
pychemcurv

Release 2022.3.17

Germain Salvato Vallverdu, Julia Sabalot-cuzzubbo, Dider Bégué,

Feb 29, 2024

INTRODUCTION

1	Features	3
2	Citing pychemcurv	5
3	Installation	7
3.1	Installation from PyPi	7
3.2	Installation from source	7
3.3	Install in developer mode	8
4	Run the web application	9
4.1	Common error on local execution	9
5	Licence and contact	11
6	Core classes	13
6.1	Vertex classes	13
6.2	POAV: Pi-Orbital Axis Vector	17
7	pychemcurv.analysis	21
7.1	CurvatureAnalyzer class	21
8	pychemcurv.vis	23
8.1	CurvatureViewer class	24
9	pychemcurv.geometry	27
10	Introduction	29
10.1	Features	29
10.2	Citing pychemcurv	30
10.3	Installation	30
10.4	Run the web application	31
10.5	Licence and contact	32
	Bibliography	33
	Python Module Index	35
	Index	37

Table of contents

- *Introduction*
 - *Features*
 - *Citing pychemcurv*
 - *Installation*
 - * *Installation from PyPi*
 - * *Installation from source*
 - * *Install in developer mode*
 - *Run the web application*
 - * *Common error on local execution*
 - *Licence and contact*

pychemcurv is a python package for structural analyzes of molecular systems or solid state materials focusing on the local curvature at an atomic scale. The local curvature is then used to compute the hybridization of molecular orbitals.

The main features of the library are available from a [Plotly/Dash](https://pychemcurv.onrender.com/) web application available here: pychemcurv.onrender.com/. The web-app allows to upload simple xyz files and compute the local geometrical properties and the hybridization properties. The application source code is available in a separate repository at [pychemcurv-app](#).

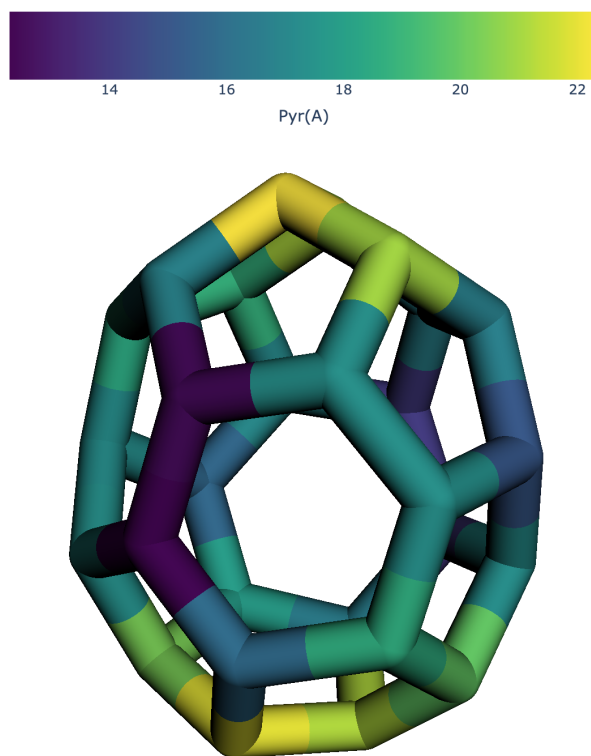


Fig. 1: Pyramidalization angle of a C_{28} fullerene mapped on the structure with a colorscale.

FEATURES

Pychemcurv is divided in two parts. The first one is a standard python package which provides two main classes to compute the local curvature at the atomic scale and the hybridization of a given atom. Second, a [Plotly/Dash](#) web application is provided in order to perform a geometrical and electronic analyzes on molecules or materials.

The web application is available at pychemcurv.onrender.com/. The web-app allows to upload simple xyz files and compute the local geometrical properties and the hybridization properties. The application source code is available in a separate repository at [pychemcurv-app](#).

Some jupyter notebooks are provided in the `notebooks/` folder and present use cases of the classes implemented in this package. You can access to these notebooks online with [binder](#).

CITING PYCHEMCURV

Please, consider to cite the following papers when using either the *pychemcurv* library or the web application.

Julia Sabalot-Cuzzubbo, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Relating the molecular topology and local geometry: Haddon's pyramidalization angle and the Gaussian curvature*, J. Chem. Phys. **152**, 244310 (2020).

Julia Sabalot-Cuzzubbo, N. Cresson, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Haddon's POAV2 vs POAV theory for non-planar molecules*, J. Chem. Phys. **159**, 174109 (2023).

INSTALLATION

3.1 Installation from PyPi

From November 2023, pychemcurv is available on pypi. You can install it directly using pip.

```
python -m pip install pychemcurv
```

3.2 Installation from source

Before installing pychemcurv it is recommended to create a virtual environment using conda or virtuelenv.

In this environment, using pip directly from the github repository, run

```
pip install git+git://github.com/gVallverdu/pychemcurv.git
```

Alternatively, you can first clone the pychemcurv repository

```
git clone https://github.com/gVallverdu/pychemcurv.git
```

and then install the module and its dependencies using

```
pip install .
```

If you want to use the web application locally or if you want to use [nglview](#) to display structures in jupyter notebooks you need to install more dependencies. The setup configuration provides the viz extra so, using pip, run one of

```
pip install .[viz]
```

```
# escape square bracket with zsh  
pip install .\[viz\]
```

If you have installed nglview you may have to enable the jupyter extension

```
jupyter-nbextension enable nglview --py --sys-prefix
```

3.3 Install in developer mode

In order to install in developer mode, first create an environment (using one of the provided file for example) and then install using pip

```
pip install -e .[viz]
```

If you want to build the documentation you also need to install sphinx. A dedicated requirements file is provided in the docs/ folder.

RUN THE WEB APPLICATION

The web application is available in this separate repository: [pychemcurv-app](https://github.com/gVallverdu/pychemcurv-app) <https://github.com/gVallverdu/pychemcurv-app>. The main aim of the application is to use the pychemcurv package and visualize the geometrical or chemical atomic quantities mapped on the chemical structure of your system.

The application is available online at this address: pychemcurv.onrender.com/.

Demo video:

In order to run the application locally, you have to clone the repository and install all the dependencies. In particular dash and dash-bio. You can do that from the `requirements.txt` provided in the repository of the application. Here is a short procedure in order to install and run the application locally. It assumes pychemcurv is already installed in a python environment called (curv):

```
[user@computer] (curv) > $ git clone https://github.com/gVallverdu/pychemcurv-app.git
[user@computer] (curv) > $ cd pychemcurv-app/
[user@computer] (curv) > $ python -m pip install -r requirements.txt
[user@computer] (curv) > $ python app.py
Running on http://127.0.0.1:8050/
Debugger PIN: 065-022-191
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
```

Open the provided url to use the application.

You can switch off/on the debug mode by setting `debug=False` on the last line of the `app.py` file.

4.1 Common error on local execution

If the application does not start with an error such as:

```
socket.gaierror: [Errno 8] nodename nor servname provided, or not known
```

Go to the last lines of the file `app.py` and comment/uncomment the last lines to get something that reads

```
if __name__ == '__main__':
    app.run_server(debug=True, host='127.0.0.1')
    # app.run_server(debug=False)
```


LICENCE AND CONTACT

This software was developed at the Université de Pau et des Pays de l'Adour (UPPA) in the Institut des Sciences Analytiques et de Physico-Chimie pour l'Environnement et les Matériaux (IPREM) and the Institut Pluridisciplinaire de Recherches Appliquées (IPRA) and is distributed under the MIT licence.

Authors

- Germain Salvato Vallverdu: germain.vallverdu@univ-pau.fr
- Julia Sabalot-cuzzubbo julia.sabalot@univ-pau.fr
- Didier Bégue: didier.begue@univ-pau.fr
- Jacky Cresson: jacky.cresson@univ-pau.fr



CORE CLASSES

Module `pychemcur.core` implements several classes in order to represents a vertex of a molecular squeleton and compute geometrical and chemical indicators related to the local curvature around this vertex.

A complete and precise definition of all the quantities computed in the classes of this module can be found in article [JCP2020].

6.1 Vertex classes

6.1.1 VertexAtom class

class `pychemcurv.core.VertexAtom(a, star_a)`

This class represents an atom (or a point) associated to a vertex of the squeleton of a molecule. The used notations are the following. We denote by A a given atom characterized by its cartesian coordinates corresponding to a vector in \mathbb{R}^3 . This atom A is bonded to one or several atoms B . The atoms B , bonded to atoms A belong to $\star(A)$ and are characterized by their cartesian coordinates defined as vectors in \mathbb{R}^3 . The geometrical object obtained by drawing a segment between bonded atoms is called the skeleton of the molecule and is the initial geometrical picture for a molecule. This class is defined from the cartesian coordinates of atom A and the atoms belonging to $\star(A)$.

More generally, the classes only considers points in \mathbb{R}^3 . The is not any chemical consideration here. In consequence, the class can be used for all cases where a set of point in \mathbb{R}^3 is relevant.

Parameters

- **a** (`np.ndarray`) – cartesian coordinates of point/atom A in \mathbb{R}^3
- **star_a** (`nd.array`) – ($N \times 3$) cartesian coordinates of points/atoms B in $\star(A)$

static from_pyramid(*length, theta, n_star_A=3, radians=False, perturb=None*)

Set up a `VertexAtom` from an ideal pyramidal structure. Build an ideal pyramidal geometry given the angle θ and randomize the positions by adding a noise of a given magnitude. The vertex of the pyramid is the point A and $\star(A)$. are the points linked to the vertex. The size of $\star(A)$. is at least 3.

θ is the angle between the normal vector of the plane defined from $\star(A)$ and the bonds between A and $\star(A)$. The pyramidalisation angle is defined from θ such as

$$pyrA = \theta - \frac{\pi}{2}$$

Parameters

- **length** (`float`) – the bond length
- **theta** (`float`) – Angle to define the pyramid

- **n_star_A** (*int*) – number of point bonded to A the vertex of the pyramid.
- **radian** (*bool*) – True if theta is in radian (default False)
- **perturb** (*float*) – Give the width of a normal distribution from which random numbers are choosen and added to the coordinates.

Returns

A VertexAtom instance

property a

Coordinates of atom A

property star_a

Coordinates of atoms B belonging to $\star(A)$

property reg_star_a

Regularized coordinates of atoms/points B in $\star(A)$ such as all distances between A and points B are equal to unity. This corresponds to $Reg_{\epsilon} \star(A)$ with $\epsilon = 1$.

property normal

Unitary vector normal to the plane or the best fitting plane of atoms/points B_i in $\star(A)$.

property reg_normal

Unitary vector normal to the plane or the best fitting plane of atoms/points $RegB_i$ in $\star(A)$.

property com

Center of mass of atoms/points B in $\star(A)$

property distances

Return all distances between atom A and atoms B belonging to $\star(A)$. Distances are in the same order as the atoms in `vertex.star_a`.

get_angles(radians=True)

Compute angles θ_{ij} between the bonds AB_i and AB_j , atoms B_i and B_j belonging to $\star(A)$. The angle θ_{ij} is made by the vectors AB_i and AB_j in the affine plane defined by this two vectors and atom A. The computed angles are such as bond AB_i are in a consecutive order.

Parameters

radians (*bool*) – if True (default) angles are returned in radians

property angular_defect

Compute the angular defect in radians as a measure of the discrete curvature around the vertex, point A.

The calculation first looks for the best fitting plane of points belonging to $\star(A)$ and sorts that points in order to compute the angles between the edges connected to the vertex (A). See the `get_angles` method.

property pyr_distance

Compute the distance of atom A to the plane define by $\star(A)$ or the best fitting plane of $\star(A)$. The unit of the distance is the same as the unit of the coordinates of A and $\star(A)$.

as_dict(radians=True, list_obj=False)

Return a dict version of all the properties that can be computed using this class. Use `list_obj=True` to get a valid JSON object.

Parameters

- **radians** (*bool*) – if True, angles are returned in radians (default)
- **list_obj** (*bool*) – if True, numpy arrays are converted into list object (default False)

Returns

A dict

write_file(*species*='C', *filename*='vertex.xyz')

Write the coordinates of atom A and atoms $\star(A)$ in a file in xyz format. You can set the name of species or a list but the length of the list must be equal to the number of atoms. If filename is None, returns the string corresponding to the xyz file.

Parameters

- **species** (*str*, *list*) – name of the species or list of the species names
- **filename** (*str*) – path of the output file or None to get a string

Returns

None if filename is a path, else, the string corresponding to the xyz file.

6.1.2 TrivalentVertex class

class pychemcurv.core.TrivalentVertex(*a*, *star_a*)

This object represents an atom (or a point) associated to a vertex of the skeleton of a molecule bonded to exactly 3 other atoms (or linked to 3 other points). This correspond to the trivalent case.

We denote by A a given atom characterized by its cartesian coordinates corresponding to a vector in \mathbb{R}^3 . This atom A is bonded to 3 atoms B. The atoms B, bonded to atom A belong to $\star(A)$ and are characterized by their cartesian coordinates defined as vectors in \mathbb{R}^3 . The geometrical object obtained by drawing a segment between bonded atoms is called the skeleton of the molecule and is the initial geometrical picture for a molecule. This class is defined from the cartesian coordinates of atom A and the atoms belonging to $\star(A)$.

More generally, the classes only considers points in \mathbb{R}^3 . The is not any chemical consideration here. In consequence, the class can be used for all cases where a set of point in \mathbb{R}^3 is relevant.

The following quantities are computed according the reference [JCP2020]

pyramidalization angle, pyrA

The pyramidalization angle, **in degrees**. $\text{pyrA} = \theta - \pi/2$ where θ is the angle between the normal vector of the plane containing the atoms B of $\star(A)$ and a vector along a bond between atom A and one B atom.

An exact definition of pyrA needs that A is bonded to exactly 3 atoms in order to be able to define a unique plane that contains the atoms B belonging to $\star(A)$. Nevertheless, pyrA is computed if more than 3 atoms are bonded to atom A by computing the best fitting plane of atoms belonging to $\star(A)$.

pyramidalization angle, pyrA_r

The pyramidalization angle **in radians**.

improper angle, improper

The improper angle corresponding to the dihedral angle between the planes defined by atoms (i, j, k) and (j, k, l), atom i being atom A and atoms j, k and l being atoms of $\star(A)$. In consequence, the improper angle is defined only if there are 3 atoms in $\star(A)$.

The value of the improper angle is returned in radians.

angular defect, angular_defect

The angular defect is defined as

where α_F are the angles at the vertex A of the faces $F \in \star(A)$. The angular defect is computed whatever the number of atoms in $\star(A)$.

The value of the angular defect is returned in radians.

spherical curvature, spherical_curvature

The spherical curvature is computed as the radius of the osculating sphere of atoms A and atoms belonging to $\star(A)$. The spherical curvature is computed as

$$\kappa(A) = \frac{1}{\sqrt{\ell^2 + \frac{(OA^2 - \ell^2)^2}{4z_A^2}}}$$

where O is the center of the circumscribed circle of atoms in $\star(A)$; A the vertex atom ; OA the distance between O and A ; ℓ the distance between O and atoms B of $\star(A)$; z_A the distance of atom A to the plane defined by $\star(A)$. The spherical curvature is defined only if there are 3 atoms in $\star(A)$.

pyramidalization distance pyr_distance

Distance of atom A to the plane define by $\star(A)$ or the best fitting plane of $\star(A)$.

The value of the distance is in the same unit as the coordinates.

If the number of atoms B in $\star(A)$ is not suitable to compute some properties, *np.nan* is returned.

Note that the plane defined by atoms B belonging to $\star(A)$ is exactly defined *only* in the case where there are three atoms B in $\star(A)$. In the case of pyrA, if there are more than 3 atoms in $\star(A)$, the class use the best fitting plane considering all atoms in $\star(A)$ and compute the geometrical quantities.

Parameters

- **a** (*np.ndarray*) – cartesian coordinates of point/atom A in \mathbb{R}^3
- **star_a** (*nd.array*) – (N x 3) cartesian coordinates of points/atoms B in $\star(A)$

static from_pyramid(length, theta, radians=False, perturb=None)

Set up a VertexAtom from an ideal pyramidal structure. Build an ideal pyramidal geometry given the angle theta and randomize the positions by adding a noise of a given magnitude. The vertex of the pyramid is the point A and $\star(A)$. are the points linked to the vertex. The size of $\star(A)$. is 3.

θ is the angle between the normal vector of the plane defined from $\star(A)$ and the bonds between A and $\star(A)$. The pyramidalisation angle is defined from θ such as

$$\text{pyrA} = \theta - \frac{\pi}{2}$$

Parameters

- **length** (*float*) – the bond length
- **theta** (*float*) – Angle to define the pyramid
- **radian** (*bool*) – True if theta is in radian (default False)
- **perturb** (*float*) – Give the width of a normal distribution from which random numbers are choosen and added to the coordinates.

Returns

A TrivalentVertex instance

property improper

Compute the improper angle in randians between planes defined by atoms (i, j, k) and (j, k, l). Atom A, is atom i and atoms j, k and l belong to $\star(A)$.



This quantity is available only if the length of $\star(A)$ is equal to 3.

property `pyrA_r`

Return the pyramidalization angle in radians.

property `pyrA`

Return the pyramidalization angle in degrees.

property `spherical_curvature`

Compute the spherical curvature associated to the osculating sphere of points A and points B belonging to $\star(A)$. Here, we assume that there is exactly 3 atoms B in $\star(A)$.

as_dict(*radians=True, list_obj=False*)

Return a dict version of all the properties that can be computed using this class. Use *list_obj=True* to get a valid JSON object.

Parameters

- **radians** (*bool*) – if True, angles are returned in radians (default)
- **list_obj** (*bool*) – if True, numpy arrays are converted into list object (default False)

Returns

A dict.

6.2 POAV: Pi-Orbital Axis Vector

POAV stands for π -Orbital Axis Vector. The definition of this vector has its origin in the works of R.C. Haddon. The definitions and the relation between POAV and the local curvature of a molecule using new geometrical object such as the angular defect have been established in our recent work [JCP2020]. An discussion on the two POAV1 and POAV2 quantities is published in the work [POAV2].

Hereafter, the two classes POAV1 and POAV2 aim to compute quantities related to the two definitions of the POAV vector.

6.2.1 POAV1

class `pychemcurv.core.POAV1`(*vertex*)

In the case of the POAV1 theory the POAV vector has the property to make a constant angle with each bond connected to atom A.

This class computes indicators related to the POAV1 theory of R.C. Haddon following the link established between `pyrA` and the hybridization of a trivalent atom in reference [JCP2020].

A chemical picture of the hybridization can be drawn by considering the contribution of the p atomic orbitals to the system σ , or the contribution of the s atomic orbital to the system π . This is achieved using the `m` and `n` quantities. For consistency with POAV2 class, the attributes, `hybridization`, `sigma_hyb_nbr` and `pi_hyb_nbr` are also implemented but return the same values.

POAV1 is defined from the local geometry of an atom at a vertex of the molecule's skeleton.

Parameters

vertex (`TrivalentVertex`) – the trivalent vertex atom

property `pyrA`

Pyramidalization angle in degrees

property pyrA_r

Pyramidalization angle in radians

property poav

Return a unitary vector along the POAV vector

property c_pi

Value of c_π in the ideal case of a C_{3v} geometry. Equation (22), with $c_{1,2} = \sqrt{2/3}$.

$$c_\pi = \sqrt{2} \tan Pyr(A)$$

property lambda_pi

value of λ_π in the ideal case of a C_{3v} geometry. Equation (23), with $c_{1,2}^2 = 2/3$.

$$\lambda_\pi = \sqrt{1 - 2 \tan^2 Pyr(A)}$$

property m

value of hybridization number m, see equation (44)

$$m = \left(\frac{c_\pi}{\lambda_\pi} \right)^2$$

property n

value of hybridization number n, see equation (47)

$$n = 3m + 2$$

property pi_hyb_nbr

This quantity measure the weight of the s atomic orbital with respect to the p atomic orbital in the h_π hybrid orbital along the POAV vector.

This is equal to m.

property sigma_hyb_nbr

This quantity measure the weight of the p atomic orbitals with respect to s in the h_i hybrid orbitals along the bonds with atom A.

This is equal to n

property hybridization

Compute the hybridization such as

$$sp^{(2+c_\pi^2)/(1-c_\pi^2)}$$

This quantity corresponds to the amount of p AO in the system σ . This is equal to n and corresponds to the \tilde{n} value defined by Haddon.

TODO: verifier si cette quantité est égale à n uniquement dans le cas C_{3v} .

as_dict(radians=True, include_vertex=False, list_obj=False)

Return a dict version of all the properties that can be computed with this class. Note that in the case of λ_π and c_π the squared values are returned as they are more meaningful. Use `list_obj= True` to obtain a valid JSON object.

Parameters

- **radians** (*bool*) – if True, angles are returned in radians (default)
- **include_vertex** (*bool*) – if True, include also vertex data
- **list_obj** (*bool*) – if True, numpy arrays are converted into list object (default False)

Returns

A dict.

6.2.2 POAV2

class pychemcurv.core.POAV2(*vertex*)

In the case of the POAV2 theory the POAV2 vector on atom A is such as the set of hybrid molecular orbitals h_π, h_1, h_2, h_3 is orthogonal; where the orbitals h_i are hybrid orbitals along the bonds with atoms linked to atom A and h_π is the orbital along the POAV2 \vec{u}_π vector.

This class computes indicators related to the POAV2 theory of R.C. Haddon following the demonstrations in the reference [POAV2].

POAV1 is defined from the local geometry of an atom at a vertex of the molecule's skeleton.

Parameters

vertex (*TrivalentVertex*) – the trivalent vertex atom

property matrix

Compute and return the sigma-orbital hybridization numbers n1, n2 and n3

property u_pi

Return vector u_π as the basis of the zero space of the matrix M. This unitary vector support the POAV2 vector.

property sigma_hyb_nbrs

Compute and return the sigma-orbital hybridization numbers n1, n2 and n3. These quantities measure the weight of the p atomic orbitals with respect to s in each of the h_i hybrid orbitals along the bonds with atom A.

property pi_hyb_nbr

This quantity measure the weight of the s atomic orbital with respect to the p atomic orbital in the h_π hybrid orbital along the POAV2 vector.

property pyrA_r

Compute the angles between vector u_π and all the bonds between atom A and atoms B in $\star(A)$.

as_dict (*radians=True, include_vertex=False, list_obj=False*)

Return a dict version of all the properties that can be computed with this class. Use *list_obj= True* to obtain a valid JSON object.

Parameters

- **radians** (*bool*) – if True, angles are returned in radians (default)
- **include_vertex** (*bool*) – if True, include also vertex data
- **list_obj** (*bool*) – if True, numpy arrays are converted into list object (default False)

Returns

A dict.

PYCHEMCURV.ANALYSIS

This module implements the *CurvatureAnalyze* class to perform curvature analyses on molecular or periodic structures.

7.1 CurvatureAnalyzer class

class pychemcurv.analysis.**CurvatureAnalyzer**(*structure*, *bond_tol*=0.2, *rcut*=2.5, *bond_order*=None)

This class provides helpful methods to analyze the local curvature on all atoms of a given structure. The structure is either a molecule or a periodic structure. Once the structure is read, the class determines the connectivity of the structure in order to define all vertices. The connectivity is defined on a distance criterion.

The class needs a *pymatgen.Structure* or *pymatgen.Molecule* object as first argument. The other arguments are used to defined if two atoms are bonded or not.

Parameters

- **structure** (*Structure*, *Molecule*) – A *Structure* or *Molecule* *pymatgen* objects
- **bond_tol** (*float*) – Tolerance used to determine if two atoms are bonded. Look at *pymatgen.core.CovalentBond.is_bonded*.
- **rcut** (*float*) – Cutoff distance in case the bond is not not known
- **bond_order** (*dict*) – Not yet implemented

property vertices

List of vertices associated to each atom of the molecule

property bonds

Set of tuples of bonded atom index

property vertices_idx

List of tuples of the indexes of atoms in each vetex. The first index is atom A, the following are atoms of $\star(A)$.

property data

Return a Data Frame that contains all the geometric and hybridization data.

property distance_matrix

Returns the distance matrix between all atoms. For periodic structures, this returns the nearest image distances.

static from_file(*path*, *periodic*=None)

Returns a *CurvatureAnalyze* object from the structure at the given path. This method relies on the file format supported with *pymatgen Molecule* and *Structure* classes.

Supported formats for periodic structure include CIF, POSCAR/CONTCAR, CHGCAR, LOCPOT, vasprun.xml, CSSR, Netcdf and pymatgen's JSON serialized structures.

Supported formats for molecule include xyz, gaussian input (gjf|g03|g09|com|inp), Gaussian output (.out) and pymatgen's JSON serialized molecules.

Parameters

- **path** (*str*) – Path to the structure file
- **periodic** (*bool*) – if True, assume that the file correspond to a periodic structure. Default is None. The method tries to read the file, first from the Molecule class and second from the Structure class of pymatgen.

get_molecular_data()

Set up a model data dictionary that contains species, coordinates and bonds of the structure. This dictionary can be used as model data for further visulization in bio-dash.

PYCHEMCURV.VIS

The `pychemcurv.vis` module implements the `CurvatureViewer` class in order to visualize a molecule or a periodic structure in a jupyter notebook and map a given properties on the atoms using a color scale.

This class needs, `nglview` and uses `ipywidgets` in a jupyter notebook to display the visualization. Run the following instructions to install `nglview` and achieve the configuration in order to be able to use `nglview` in a jupyter notebook

```
conda install nglview -c conda-forge
jupyter-nbextension enable nglview --py --sys-prefix
```

or

```
pip install nglview
jupyter-nbextension enable nglview --py --sys-prefix
```

```
Entrée [18]: 1 hbox = cv.map_view("pyrA", minval=0, radius=0.2)
              2 hbox
```

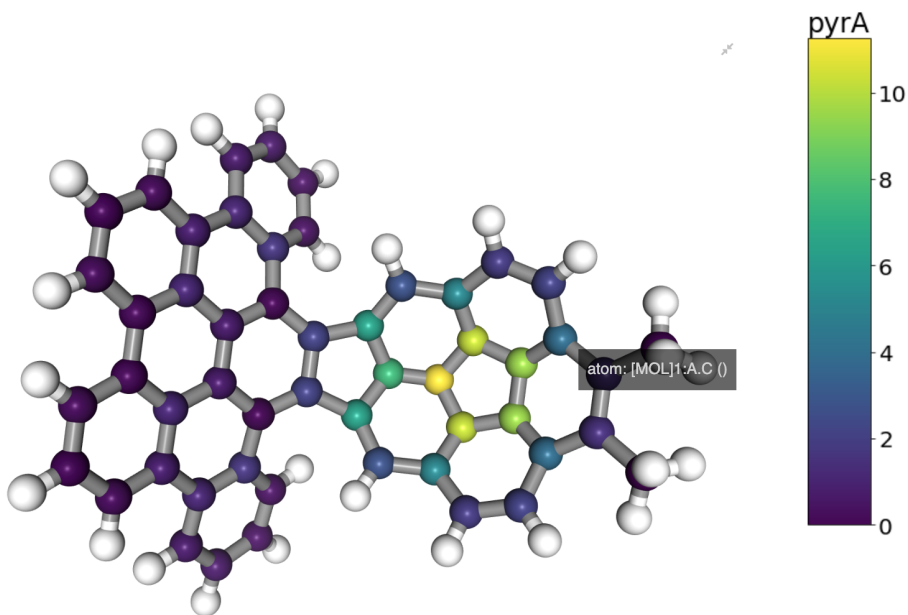


Fig. 1: Visualization of the pyramidalization angle using a color scale.

8.1 CurvatureViewer class

class pychemcurv.vis.**CurvatureViewer**(*structure*, *bond_tol*=0.2, *rcut*=2.5, *bond_order*=None)

This class provides a constructor for a NGLView widget in order to visualize the wanted properties using a color scale mapped on the 3D structure of the molecule or the structure.

The class needs a `pymatgen.Structure` or `pymatgen.Molecule` object as first argument. The other arguments are used to defined if two atoms are bonded or not.

Parameters

- **structure** (*Structure*, *Molecule*) – A Structure or Molecule pymatgen objects
- **bond_tol** (*float*) – Tolerance used to determine if two atoms are bonded. Look at *pymatgen.core.CovalentBond.is_bonded*.
- **rcut** (*float*) – Cutoff distance in case the bond is not not known
- **bond_order** (*dict*) – Not yet implemented

get_view(*representation*='ball+stick', *radius*=0.25, *aspect_ratio*=2, *unitcell*=False, *width*='700px', *height*='500px')

Set up a simple NGLView widget with the ball and stick or licorice representation of the structure.

Parameters

- **representation** (*str*) – representation: 'ball+stick' or 'licorice'
- **radius** (*float*) – bond (stick) radius
- **aspect_ratio** (*float*) – ratio between the balls and stick radiuses
- **unitcell** (*bool*) – If True and structure is periodic, show the unitcell.
- **width** (*str*) – width of the nglview widget, default '700px'
- **height** (*str*) – height of the nglview widget, default '500px'

Returns

Return a NGLWidget object

map_view(*prop*, *radius*=0.25, *aspect_ratio*=2, *unitcell*=False, *cm*='viridis', *minval*=None, *maxval*=None, *orientation*='vertical', *label*=None, *width*='700px', *height*='500px')

Map the given properties on a color scale on to the molecule using a ball and stick representations. The properties can be either the name of a column of the data computed using the CurvatureAnalyzer class, or, an array of values of a custom property. In the last case, the size of the array must be consistent with the number of atoms in the system.

Parameters

- **prop** (*str* or *array*) – name of the properties or values you want to map
- **radius** (*float*) – bond (stick) radius
- **aspect_ratio** (*float*) – ratio between the balls and stick radiuses
- **unitcell** (*bool*) – If True and structure is periodic, show the unitcell.
- **cm** (*str*) – colormap from `matplotlib.cm`.
- **minval** (*float*) – minimum value to consider for the color scale
- **maxval** (*float*) – maximum value to consider for the color scale
- **orientation** (*str*) – orientation of the colorbar 'horizontal' or 'vertical'

- **label** (*str*) – Name of the colorbar. If None, use prop.
- **width** (*str*) – width of the nglview widget, default '700px'
- **height** (*str*) – height of the nglview widget, default '500px'

Returns

Returns an ipywidgets `HBox` or `VBox` with the `NGLWidget` and a color bar associated to the mapped properties. The `NGLWidget` is the first element of the children, the colorbar is the second one.

PYCHEMCURV.GEOMETRY

This module implements utility functions to compute several geometric properties.

`pychemcurv.geometry.center_of_mass(coords, masses=None)`

Compute the center of mass of the points at coordinates *coords* with masses *masses*.

Parameters

- **coords** (*np.ndarray*) – (N, 3) matrix of the points in \mathbb{R}^3
- **masses** (*np.ndarray*) – vector of length N with the masses

Returns

The center of mass as a vector in \mathbb{R}^3

`pychemcurv.geometry.circum_center(coords)`

Compute the coordinates of the center of the circumscribed circle from three points A, B and C in \mathbb{R}^3 .

Parameters

coords (*ndarray*) – (3x3) cartesian coordinates of points A, B and C.

Returns

The coordinates of the center of the circumscribed circle

`pychemcurv.geometry.get_plane(coords, masses=None)`

Given a set of N points in \mathbb{R}^3 , compute an orthonormal basis of vectors, the first two belonging to the plane and the third one being normal to the plane. In the particular case where N equal 3, there is an exact definition of the plane as the three points define a unique plane.

If N = 3, use a gram-schmidt orthonormalization to compute the vectors. If N > 3, the orthonormal basis is obtained from SVD.

Parameters

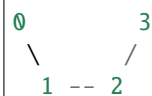
- **coords** (*np.ndarray*) – (N, 3) matrix of the points in \mathbb{R}^3
- **masses** (*np.ndarray*) – vector of length N with the masses

Returns

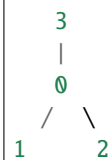
Returns the orthonormal basis (vecx, vecy, n_a), vector n_a being normal to the plane.

`pychemcurv.geometry.get_dihedral(coords)`

Compute the improper angle in radians between planes defined by points (0, 1, 2) and (1, 2, 3). The returned angle is a dihedral angle if the points 0, 1, 2 and 3 form a chain of bonded atoms in this order.



The returned angle is an improper angle if point 0 is at the center and linked to other points.

**Parameters**

coords (*ndarray*) – numpy array of the cartesian coordinates with shape (4, 3)

Returns

The dihedral angle value in radians.

INTRODUCTION

Table of contents

- *Introduction*
 - *Features*
 - *Citing pychemcurv*
 - *Installation*
 - * *Installation from PyPi*
 - * *Installation from source*
 - * *Install in developer mode*
 - *Run the web application*
 - * *Common error on local execution*
 - *Licence and contact*

pychemcurv is a python package for structural analyzes of molecular systems or solid state materials focusing on the local curvature at an atomic scale. The local curvature is then used to compute the hybridization of molecular orbitals.

The main features of the library are available from a [Plotly/Dash](#) web application available here: pychemcurv.onrender.com/. The web-app allows to upload simple xyz files and compute the local geometrical properties and the hybridization properties. The application source code is available in a separate repository at [pychemcurv-app](#).

10.1 Features

Pychemcurv is divided in two parts. The first one is a standard python package which provides two main classes to compute the local curvature at the atomic scale and the hybridization of a given atom. Second, a [Plotly/Dash](#) web application is provided in order to perform a geometrical and electronic analyzes on molecules or materials.

The web application is available at pychemcurv.onrender.com/. The web-app allows to upload simple xyz files and compute the local geometrical properties and the hybridization properties. The application source code is available in a separate repository at [pychemcurv-app](#).

Some jupyter notebooks are provided in the `notebooks/` folder and present use cases of the classes implemented in this package. You can access to these notebooks online with [binder](#).

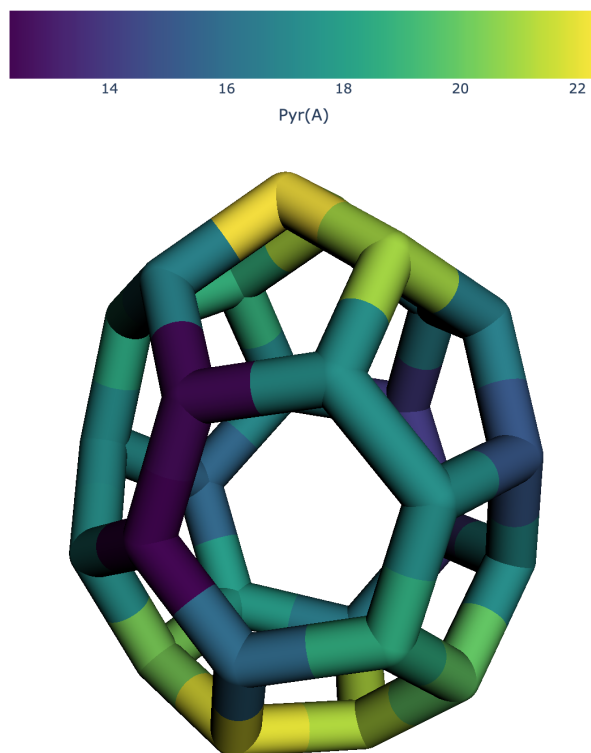


Fig. 1: Pyramidalization angle of a C_{28} fullerene mapped on the structure with a colorscale.

10.2 Citing pychemcurv

Please, consider to cite the following papers when using either the *pychemcurv* library or the web application.

Julia Sabalot-Cuzzubbo, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Relating the molecular topology and local geometry: Haddon's pyramidalization angle and the Gaussian curvature*, J. Chem. Phys. **152**, 244310 (2020).

Julia Sabalot-Cuzzubbo, N. Cresson, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Haddon's POAV2 vs POAV theory for non-planar molecules*, J. Chem. Phys. **159**, 174109 (2023).

10.3 Installation

10.3.1 Installation from PyPi

From November 2023, *pychemcurv* is available on pypi. You can install it directly using pip.

```
python -m pip install pychemcurv
```

10.3.2 Installation from source

Before installing pychemcurv it is recommended to create a virtual environment using conda or virtuelenv.

In this environment, using pip directly from the github repository, run

```
pip install git+git://github.com/gVallverdu/pychemcurv.git
```

Alternatively, you can first clone the pychemcurv repository

```
git clone https://github.com/gVallverdu/pychemcurv.git
```

and then install the module and its dependencies using

```
pip install .
```

If you want to use the web application locally or if you want to use [nglview](#) to display structures in jupyter notebooks you need to install more dependencies. The setup configuration provides the viz extra so, using pip, run one of

```
pip install .[viz]

# escape square bracket with zsh
pip install .\[viz\]
```

If you have installed nglview you may have to enable the jupyter extension

```
jupyter-nbextension enable nglview --py --sys-prefix
```

10.3.3 Install in developer mode

In order to install in developer mode, first create an environment (using one of the provided file for example) and then install using pip

```
pip install -e .[viz]
```

If you want to build the documentation you also need to install sphinx. A dedicated requirements file is provided in the docs/ folder.

10.4 Run the web application

The web application is available in this separate repository: [pychemcurv-app](https://github.com/gVallverdu/pychemcurv-app) <https://github.com/gVallverdu/pychemcurv-app>. The main aim of the application is to use the pychemcurv package and visualize the geometrical or chemical atomic quantities mapped on the chemical structure of your system.

The application is available online at this address: pychemcurv.onrender.com/.

Demo video:

In order to run the application locally, you have to clone the repository and install all the dependencies. In particular dash and dash-bio. You can do that from the `requirements.txt` provided in the repository of the application. Here is a short procedure in order to install and run the application locally. It assumes pychemcurv is already installed in a python environment called (curv):

```
[user@computer] (curv) > $ git clone https://github.com/gVallverdu/pychemcurv-app.git
[user@computer] (curv) > $ cd pychemcurv-app/
[user@computer] (curv) > $ python -m pip install -r requirements.txt
[user@computer] (curv) > $ python app.py
Running on http://127.0.0.1:8050/
Debugger PIN: 065-022-191
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
```

Open the provided url to use the application.

You can switch off/on the debug mode by setting `debug=False` on the last line of the `app.py` file.

10.4.1 Common error on local execution

If the application does not start with an error such as:

```
socket.gaierror: [Errno 8] nodename nor servname provided, or not known
```

Go to the last lines of the file `app.py` and comment/uncomment the last lines to get something that reads

```
if __name__ == '__main__':
    app.run_server(debug=True, host='127.0.0.1')
    # app.run_server(debug=False)
```

10.5 Licence and contact

This software was developed at the Université de Pau et des Pays de l'Adour (UPPA) in the Institut des Sciences Analytiques et de Physico-Chimie pour l'Environnement et les Matériaux (IPREM) and the Institut Pluridisciplinaire de Recherches Appliquées (IPRA) and is distributed under the MIT licence.

Authors

- Germain Salvato Vallverdu: germain.vallverdu@univ-pau.fr
- Julia Sabalot-cuzzubbo julia.sabalot@univ-pau.fr
- Didier Bégue: didier.begue@univ-pau.fr
- Jacky Cresson: jacky.cresson@univ-pau.fr



BIBLIOGRAPHY

- [JCP2020] Julia Sabalot-Cuzzubbo, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Relating the molecular topology and local geometry: Haddon's pyramidalization angle and the Gaussian curvature*, J. Chem. Phys. **152**, 244310 (2020). <https://aip.scitation.org/doi/10.1063/5.0008368>
- [POAV2] Julia Sabalot-Cuzzubbo, N. Cresson, Germain Salvato Vallverdu, Didier Bégué and Jacky Cresson *Haddon's POAV2 vs POAV theory for non-planar molecules*, J. Chem. Phys. **159**, 174109 (2023). <https://aip.scitation.org/doi/10.1063/5.0170800>

PYTHON MODULE INDEX

p

`pychemcurv.analysis`, [21](#)
`pychemcurv.core`, [13](#)
`pychemcurv.geometry`, [27](#)
`pychemcurv.vis`, [23](#)

A

`a` (*pychemcurv.core.VertexAtom* property), 14
`angular_defect` (*pychemcurv.core.VertexAtom* property), 14
`as_dict()` (*pychemcurv.core.POAV1* method), 18
`as_dict()` (*pychemcurv.core.POAV2* method), 19
`as_dict()` (*pychemcurv.core.TrivalentVertex* method), 17
`as_dict()` (*pychemcurv.core.VertexAtom* method), 14

B

`bonds` (*pychemcurv.analysis.CurvatureAnalyzer* property), 21

C

`c_pi` (*pychemcurv.core.POAV1* property), 18
`center_of_mass()` (in module *pychemcurv.geometry*), 27
`circum_center()` (in module *pychemcurv.geometry*), 27
`com` (*pychemcurv.core.VertexAtom* property), 14
`CurvatureAnalyzer` (class in *pychemcurv.analysis*), 21
`CurvatureViewer` (class in *pychemcurv.vis*), 24

D

`data` (*pychemcurv.analysis.CurvatureAnalyzer* property), 21
`distance_matrix` (*pychemcurv.analysis.CurvatureAnalyzer* property), 21
`distances` (*pychemcurv.core.VertexAtom* property), 14

F

`from_file()` (*pychemcurv.analysis.CurvatureAnalyzer* static method), 21
`from_pyramid()` (*pychemcurv.core.TrivalentVertex* static method), 16
`from_pyramid()` (*pychemcurv.core.VertexAtom* static method), 13

G

`get_angles()` (*pychemcurv.core.VertexAtom* method), 14

`get_dihedral()` (in module *pychemcurv.geometry*), 27
`get_molecular_data()` (*pychemcurv.analysis.CurvatureAnalyzer* method), 22
`get_plane()` (in module *pychemcurv.geometry*), 27
`get_view()` (*pychemcurv.vis.CurvatureViewer* method), 24

H

`hybridization` (*pychemcurv.core.POAV1* property), 18

I

`improper` (*pychemcurv.core.TrivalentVertex* property), 16

L

`lambda_pi` (*pychemcurv.core.POAV1* property), 18

M

`m` (*pychemcurv.core.POAV1* property), 18
`map_view()` (*pychemcurv.vis.CurvatureViewer* method), 24
`matrix` (*pychemcurv.core.POAV2* property), 19
module
 pychemcurv.analysis, 21
 pychemcurv.core, 13
 pychemcurv.geometry, 27
 pychemcurv.vis, 23

N

`n` (*pychemcurv.core.POAV1* property), 18
`normal` (*pychemcurv.core.VertexAtom* property), 14

P

`pi_hyb_nbr` (*pychemcurv.core.POAV1* property), 18
`pi_hyb_nbr` (*pychemcurv.core.POAV2* property), 19
`poav` (*pychemcurv.core.POAV1* property), 18
`POAV1` (class in *pychemcurv.core*), 17
`POAV2` (class in *pychemcurv.core*), 19
pychemcurv.analysis
 module, 21
pychemcurv.core

module, 13
pychemcurv.geometry
 module, 27
pychemcurv.vis
 module, 23
pyr_distance (pychemcurv.core.VertexAtom property),
 14
pyrA (pychemcurv.core.POAV1 property), 17
pyrA (pychemcurv.core.TrivalentVertex property), 17
pyrA_r (pychemcurv.core.POAV1 property), 17
pyrA_r (pychemcurv.core.POAV2 property), 19
pyrA_r (pychemcurv.core.TrivalentVertex property), 17

R

reg_normal (pychemcurv.core.VertexAtom property), 14
reg_star_a (pychemcurv.core.VertexAtom property), 14

S

sigma_hyb_nbr (pychemcurv.core.POAV1 property), 18
sigma_hyb_nbrs (pychemcurv.core.POAV2 property),
 19
spherical_curvature (pychem-
 curv.core.TrivalentVertex property), 17
star_a (pychemcurv.core.VertexAtom property), 14

T

TrivalentVertex (class in pychemcurv.core), 15

U

u_pi (pychemcurv.core.POAV2 property), 19

V

VertexAtom (class in pychemcurv.core), 13
vertices (pychemcurv.analysis.CurvatureAnalyzer
 property), 21
vertices_idx (pychemcurv.analysis.CurvatureAnalyzer
 property), 21

W

write_file() (pychemcurv.core.VertexAtom method),
 15