetry operations comprised in the finite point symmetry group G. The minimization in this case just refers to the orientation of the symmetry elements that define the symmetry operations in G. The key elements in this definition are the overlap terms <Q|g_iQ> between the original object Q and its image under all the h symmetry operations g_i that form group G. The precise definition on how to obtain these overlaps depends, of course, on the nature of the object Q. For molecular orbitals as obtained in a quantum chemical calculation we have:

$$\langle \varphi|\hat{g}_i\varphi\rangle = \int \varphi(\boldsymbol{r})\hat{g}_i\varphi(\boldsymbol{r})d\tau$$

which is known as a SOEV (symmetry operation expectation value). For the electron density one can use an analogous expression for the corresponding SOEV by replacing the orbital (one electron wavefunction) by the whole electron density. Using this type of symmetry measures one is then able to compare the symmetry contents of the electronic structure of molecules, for instance by comparing the inversion symmetry measure for different diatomic molecules as in the example below:



The generalitzation of CSMs for functions, is of course, not limited to chemical applications and it permits extending the notion of continuous symmetry measures to geometrical objects beyond those defined by a set of vertices. A solid object of arbitrary shape, not restricted to a polyhedron, can be described by a function corresponding to a constant mass distribution, and its corresponding shape and symmetry measures can be easily computed by numerical integration to determine the SOEVs, avoiding the cumbersome minimization over vertex pairings that appear for objects that are defined by a set of vertices.

An interesting extension for functions which are not restricted to positive values, for instance, molecular orbitals, is the possibility of calculating continuous symmetry measures for each individual irreducible representation of a given point group. A more detailed description of the development and some applications of CSMs in quantum chemistry may be found in the following references [QCSMs]:

CHAPTER 2

Installation

cosymlib is available in both the GitHub and PyPI repositories (https://pypi.org/project/cosymlib/). Installation via PyPI is simpler and it is recommended for most users. Follow the instructions below to install **cosymlib** in your computer.

2.1 Installing cosymlib from PyPI (recommended)

This installation requires **pip** (https://pip.pypa.io/en/stable/installing/) to be installed in your system. We strongly recommend the use of python environments, for more details on this, refer to https://docs.python.org/3/library/venv.html. For most users the basic installation should proceed as follows:

1. Create a virtual environment at path <venv>

```
$ python3 -m venv <venv>
```

2. Activate this virtual environment

```
# on MAC / Linux
$ source <venv>/bin/activate

# on windows (powershell) [see note below]
C:\> <venv>\Scripts\Activate.ps1
```

3. Install **cosymlib**

```
$ pip install numpy
$ pip install cosymlib
```

4. Deactivate the virtual environment

```
$ deactivate
```

To use **cosymlib** you will need to activate the virtual environment every time that you open a new shell. On Linux/MAC all the scripts contained in **cosymlib** will be accessible in this environment:

```
$ source <venv>/bin/activate
$ <script_name> <script_options>
$ deactivate
```

On Windows, to execute the scripts you should type *python* followed by the full path of the script name:

```
C:\> python <venv>\Scripts\<script_name> <script_options>
```

Note: On Windows it may be necessary to add user execution permissions to activate the environment. To do this, open a poweshell as administrator and type:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

You should do this only once in order to gain execution permissions.

2.2 Installing cosymlib's source code

Alternatively, you can download the latest version of **`cosymlib`** from github using **`git`** (https://git-scm.com) and install it manually through the `setup.py` file using **`setuptools`** (https://setuptools.readthedocs.io/).

`cosymlib` contains libraries written in Fortran that require a compiler to build them. Before installing **`cosymlib`** make sure you have a working Fortran compiler installed in your system. For UNIX based systems you can install the GNU Fortran Compiler from package repositories by opening a terminal and typing the following commands:

- **Linux**

  On YUM-based systems (Fedora/RedHat/CentOS)

  ```
  sudo yum install yum-utils
  ```

  On APT-based systems (Debian/Ubuntu)

  ```
  sudo apt-get build-dep
  ```

- **Mac**

1. Install command-line tools:

   ```
   xcode-select --install
   ```

2. Get Homebrew following the instructions at https://brew.sh, and install GCC formula by:

   ```
   brew install gcc
   ```

- **Windows**

1. Install the Windows development environment **`Visual Studio`** (https://developer.microsoft.com/en-us/windows/downloads/)
2. Install C/Fortran compiler for Windows. We have tested and recommend **`mingw`** (https://www.mingw-w64.org)

To install **`cosymlib`**, download the source code using **`git`** in your computer by typing:

```
git clone https://github.com/GrupEstructuraElectronicaSimetria/cosymlib.git
```

This creates a copy of the repository in your computer. You can keep it updated by synchronizing it with the GitHub repository by using the command:

```
git pull
```

Once this is done, move to the repository root directory (where `setup.py` is found) and type the following command to install **cosymlib** :

```
python setup.py install --user
```

Note: The `requirements.txt` file located at the repository root directory contains a list of all dependency python modules needed for **cosymlib** to run. If any of them are missing in your system you will need to install them before running **cosymlib**.

In both cases (PyPI & Github installations) the code will be installed as a **python** module. To check that it is properly installed you can run the **python** interpreter and execute:

```
import cosymlib
```

If the execution does not show any errors, then **cosymlib** has been installed successfully.

Note: For users with Apple M1, the **scipy** library might not properly install when following the instructions above. To solve this, install it manually:

```
brew install openblas
brew install lapack
brew install python
pip install cython pybind11 pythran numpy
OPENBLAS=$(brew --prefix openblas) CFLAGS="-falign-functions=8 ${CFLAGS}" pip install␣
↪--no-use-pep517 scipy==1.7.0
```

Note: When using an IDE, remember to select the python interpreter in the hombrew path. To find it:

```
which python3
>> /opt/homebrew/bin/python3
```

CHAPTER 3

How to use cosymlib

Cosymlib is a a python library for computing continuous symmetry & shape measures (CSMs & CShMs). Besides using the APIs contained in **cosymlib** to build your own custom-made python programs we have also written some general scripts to perform standard tasks such as calculating a continuous shape measure for a given structure without the need of writing a python script. All this general task scripts are called using a similar syntax which includes the name of the script, the name of the file containing the structural data, and optional arguments specifying the tasks we want to perform:

```
$ script filename -task1 -task2 ... -taskn
```

For instance, consider a `struct.xyz` file containing the following structural information for a H_4 molecule in an approximately square geometry:

```
4
H4 Quadrangle
H    1.1   0.9  0.0
H   -1.0   1.1  0.0
H   -0.9  -1.2  0.0
H    1.1  -1.0  0.0
```

If we would like to compute the square shape measure S(SP-4) for this 4-vertex polygon we simply can call the shape script indicating the name of our .xyz file containing the coordinates and use the `-m` flag (m stands for measure) with the SP-4 label to indicate that we want to compute a shape measure using a perfect square (SP-4 stands for square planar structure with 4 vertices) as the reference shape:

```
$ shape struct.xyz -m SP-4
```

and the shape script wil call the APIs in **cosymlib** to read first our input file, generate a molecule object, calculate the S(SP-4) continuous shape measure for it, and print the result of the calculation on the screen:

```
----------------------------------------------------------------------
 COSYMLIB v0.10.5
 Electronic Structure & Symmetry Group
 Institut de Quimica Teorica i Computacional (IQTC)
```

(continues on next page)

(continued from previous page)

```
 Universitat de Barcelona
-----------------------------------------------------------------------
 Structure         SP-4
 H4,               0.520,
-----------------------------------------------------------------------
                   End of calculation
-----------------------------------------------------------------------
```

If, for instance, we also want the coordinates for the square with the optimal overlap with our problem structure, we just need to include the `-s` flag (where s stands for structure) in our call:

```
$ shape struct.xyz -m SP-4 -s
```

A longer, explicit version for some flags is also available using a double - sign. With these explicit flags the previous command becomes:

```
$ shape struct.xyz --measure SP-4 --structure
```

The general task scripts include also `gsym` and `cchir` for calculating continuous symmetry and chirality measures for polyhedral structures, `shape_map` for plotting shape maps, as well as the `esym` and `mosym` scripts for the continuous symmetry analysis of electron densities and the pseudosymmetry analysis of molecular orbitals, respectively.

Besides these six basic scripts, we have also developed `cosym`, a general script that allows to perform any of the basic calculations above. We could, for instance, use directly `cosym` to calculate the previous shape measure using the following command:

```
$ cosym struct.xyz -shp_m SP-4 -s
```

Note that when using `cosym` some of the optional flags in `shape` change to indicate which type of calculation we would like to perform. For instance, `-m` becomes `-shp_m` to distinguish it from a symmetry measure (`-m` flag in `sym`) that becomes `-sym_m` when called from `cosym`. On the other hand, other arguments such as `-s`, which have the same meaning when calculating shape, symmetry or chirality measures, remain unchanged when used in combination with the general `cosym` script.

Taking into account the users of our previous programs, we have also written a stand-alone script `shape_classic` which is able to read an `old_shape.dat` input file containing both the structural information and the necessary keywords to run a full CShM calculation as it was done in our previous `SHAPE` program.

In the sections below you can find a detailed description of all stand-alone scripts as well as all APIs included in the present distribution of **cosymlib**.

3.1 General task scripts

The **cosymlib** library includes several scripts to perform basic tasks that can be run in a terminal as command line instructions, without the need of writing a full python script. The following subsections describe the general usage of all of them.

3.1.1 shape

`shape` can be used for computing continuous shape measures (CShMs) for geometrical structures defined by the cartesian coordinates for a set of vertices, for instance, a molecular structure defined by the positions of the atomic nuclei.

The minimal information needed to run `shape` is an input file containing the coordinates of a set of vertices. Since `shape` is mainly intended to be used in the context of structural chemistry, the main source of structural information will be a `fname.xyz` file containing a molecular geometry in xyz format (http://en.wikipedia.org/wiki/XYZ_file_format).

An example of a `cocl6.xyz` file with the structure for a perfect octahedral $CoCl_6$ fragment with 2.4Å Co-Cl interatomic distances is:

```
7
CoCl6
Co    0.0   0.0  0.0
Cl   -2.4   0.0  0.0
Cl    2.4   0.0  0.0
Cl    0.0   2.4  0.0
Cl    0.0  -2.4  0.0
Cl    0.0   0.0  2.4
Cl    0.0   0.0 -2.4
```

The first line in the file indicates the number of atoms (vertices in the geometric structure), the second line contains a free-format descriptive title, and the following lines (as many as indicated in the first line) contain a label (usually the atomic symbol) and the cartesian coordinates x, y, z for each atom (vertex) in the structure.

`fname.xyz` files read by shape may contain a single structure as in the previous example or multiple structures (all with the same number N of atoms). In this case you must append a

```
N
Structure_name
label_1  x1  y1  z1
 ...
label_N  xN  yN  zN
```

block for each structure, without leaving any blank lines between them. Note that even if the number of vertices N must be the same for all structures, it is mandatory to include it explicitly for each block.

Shape is also able to read input structures from files in other formats used in structural chemistry. A detailed description of the structural files read by shape can be found in the information on input formats section.

The basic call to the shape script must provide the the file containing the input structure and the reference shape with respect to which the shape measure is calculated.

```
$ shape input_file -m SH
```

where `input_file` is a file containing the structural information in a valid format, for instance a .xyz file, `-m` requests a shape measure calculation, and is `SH` a label indicating a given reference structure, for instance `SP-4` for a square or `OC-6` for an octahedron. Note that the reference shape must be compatible with the problem structure, i. e., they must both contain the same number of atoms (vertices). To obtain a list of the labels for the reference structures compatible with a given input structure you may use:

```
$ shape input_file -l
```

If `input_file` contains, for instance, a structure with 6 atoms (vertices) your will get the following output on screen:

```
Available reference structures with 6 Vertices:

Label       Sym       Info

HP-6        D6h       Hexagon
PPY-6       C5v       Pentagonal pyramid
OC-6        Oh        Octahedron
TPR-6       D3h       Trigonal prism
JPPY-6      C5v       Johnson pentagonal pyramid J2
```

We can then use this information to compute the desired continuous shape measure

```
$ shape input_file -m OC-6
```

if we want to compute the octahedral shape measure. For a file containing a perfect octahedron of carbon atoms:

```
6
C6_octa
    C   -1.0   0.0  0.0
    C    1.0   0.0  0.0
    C    0.0   1.0  0.0
    C    0.0  -1.0  0.0
    C    0.0   0.0  1.0
    C    0.0   0.0 -1.0
```

the program will return:

```
Starting...
----------------------------------------------------------------------
COSYM v0.7.4
Electronic Structure Group,  Universitat de Barcelona
----------------------------------------------------------------------

Structure     OC-6

C6_octa,        0.000

End of cosym calculation
```

Indicating that it is indeed a perfect octahedron, S(OC-6) = 0.000. If we want to know how far this octahedron is from the reference triangular prism we may use:

```
$ shape input_file -m TPR-6
```

which returns a value of S(TPR-6) = 16.737. Note that since shape measures are independent from size, position, or orientation of the problem structure, we would obtain exactly the same values for any perfect octahedron in `input_file`.

When studing the shape of the coordination sphere around a given atom, let us say a transition metal atom M surrounded by n atoms L coming from the surrounding ligands, it is possible to consider just the L_n polyhedron or a centered ML_n "polyhedron". We will obtain different information from each calculation. While considering the L_n polyhedron, we will know how different it is from the ideal references, but if we are interested in distortions due to displacements of the central atom from the geometric center we will need to compare the centered ML_n "polyhedron" with the ideal references where the central atom is located at the geometric center of the object. Since the central M atom and the n surrounding ligands are not equivalent (no M <-> L permutations are allowed when computing the shape measure) it is necessary to indicate that the structure in `input_file` corresponds to a centered ML_n polyhedron and not to a simple L_{n+1} polyhedron. This is achieved by including the `-c N` flag in the shape command, where N is an integer number indicating the position of the central atom in `input_file` (for a file with multiple structures the central atom should be in the same position for all of them). If one uses the `cocl6.xyz` file above as `input_file` indicating that the first atom in the structure (the Co atom) is in the center (`-c 1`)

```
$ shape cocl6.xyz -l -c 1
```

we get the following valid labels:

```
Available reference structures with 6 Vertices:

Label       Sym       Info

HP-6        D6h       Hexagon
PPY-6       C5v       Pentagonal pyramid
OC-6        Oh        Octahedron
TPR-6       D3h       Trigonal prism
JPPY-6      C5v       Johnson pentagonal pyramid J2
```

note that, although these labels the same as those for a structure with 6 atoms where we do not include a central atom, a calculation including the `-c N` flag is not equivalent to a calculation where the central atom is ignored, that is just for the L_n polyhedron. If one wants to calculate the shape measure just for the “empty” L_n shell one needs to prepare a different input file deleting the line corresponding to the central atom and reducing the number of atoms by 1.

If we try omitting the `-c N` flag for the `cocl6.xyz` file we get a different result. Using

```
$ shape cocl6.xyz -l
```

we find:

```
Available reference structures with 7 Vertices:

Label       Sym       Info

HP-7        D7h       Heptagon
HPY-7       C6v       Hexagonal pyramid
PBPY-7      D5h       Pentagonal bipyramid
COC-7       C3v       Capped octahedron
CTPR-7      C2v       Capped trigonal prism
JPBPY-7     D5h       Johnson pentagonal bipyramid J13
JETPY-7     C3v       Johnson elongated triangular pyramid J7
```

which are the possible reference structures for empty L_7 polyhedra, since now the Co atom is being considered on equal foot to all other six Cl atoms, even if this might make no sense from a chemical point of view. The list of currently available reference structures in the cosymlib program is at *the page end*.

To calculate the octahedral shape measure for the $CoCl_6$ structure contained in the `cocl6.xyz` file we will use:

```
$ shape cocl6.xyz -c 1 -m OC-6
```

which will return a S(OC-6)= 0.000 value since the six Cl atoms in the structure form a perfect octahedron with the Co atom sitting exactly in its geometric center. Note also that, as shown in this example, the position of the `-c N` and `-m OC-6` flags, or the `input_file` in the call to the shape script is totally irrelevant and any combination such as:

```
$ shape cocl6.xyz -c 1 -m OC-6

$ shape  -c 1 -m OC-6 cocl6.xyz

$ shape  -m OC-6 cocl6.xyz  -c 1
```

will result in exactly the same CShM calculation.

Somtimes we are not just interested in the shape measure, that is, how far our problem shape is from the ideal reference, but also we would like to have the coordinates of the ideal reference shape with the size, position, and orientation that is closest to our problem shape. To achieve this we just need to include the `-s` flag in our call.

Let us consider a `struct.xyz` file containing the geometry for an approximately square H_4 molecule.

```
4
H4 Quadrangle
H    1.1   0.9  0.0
H   -1.0   1.1  0.0
H   -0.9  -1.2  0.0
H    1.1  -1.0  0.0
```

If we want to know how far it is from having a perfectly square geometry and which is the closest square to its actual distorted structure we may use:

```
$ shape struct.xyz  -m SP-4 -s
```

which will yield:

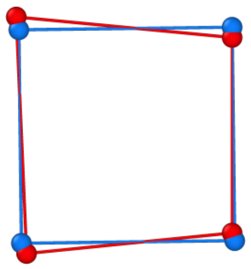
```
Starting...
----------------------------------------------------------------------
 COSYM v0.7.4
 Electronic Structure Group,  Universitat de Barcelona
----------------------------------------------------------------------

Structure     SP-4

H4,            0.520

4
H4
H     1.100000    0.900000    0.000000
H    -1.000000    1.100000    0.000000
H    -0.900000   -1.200000    0.000000
H     1.100000   -1.000000    0.000000
4
H4_SP-4
H     1.100000    1.000000    0.000000
H    -0.975000    0.975000    0.000000
H    -0.950000   -1.100000    0.000000
H     1.125000   -1.075000    0.000000
```

from which we find that the problem structure has an approximate square planar geometry with a small departure from the ideal shape, S(SP-4) = 0.520, together with the coordinates of the problem structure and its closest ideal (square) structure, which we can use to plot the superposition of problem structure (in red) and the ideal reference (in blue):



Other optional flags to control the execution of shape are:

`shape -h` (no input file needed) returns a list of all available flags and their use

Running `shape` with the `-o file_name` flag prints all output into the `file_name` file

Running `shape` with the `-r` flag prints the coordinates of the reference shape in a file named `Ln.xyz` or `MLn.xyz` where n is the number of vertices of the polyhedron.

The `- info` flag may be used to print the coordinates of the input structure

You may use `-fixp` to disable the minimization over the permutation of vertices while searching for the shape measure. If you include the `-fixp` in your call, the minimization will be carried out considering only the distance between the i-th vertex in the problem structure with the i-th vertex in the reference shape. Although this option allows a drastic reduction of the computational cost, it should be used with care since the actual shape measure is defined for the permutation thats gives the lowest value of S. For large structures the `-fixp` option will probably be the only way of obtaining a shape measure, but this procedure is only justified for structures with small distortions from the reference structure. Before doing the actual calculation it will be necessary to run shape with the `-r` flag to print the coordinates of the reference shape and order the vertices in the problem structure accordingly.

A quite useful flag is `-cref filename` that allows the user to specify a custom reference structure in the filename file. Use this option if you want to use a reference structure different from any of those provided by shape. To use this feature you will need to include the `-m custom` flag in your call:

```
$ shape input_file -m custom -cref filename
```

Besides the shorthand version of the flags described above, it is also possible to use an explicit version by writing them preceded by a double `--` sign. The explicit versions of the flags are:

Short Flag	Explicit flag
`-h`	`--help`
`-m`	`--measure`
`-l`	`--labels`
`-s`	`--structure`
`-o`	`--output_name`
`-c`	`--central_atom`
`-r`	`--references`
`-cref`	`--custom_ref`
`-fixp`	`--fix_permutation`

Sometimes, to avoid a cumbersome repetition of several flags in the call of the shape module we may write all flags in an input file and just call shape indicating the file with the structural input and the file with the options of the calculation. For example, if the original call is:

```
$ shape struct.xyz -c 1 -m OC-6 -s -o struct.out
```

You can create a new file called `struct.yml` (the name for the file can be freely chosen and does not need to be the same as for the structure) containing the options in YAML format (http://en.wikipedia.org/wiki/YAML):

```
central_atom :   1
measure      :   OC-6
structure    :   True
output       :   struct.out
```

and then call shape just using:

```
$ shape struct.xyz struct.yml
```

Note that you must use the explicit version of the flags in the `.yml` file. If a flag such as `-s` does not need any additional argument, you must include `True` in the `.yml` file.

3.1.2 shape_classic

To run `shape_classic` you only need an `old_shape.dat` input file containing both the structural information and the necessary keywords to run a full CShM calculation as in the old `SHAPE` program:

```
$ shape_classic old_shape.dat
```

The script will perform all tasks indicated in the input file, creating the necessary output files, normally `old_shape.out` and `old_shape.tab` with the same information as when using our previous `SHAPE` program. Follow the link below for a pdf version of the user guide for SHAPE ver. 2.1 where you will find all information to perform a continuous shape analysis using this option.

`SHAPE ver. 2.1 User's guide`

3.1.3 gsym

In the case of running a continuous symmetry measure (CSM), the `gsym` script is required plus an input file containing a geometric structure as the one used in the shape script. Since the main difference with the continuous shape measures is that the reference structure now must contain one or more symmetry elements, the user will need to specify which symmetry operation wants to analyse for the input geometry. The Th_8molecule can be a good example to show the S_4symmetry that the molecule contains. The Th8.xyz file is shown below:

```
8
Th8
Th  -16.80062  -0.55052  -13.74098
Th  -12.80008  -0.09601  -14.54017
Th  -15.57778   1.38797  -17.17300
Th  -20.18823  -1.83274  -15.67222
Th  -16.79762  -4.14442  -15.76983
Th  -12.79709  -3.68990  -16.56902
Th  -18.96539   0.10576  -19.10423
Th  -15.57478  -2.20592  -19.20184
```

The simplest way to compute the S_4CSM measure for $Th_8$is to run the following command:

```
$ gsym Th8.xyz -m S4
```

which is equivalent to:

```
$ gsym -m S4 Th8.xyz
```

and will return the CSM result in the cosymlib format:

```
----------------------------------------------------------------------
 COSYMLIB v0.9.5
 Electronic Structure & Symmetry Group
 Institut de Quimica Teorica i Computacional (IQTC)
 Universitat de Barcelona
----------------------------------------------------------------------

Evaluating symmetry operation : S4

th8           0.000

----------------------------------------------------------------------
                    End of calculation
----------------------------------------------------------------------
```

If the user wants to save this information in a file, the `-o (or --output)` flag should be added as well as the output file name. For example, the following call will execute the previous CSM calculation and will store the information in the Th8.txt file:

```
$ gsym Th8.xyz -m S4 -o Th8.txt
```

Additional commands (or flags) can be added to the command line to control the type of calculation that will be run. For example, the `-c N` command, where N is the position of an atom of the input file, explicitly tells the program which atom acts as a central atom of the molecule and that only could permut by himself. Alternatively, the `--center x y z`, where x, y and z are the 3D coordinates of a point in space, will set the origin of a structure. Similarly, the `-s` and `-l` commands work as in the shape case. The first returns the coordinates of the ideal reference geometry with the size, position, and orientation that is closest to our problem shape and belongs to a symmetry group. The second gives information of the available symmetry groups in cosymlib

```
Available symmetry groups

E      Identity Symmetry
Ci     Inversion Symmetry Group
Cs     Reflection Symmetry Group
Cn     Rotational Symmetry Group (n: rotation order)
Sn     Rotation-Reflection Symmetry Group (n: rotation-reflection order)
```

The `--info` flag may be used to print the coordinates of the input structure. Supplementary, other flags are available to the user to control if the calculation should take into account the connectivity `--ignore_connectivity`, the atom nature `--ignore_atoms_labels`, the connectivity threshold that controls if two atoms are connected `--connectivity_thresh` or the file that contains a custom connectivity `--connectivity_file`. In the last case, the format of the connectivity file should be as follow,

```
1    2    3    4    5
2    1
3    1
```

(continues on next page)

(continued from previous page)

```
4    1
5    1
```

where each number is related to the position of the each atom on the input file. For example, in the first line, the connectivity file tells the program that the atom in position one of the the input file is connected to the second, third, fourth and fifth atoms, while the second line tells that the second atom is only connected to the first atom. For a methane molecule where the input file is written as follow,

```
5
Methane
  C      0.0000      0.0000      0.0000
  H      0.5288      0.1610      0.9359
  H      0.2051      0.8240     -0.6786
  H      0.3345     -0.9314     -0.4496
  H     -1.0685     -0.0537      0.1921
```

the connectivity file force the carbon atom to be connected to all hydrogen atoms and viceversa. Finally, a list of available flags and their uncontracted form is listed below.

Short Flag	Explicit flag
`-h`	`--help`
`-m`	`--measure`
`-s`	`--structure`
`-c`	`--central_atom`
`-o`	`--output`
`-l`	`--labels`
`-v`	`--version`

Note: the actual program only runs for C_n, C_s, C_i and S_n symmetry groups as well as their symmetry operations.

3.1.4 cchir

The `cchir` script allows the user to calculate the chirality of a structure by calculating the continuous symmetry measure of the S_n improper rotation group. By default the `-m` flag will measure the S_1 symmetry which is equivalent ot the C_s symmetry group. However, the user can control the order of the impropert rotation by the `-order n` flag where n=1,2,4,6,… Other additional flags derived from the gsym script that have the same interaction with the chirality measure are available and list below. For more information of these commands go to gsym subsection of this page.

Common cchir and gsym commands
`-o` or `--ouput`
`-c` or `--central_atom`
`-v` or `--version`
`--info`
`--center`

3.1.5 esym

We are currently working on this feature of the program regarding the electronic structure symmetry of molecules, therefore the actual script is under construction.

3.1.6 cosym

This script is a general script that cover all the previous scripts.

3.2 Specific task scripts

In this section the specific task scripts are described.

3.2.1 shape_map

The `shape_map` script calculate the continuous shape measures of a single or multiple structures with two reference structures in the same way the shape script does. However, it computes additional information like the minimum distortion pathway between the two reference structures, plus the deviation and the generalized coordinate of each user's structure. The most common commands available in the script are similar to the commands found in the shape script. The required commands are the `-m_1 SH1` (or ``--measure_1`) and the `-m_2 SH2` flags, where SH1 and SH2 are the reference structure labels available in the program. Additionally, these flags can be substituted by the `-m_custom_1 SH1` or the `-m_custom_2 SH2` to indicate the program that SH1 and/or SH2 are the files containing a custom reference structure. Moreover, a set of flags are available to control the different plot options on the shape_map. The `--min_dev MIN_DEV` and `--max_dev MAX_DEV` will only show the structures that are between the minimum and maximum deviation values (MIN_DEV and MAX_DEV), while the `--min_gco MIN_GCO` and `--max_gco MAX_GCO` show the structure that are at the MIN_GCO to MAX-GCO range of the generalized coordinate. In addition, the user can plot more resolution minimal distortion pathways by setting the number of structures needed to compute the curve with the `--n_points N_POINTS` flag.

Finally, a set of mutual flags found in all scripts is available and listed below:

Short Flag	Explicit flag
`-h`	`--help`
`-l`	`--labels`
`-o`	`--output_name`
`-c`	`--central_atom`
`-v`	`--version`

3.3 Using cosymlib's APIs

The current API's are under construction and a set of tutorials will be provide in a near future.

3.4 Shape references

Here are the available shape reference's labels and their symmetry that can be used by the shape program.

Vertices	Label	Shape	Symmetry
2	L-2	Linear	$D_{\infty h}$
	vT-2	Divacant tetrahedron (V-shape, 109.47°)	C_{2v}
	vOC-2	Tetravacant octahedron (L-shape, 90.00°)	C_{2v}
3	TP-3	Trigonal planar	D_{3h}
	vT-3	Pyramid[b] (vacant tetrahedron)	C_{3v}
	fac-vOC-3	fac-Trivacant octahedron	C_{3v}
	mer-vOC-3	mer-Trivacant octahedron (T-shape)	C_{2v}
4	SP-4	Square	D_{4h}
	T-4	Tetrahedron	T_d
	SS-4	Seesaw or sawhorse[b] (cis-divacant octahedron)	C_{2v}
	vTBPY-4	Axially vacant trigonal bipyramid	C_{3v}
5	PP-5	Pentagon	D_{5h}
	vOC-5	Vacant octahedron[b] (Johnson square pyramid, J1)	C_{4v}
	TBPY-5	Trigonal bipyramid	D_{3h}
	SPY-5	Square pyramid[c]	C_{4v}
	JTBPY-5	Johnson trigonal bipyramid (J12)	D_{3h}
6	HP-6	Hexagon	D_{6h}
	PPY-6	Pentagonal pyramid	C_{5v}
	OC-6	Octahedron	O_h
	TPR-6	Trigonal prism	D_{3h}
	JPPY-6	Johnson pentagonal pyramid (J2)	C_{5v}
7	HP-7	Heptagon	D_{7h}
	HPY-7	Hexagonal pyramid	C_{6v}
	PBPY-7	Pentagonal bipyramid	D_{5h}
	COC-7	Capped octahedron[a]	C_{3v}
	CTPR-7	Capped trigonal prism[a]	C_{2v}
	JPBPY-7	Johnson pentagonal bipyramid (J13)	D_{5h}
	JETPY-7	Elongated triangular pyramid (J7)	C_{3v}
8	OP-8	Octagon	D_{8h}
	HPY-8	Heptagonal pyramid	C_{7v}
	HBPY-8	Hexagonal bipyramid	D_{6h}
	CU-8	Cube	O_h
	SAPR-8	Square antiprism	D_{4d}
	TDD-8	Triangular dodecahedron	D_{2d}
	JGBF-8	Johnson - Gyrobifastigium (J26)	D_{2d}
	JETBPY-8	Johnson - Elongated triangular bipyramid (J14)	D_{3h}
	JBTP-8	Johnson - Biaugmented trigonal prism (J50)	C_{2v}
	BTPR-8	Biaugmented trigonal prism	C_{2v}
	JSD-8	Snub disphenoid (J84)	D_{2d}
	TT-8	Triakis tetrahedron	T_d
	ETBPY-8	Elongated trigonal bipyramid (see 8)	D_{3h}
9	EP-9	Enneagon	D_{9h}
	OPY-9	Octagonal pyramid	C_{8v}
	HBPY-9	Heptagonal bipyramid	D_{7h}
	JTC-9	Triangular cupola (J3) = trivacant cuboctahedron	C_{3v}
	JCCU-9	Capped cube (Elongated square pyramid, J8)	C_{4v}
	CCU-9	Capped cube	C_{4v}

Continued on next page

Table 1 – continued from previous page

Vertices	Label	Shape	Symmetry
	JCSAPR-9	Capped sq. antiprism (Gyroelongated square pyramid J10)	C_{4v}
	CSAPR-9	Capped square antiprism	C_{4v}
	JTCTPR-9	Tricapped trigonal prism (J51)	D_{3h}
	TCTPR-9	Tricapped trigonal prism	D_{3h}
	JTDIC-9	Tridiminished icosahedron (J63)	C_{3v}
	HH-9	Hula-hoop	C_{2v}
	MFF-9	Muffin	C_s
10	DP-10	Decagon	D_{10h}
	EPY-10	Enneagonal pyramid	C_{9v}
	OBPY-10	Octagonal bipyramid	D_{8h}
	PPR-10	Pentagonal prism	D_{5h}
	PAPR-10	Pentagonal antiprism	D_{5d}
	JBCCU-10	Bicapped cube (Elongated square bipyramid J15)	D_{4h}
	JBCSAPR-10	Bicapped square antiprism (Gyroelongated square bipyramid J17)	D_{4d}
	JMBIC-10	Metabidiminished icosahedron (J62)	C_{2v}
	JATDI-10	Augmented tridiminished icosahedron (J64)	C_{3v}
	JSPC-10	Sphenocorona (J87)	C_{2v}
	SDD-10	Staggered dodecahedron (2:6:2)[e]	D_2
	TD-10	Tetradecahedron (2:6:2)	C_{2v}
	HD-10	Hexadecahedron (2:6:2, or 1:4:4:1)	D_{4h}
11	HP-11	Hendecagon	D_{11h}
	DPY-11	Decagonal pyramid	C_{10v}
	EBPY-11	Enneagonal bipyramid	D_{9h}
	JCPPR-11	Capped pent. Prism (Elongated pentagonal pyramid J9)	C_{5v}
	JCPAPR-11	Capped pent. antiprism (Gyroelongated pentagonal pyramid J11)	C_{5v}
	JaPPR-11	Augmented pentagonal prism (J52)	C_{2v}
	JASPC-11	Augmented sphenocorona (J87)	C_s
12	DP-12	Dodecagon	D_{12h}
	HPY-12	Hendecagonal pyramid	C_{11v}
	DBPY-12	Decagonal bipyramid	D_{10h}
	HPR-12	Hexagonal prism	D_{6h}
	HAPR-12	Hexagonal antiprism	D_{6d}
	TT-12	Truncated tetrahedron	T_d
	COC-12	Cuboctahedron	O_h
	ACOC-12	Anticuboctahedron (Triangular orthobicupola J27)	D_{3h}
	IC-12	Icosahedron	I_h
	JSC-12	Square cupola (J4)	C_{4v}
	JEPBPY-12	Elongated pentagonal bipyramid (J16)	D_{6h}
	JBAPPR-12	Biaugmented pentagonal prism (J53)	C_{2v}
	JSPMC-12	Sphenomegacorona (J88)	C_s
20	DD-20	Dodecahedron[d]	I_h
24	TCU-24	Truncated cube	O_h
	TOC-24	Truncated octahedron	O_h
48	TCOC-48	Truncated cuboctahedron	O_h
60	TRIC-60	Truncated icosahedron (fullerene)	I_h

a Non regular polyhedron.

b A regular polyhedron with one or two vertices removed.

c Spherical distribution of vertices with mass center at the origin (apical-basal bond angles of 104.45°).

d For polyhedra with more than 12 vertices the calculation times may be unpractical, for now avoid this calculations an upgrade is comming soon.

e This is a chiral polyhedron. It must be noticed that the algorithm used by Shape does not distinguish the two enantiomers of a chiral shape. Therefore, whenever a chiral reference polyhedron is used, the resulting shape measures may not refer to that specific polyhedron but to its enantiomer.

CHAPTER 4

Tutorials

CHAPTER 5

Useful links

Electronic Structure Group

iqtc Institut de Química Teòrica i Computacional UNIVERSITAT DE BARCELONA

DIPC Donostia International Physics Center

Prof. David Avnir Homepage

Dr. Inbal Tuvi-Arad Homepage

Kimika Teorikoa

Euskal Herriko Unibertsitatea

CHAPTER 6

API Reference

6.1 Cosymlib

class cosymlib.**Cosymlib**(*structures*, *ignore_atoms_labels=False*, *ignore_connectivity=False*, *connectivity=None*, *connectivity_thresh=None*, *charge_eh=0*, *mode=0*, *precision=3*)

This class contains all the high level methods used in the command line interface scripts. The methods return formatted results of multiple molecules calculations

Parameters

- **structures** (list, *Geometry*, *Molecule*) – List of *Geometry* or *Molecule*
- **ignore_atoms_labels** (*bool*) – Ignore atomic element labels is symmetry calculations
- **ignore_connectivity** (*bool*) – Ignore connectivity in symmetry calculations
- **connectivity** (*list*) – List of pairs if atom indices that are considered connected
- **connectivity_thresh** (*bool*) – Connectivity threshold (Ionic radius is used as reference)

get_geometries()

Get the geometries

Returns List of *Geometry* objects

Return type list

get_molecule_path_deviation(*shape_label1*, *shape_label2*, *central_atom=0*)

Get molecule path deviation

Parameters

- **shape_label1** (*str*) – First shape reference label
- **shape_label2** (*str*) – Second shape reference label

- **`central_atom`** (*`int`*) – Position of the central atom

`get_n_atoms`()
Get the number of atoms if all structures contains the same number of atoms, else raise exception.

Returns Number of atoms

Return type int

`get_point_group`(*tol=0.01*)
Get the point group of all structures

Parameters **`tol`** (*`float`*) – Tolerance

Returns a list of point group labels

Return type list

`get_shape_measure`(*label*, *kind*, *central_atom=0*, *fix_permutation=False*)
Get shape measure

Parameters

- **`label`** (*`str`*) – Reference shape label
- **`kind`** (*`str`*) – function name suffix
- **`central_atom`** (*`int`*) – Position of the central atom
- **`fix_permutation`** (*`bool`*) – Do not permute atoms during shape calculations

Returns Shape measures

Return type list

`molecules`
Get the molecules

Returns List of *`Molecule`* objects

Return type list

`print_chirality_measure`(*order=1*, *central_atom=0*, *center=None*, *permutation=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Prints the chirality measure

Parameters

- **`order`** (*`int`*) – Order of the chirality measure (1: Cs, 2:Ci, n:S_n)
- **`central_atom`** (*`int`*) – Position of the central atom
- **`center`** (*`int`*) – Center of symmetry in Cartesian coordinates. If None center is optimized
- **`permutation`** (*`list, tuple`*) – Define permutation
- **`output`** (*`hook`*) – Display hook

`print_esym_orientation`(*group*, *axis=None*, *axis2=None*, *center=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)
Prints down an xyz file of the molecule with the orientation_axis

Parameters

- **`group`** (*`string`*) – Symmetry group

- **axis** (*list*) – Main symmetry axis of group
- **axis2** (*list*) – Secondary symmetry axis of group
- **center** (*list, tuple*) – Center
- **output** (*hook*) – Display hook

print_geometric_measure_info(*label, multi=1, central_atom=0, center=None, output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Prints geometric symmetry measure verbose

Parameters

- **label** (*str*) – Symmetry point group label
- **multi** (*int*) – Number of symmetry axis to find
- **central_atom** (*int*) – Position of the central atom
- **center** (*list*) – Center of symmetry in Cartesian coordinates. If None center is optimized
- **output** – Display hook

Type hook

print_geometric_symmetry_measure(*label, central_atom=0, center=None, permutation=None, output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Prints geometric symmetry measure in format

Parameters

- **label** (*str*) – Symmetry point group label
- **central_atom** (*int*) – Position of the central atom
- **center** (*list, tuple*) – Center of symmetry in Cartesian coordinates. If None center is optimized
- **permutation** (*list, tuple*) – Define permutation
- **output** (*hook*) – Display hook

print_info()

Prints general information about the structures

print_minimum_distortion_path_shape(*shape_label1, shape_label2, central_atom=0, min_dev=None, max_dev=None, min_gco=None, max_gco=None, num_points=20, output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print the minimum distortion path

Parameters

- **shape_label1** (*str*) – First reference shape label
- **shape_label2** (*str*) – Second reference shape label
- **central_atom** (*int*) – Position of the central atom
- **min_dev** (*float*) –
- **max_dev** (*float*) –

- **`min_gco`** (*`float`*) –
- **`max_gco`** (*`float`*) –
- **`num_points`** (*`int`*) – Number of points
- **`output1`** (*`hook`*) – Display hook

`print_point_group`(*tol=0.01*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Print point group of all structures

Parameters **`tol`** (*`float`*) – Tolerance

`print_shape_measure`(*shape_reference*, *central_atom=0*, *fix_permutation=False*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Prints the shape measure of all structures in format

Parameters

- **`shape_reference`** (*`list`*) – List of references and/or geometries
- **`central_atom`** (*`int`*) – Position of the central atom
- **`fix_permutation`** (*`bool`*) – Do not permute atoms during shape calculations
- **`output`** (*`hook`*) – Display hook

`print_shape_structure`(*shape_reference*, *central_atom=0*, *fix_permutation=False*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Prints the nearest shape structure in format

Parameters

- **`shape_reference`** (*`list`*) – List of references and/or geometries
- **`central_atom`** (*`int`*) – Position of the central atom
- **`fix_permutation`** (*`bool`*) – Do not permute atoms during shape calculations
- **`output`** (*`hook`*) – Display hook

`print_symmetry_nearest_structure`(*label*, *central_atom=0*, *center=None*, *permutation=None*, *output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Prints the nearest structure to ideal symmetric structure

Parameters

- **`label`** (*`str`*) – Symmetry point group label
- **`central_atom`** (*`int`*) – Position of the central atom
- **`center`** (*`int`*) – Center of symmetry in Cartesian coordinates. If None center is optimized
- **`permutation`** (*`list, tuple`*) – Define permutation
- **`output`** (*`hook`*) – Display hook

6.2 Molecule

class `cosymlib.molecule.`**`Molecule`**(*geometry*, *electronic_structure=None*, *name=None*)

This is the main class that contains all the information and calculations methods that can a apply to a single molecule. The functionality is divided in two objects: Geometry and Electronic Structure. In the base class (Molecule) implements the methods thatrequire both the electronic structure and molecular geometry information such as the symmetry of the wave function.

Parameters

- **`geometry`** (`Geometry, Molecule`) – The geometry
- **`electronic_structure`** (`ElectronicStructure, str`) – The electronic structure
- **`name`** (`str`) – Molecule name

`electronic_structure`

Get the electronic structure

Returns The electronic structure

Return type *ElectronicStructure*

`geometry`

Get the geometry

Returns The geometry

Return type *Geometry*

`get_charge`()

Get the charge of the molecule

`get_connectivity`()

Get the atoms connectivity

Returns the atoms connectivity

Return type list

`get_n_atoms`()

Get the number of atoms

Returns number of atoms

Return type int

`get_pointgroup`(*tol=0.01*)

Get the symmetry point group

Parameters **`tol`** (`float`) – The tolerance

Returns The point group label

Return type str

`get_positions`()

Get the positions in Cartesian coordinates

Returns the coordinates

Return type list

`get_symbols`()

Get the atomic elements symbols

Returns the symbols

Return type list

6.3 Geometry

class `cosymlib.molecule.geometry.`**`Geometry`**(*positions*, *symbols=()*, *name=''*, *connectivity=None*, *connectivity_thresh=1.2*)

This class contains the methods related to shape and geometric symmetry calculations

Parameters

- **`positions`** (*`list`*) – Cartesian coordinates
- **`symbols`** (*`list`*) – Atomic elements symbols
- **`name`** (*`str`*) – Geometry name
- **`connectivity`** (*`list`*) – Connectivity list
- **`connectivity_thresh`** (*`float`*) – Connectivity threshold

Example

```
water = Geometry(positions=[[0.0, 0.0, 0.0],
                            [0.0, 1.0, 0.0],
                            [0.0, 0.0, 1.0],
                 symbols=['O', 'H', 'H'],
                 name='water molecule',
                 connectivity=[[1, 2], [1, 3]])
```

`get_connectivity`()

Get connectivity as a list of pairs of connected atoms

Returns The connectivity

Return type list

`get_n_atoms`()

Get the number of atoms

Returns number of atoms

Return type int

`get_pointgroup`(*tol=0.01*)

Get the symmetry point group

Parameters **`tol`** (*`float`*) – The tolerance

Returns The point group

Return type str

`get_positions`()

Get the positions in Cartesian coordinates

Returns the coordinates

Return type list

`get_shape_measure`(*shape_label*, *central_atom=0*, *fix_permutation=False*)

Get the Shape measure

Parameters

- **shape_label** (*str*) – Reference shape label
- **central_atom** (*int*) – the central atom position
- **fix_permutation** (*bool*) – Do not permute atoms

Returns The measure

Return type float

get_symbols()

Get the atomic elements symbols

Returns the symbols

Return type list

get_symmetry_measure(*label*, *central_atom=0*, *center=None*, *multi=1*, *permutation=None*)

Get the symmetry measure

Parameters

- **label** (*str*) – Symmetry point group
- **central_atom** (*int*) – central atom position (0 if no central atom)
- **center** (*list*) – center of the measure in Cartesian coordinates
- **permutation** (*list, tuple*) – define permutation

Returns The symmetry measure

Return type float

get_symmetry_nearest_structure(*label*, *central_atom=0*, *center=None*, *multi=1*, *permutation=None*)

Returns the nearest ideal structure

Parameters

- **label** (*str*) – symmetry point group
- **central_atom** (*int*) – central atom position (0 if no central atom)
- **center** (*list*) – center of the measure in Cartesian coordinates
- **permutation** (*list, tuple*) – Define permutation

Returns The structure

Return type *Geometry*

get_symmetry_permutation(*label*, *central_atom=0*, *center=None*, *multi=1*, *permutation=None*)

Get the optimum atoms permutation for geometrical symmetry measures

Parameters

- **label** (*str*) – Symmetry point group
- **central_atom** (*int*) – central atom position (0 if no central atom)
- **center** (*list*) – center of the measure in Cartesian coordinates
- **permutation** (*list, tuple*) – define permutation

Returns The symmetry measure

Return type float

6.4 Electronic structure

class cosymlib.molecule.electronic_structure.**ElectronicStructure**(*basis*, *orbital_coefficients*, *multiplicity=None*, *alpha_energies=None*, *beta_energies=None*, *alpha_occupancy=None*, *beta_occupancy=None*)

This class contains basically the access to electronic structure data

Parameters

- **basis** (*dict*) – The basis set
- **orbital_coefficients** (*list*) – Molecular orbital coefficients
- **multiplicity** (*int*) – The multiplicity
- **alpha_energies** (*list*) – Alpha molecular orbital energies in Hartree
- **beta_energies** (*list*) – Beta molecular orbital energies in Hartree
- **alpha_occupancy** (*list*) – Occupancy of alpha orbitals
- **beta_occupancy** (*list*) – Occupancy of beta orbitals

alpha_electrons
get the number of alpha electrons

Returns alpha electrons

alpha_energies
get the energies of the alpha molecular orbitals

Returns the energies

basis
get the basis set name

Returns basis set

beta_electrons
get the number of beta electrons

Returns beta electrons

beta_energies
get the energies of the beta molecular orbitals

Returns the energies

coefficients_a
get the alpha molecular orbitals coefficients

Returns the alpha molecular orbitals

coefficients_b
get the beta molecular orbitals coefficients

Returns the beta molecular orbitals

multiplicity
get the multiplicity

Returns the multiplicity

s2
get the expected value of spin square operator S2 = (s * (s + 1))

Returns S2

Cosymlib is being developed by the **Electronic Structure & Symmetry (ESS)** group at the Department de Ciència de Materials i Química Física and the Institut de Química Teòrica i Computacional (IQTCUB). University of Barcelona.

Bibliography

[AVN] a) H. Zabrodsky, S. Peleg, D. Avnir, "Continuous symmetry measures", *J. Am. Chem. Soc.* (1992) **114**, 7843-7851.

b) H. Zabrodsky, S. Peleg, D. Avnir, "Continuous Symmetry Measures II: Symmetry Groups and the Tetrahedron", *J. Am. Chem. Soc.* (1993) **115**, 8278–8289.

c) H. Zabrodsky, D. Avnir, "Continuous Symmetry Measures, IV: Chirality" *J. Am. Chem. Soc.* (1995) **117**, 462–473.

d) H. Zabrodsky, S. Peleg, D. Avnir, "Symmetry as a Continuous Feature" IEEE, *Trans. Pattern. Anal. Mach. Intell.* (1995) **17**, 1154–1166.

e) M. Pinsky, D. Avnir, "Continuous Symmetry Measures, V: The Classical Polyhedra" *Inorg. Chem.* (1998) **37**, 5575–5582.

[CShM] a) M. Pinsky, D. Avnir, "Continuous Symmetry Measures, V: The Classical Polyhedra" *Inorg. Chem.* (1998) **37**, 5575–5582.

b) D. Casanova, J. Cirera, M. Llunell, P. Alemany, D. Avnir, and S. Alvarez, "Minimal Distortion Pathways in Polyhedral Rearrangements" *J. Am. Chem. Soc.* (2004) **126**, 1755–1763.

c) S.Alvarez, P. Alemany, D. Casanova, J. Cirera, M. Llunell, D. Avnir, "Shape maps and polyhedral interconversion paths in transition metal chemistry" *Coord. Chem. Rev.* (2005) **249**, 1693–1708.

d) K. M. Ok, P. S. Halasyamani, D. Casanova, M. Llunell, P. Alemany, S. Alvarez, "Distortions in Octahedrally Coordinated d^0 Transition Metal Oxides: A Continuous Symmetry Measures Approach" *Chem. Mater.* (2006) **18**, 3176–3183.

e) A. Carreras, E. Bernuz, X. Marugan, M. Llunell, P. Alemany, "Effects of Temperature on the Shape and Symmetry of Molecules and Solids" *Chem. Eur. J.* (2019) **25**, 673 – 691.

[CSM] a) H. Zabrodsky, S. Peleg, D. Avnir, "Continuous symmetry measures" *J. Am. Chem. Soc.* (1992) **114**, 7843-7851.

b) Y. Salomon, D. Avnir, "Continuous symmetry measures: A note in proof of the folding/unfolding method" *J. Math. Chem.* (1999) **25**, 295–308.

c) M. Pinsky, D. Casanova, P. Alemany, S. Alvarez, D. Avnir, C. Dryzun, Z. Kizner, A. Sterkin, "Symmetry operation measures" *J. Comput. Chem.* (2008) **29**, 190–197.

d) M. Pinsky, C. Dryzun, D. Casanova, P. Alemany, D. Avnir, "Analytical methods for calculating Continuous Symmetry Measures and the Chirality Measure" *J. Comput. Chem.* (2008) **29**, 2712–2721.

e) C. Dryzun, A. Zait, D. Avnir, "Quantitative symmetry and chirality—A fast computational algorithm for large structures: Proteins, macromolecules, nanotubes, and unit cells" *J. Comput. Chem.* (2011) **32**, 2526–2538

f) M. Pinsky, A. Zait, M. Bonjack, D. Avnir, "Continuous symmetry analyses: C_{nv} and D_n measures of molecules, complexes, and proteins" *J. Comput. Chem.* (2013) **34**, 2–9.

g) C. Dryzun, "Continuous symmetry measures for complex symmetry group" *J. Comput. Chem.* (2014) **35**, 748–755.

h) G.Alon, I. Tuvi-Arad, "Improved algorithms for symmetry analysis: structure preserving permutations" *J. Math. Chem.* (2018) **56**, 193–212.

[CCM] a) H. Zabrodsky, D. Avnir, "Continuous Symmetry Measures, IV: Chirality" *J. Am. Chem. Soc.* (1995) **117**, 462–473.

b) M. Pinsky, C. Dryzun, D. Casanova, P. Alemany, D. Avnir, "Analytical methods for calculating Continuous Symmetry Measures and the Chirality Measure" *J. Comput. Chem.* (2008) **29**, 2712–2721.

c) C. Dryzun, A. Zait, D. Avnir, "Quantitative symmetry and chirality — A fast computational algorithm for large structures: Proteins, macromolecules, nanotubes, and unit cells" *J. Comput. Chem.* (2011) **32**, 2526–2538

[QCSMs] a) C. Dryzun, D. Avnir, "Generalization of the Continuous Symmetry Measure: The Symmetry of Vectors, Matrices, Operators and Functions" *Phys. Chem. Chem. Phys.* (2009) **11**, 9653–9666.

b) C. Dryzun, D. Avnir, "Chirality Measures for Vectors, Matrices, Operators and Functions" *ChemPhysChem* (2011) **12**, 197–205.

c) P. Alemany, "Analyzing the Electronic Structure of Molecules Using Continuous Symmetry Measures" *Int. J. Quantum Chem.* (2013) **113**, 1814–1820;

d) P. Alemany, D. Casanova, S. Alvarez, C. Dryzun, D. Avnir, "Continuous Symmetry Measures: a New Tool in Quantum Chemistry" *Rev. Comput. Chem.* (2017) **30**, 289–352.

Python Module Index

Index