# OctaDist Documentation

***Release 3.0.0***

**OctaDist Development Team**

**Jun 02, 2021**

# User Documentation

**OctaDist: A tool for computing the distortion parameters in coordination complexes.**

OctaDist (**Octa**hedral **Dist**ortion calculator) is an inorganic chemistry and crystallography program for computing the distortion parameters, such as distance and angle distortions, in coordination complexes. For example, they are used for tracking structural change of the spin-crossover complex when the electronics spin-state changes from low-spin to high-spin, and vice versa. OctaDist can also be used to study other kind of the metal complex such as perovskite and metal-organic framework (MOF).

- Official homepage: https://octadist.github.io
- Github repository: https://github.com/OctaDist/OctaDist

# CHAPTER 1

## Program Status

OctaDist is maintained on Github version control system. All versions has been continuously tested using Travis CI. Currently, OctaDist project has two branches: Master (stable) and nightly-build (dev).

| Branch | Version | Status |
|---|---|---|
| Master | 3.0.0 | Active |
| Nightly-build | 3.1.0 | Active |

**Note:** OctaDist is open-source computer software and freely distributed under The GNU General Public License v3.0.

**Tip:** This documentation is generated be both user and reference code manuals. For more details, please go to the development page.

# CHAPTER 2

## Citation

Please cite this project when you use OctaDist for scientific publication.

```
Ketkaew, R.; Tantirungrotechai, Y.; Harding, P.; Chastanet, G.; Guionneau, P.;␣
↪Marchivie, M.; Harding, D. J.
OctaDist: A Tool for Calculating Distortion Parameters in Spin Crossover and␣
↪Coordination Complexes.
Dalton Trans., 2021,50, 1086-1096. https://doi.org/10.1039/D0DT03988H
```

BibTeX

```
@article{Ketkaew2021,
   doi = {10.1039/d0dt03988h},
   url = {https://doi.org/10.1039/d0dt03988h},
   year = {2021},
   publisher = {Royal Society of Chemistry ({RSC})},
   volume = {50},
   number = {3},
   pages = {1086--1096},
   author = {Rangsiman Ketkaew and Yuthana Tantirungrotechai and Phimphaka Harding␣
↪and Guillaume Chastanet and Philippe Guionneau and Mathieu Marchivie and David J.␣
↪Harding},
   title = {OctaDist: a tool for calculating distortion parameters in spin crossover␣
↪and coordination complexes},
   journal = {Dalton Transactions}
}
```

CHAPTER 3

# Bug report

For reporting a bug in OctaDist, please submit issues on OctaDist Github issues page. We appreciate all help and contribution in getting program development.

CHAPTER 4

# User Documentation

genindex, modindex, search

## 4.1 Getting Started

Welcome to the first section of the OctaDist documentation. Here you can find all information of OctaDist.

### 4.1.1 Why OctaDist?

Octahedral complex can be simply classified into two types: regular and distorted octahedron. The complexes with regular octahedral geometry (perfect octahedron) are expected to form, when all of the ligands are of the same kind. In contrast, if the ligands are of different kinds, the complex would turns the distorted octahedron instead. Octahedral distortion parameters has been widely used for determining the change of the distortion of the complexes.

Even though the people in community generally calculate the octahedral distortion parameters for their complexes, but they not used a certain way to do this. Moreover, there is no software for determining this kind of parameter yet. Therefore, we present the OctaDist program as a choice for those who are interested in this.

### 4.1.2 Features

#### Structural distortion analysis

- Determination of regular, irregular distorted, very distorted, and non-octahedral octahedral complexes
- **Calculation of octahedral distortion parameters**
  - Mean distance: $d_{mean}$
  - Distance distortion: $\zeta$
  - Angle distortion: $\Sigma$
  - Torsional distortion: $\Theta$

- Tilting distortion parameter: $\Delta$

### Molecular visualizations

- 3D modelling of complex
- Display of eight faces of octahedron
- Atomic orthogonal projection and projection plane
- Twisting triangular faces
- Molecular superposition (Overlay)

### Tools and Utilities

- Structural parameters
- Surface area
- Scripting Run supported
- Relationship plot between parameters
- Least-squares plane of selected ligand atoms
- Jahn-Teller distortion parameters
- Root-mean-square deviation of atomic positions (RMSD)

### Capabilities

- Cross-platform for both 32-bit and 64-bit systems
- Graphical user interface (GUI)
- Command line interface (CLI)
- User-friendly interactive scripting code
- User-adjustable program setting
- Simple and flexible processes of use
- On top of huge and complicated complexes
- Support for several output of computational chemistry software, including Gaussian, Q-Chem, ORCA, and NWChem

### Architectures

- Python-based program binding to Tkinter GUI toolkit and tested on PyCharm (Community Edition)
- Encapsulation of data, variable, and function as Class/Object.

### 4.1.3 Distortion parameters

Mathematical expression of the octahedral distortion parameters are given by following equations

- $\zeta$ parameter[1]

$$\zeta = \sum_{i=1}^{6} |d_i - d_{mean}|$$

where $d_i$ is individual M-X bond distance and $d_{mean}$ is mean metal-ligand bond distance.

- $\Delta$ parameter[2]

$$\Delta = \frac{1}{6} \sum_{i=1}^{6} (\frac{d_i - d_{mean}}{d_{mean}})^2$$

where $d_i$ is individual M-X bond distance and $d_{mean}$ is mean metal-ligand bond distance.

- $\Sigma$ parameter[3]

$$\Sigma = \sum_{i=1}^{12} |90 - \phi_i|$$

where $\phi_i$ in individual cis angle.

- $\Theta$ parameter[4]

$$\Theta = \sum_{i=1}^{24} |60 - \theta_i|$$

where $\theta_i$ is individual angle between two vectors of two twisting face.

### 4.1.4 System requirements

Minimum system requirements for OctaDist:

- Windows 7/8/10
- Linux (X11 Start)
- OS X 10.8+ and macOS 10.12+

## 4.2 Download OctaDist

### 4.2.1 Stable Version

The latest stable release of OctaDist is available for following OS and platforms:

[1] M. Buron-Le Cointe, J. Hébert, C. Baldé, N. Moisan, L. Toupet, P. Guionneau, J. F. Létard, E. Freysz, H. Cailleau, and E. Collet. - Intermolecular control of thermoswitching and photoswitching phenomena in two spin-crossover polymorphs. Phys. Rev. B 85, 064114.

[2] M. W. Lufaso and P. M. Woodward. - Jahn–Teller distortions, cation ordering and octahedral tilting in perovskites. Acta Cryst. (2004). B60, 10-20. DOI: 10.1107/S0108768103026661

[3] J. K. McCusker, A. L. Rheingold, D. N. Hendrickson. Variable-Temperature Studies of Laser-Initiated 5T2 → 1A1 Intersystem Crossing in Spin-Crossover Complexes: Empirical Correlations between Activation Parameters and Ligand Structure in a Series of Polypyridyl. Ferrous Complexes. Inorg. Chem. 1996, 35, 2100.

[4] M. Marchivie, P. Guionneau, J.-F. Létard, D. Chasseau. Photo-induced spin-transition: the role of the iron(II) environment distortion. Acta Crystal-logr. Sect. B Struct. Sci. 2005, 61, 25.

| Platform | Version | Download | |
|---|---|---|---|
| | | Full version | Lite version |
| Windows OS | | Full (exe) / Full (zip) | Lite (exe) / Lite (zip) |
| Linux OS | | Full (tar.gz) | N/A |
| macOS | | | |
| PyPI | | `pip install octadist` | |
| Anaconda | | `conda install -c rangsiman octadist` | |

**Note:** Both full and lite versions of OctaDist are open-source and free to download under the GNU v.3 license. The full version contains all capabilities including standard calculations, structural analysis, and molecular visualization, whereas the lite version includes only standard calculations.

### 4.2.2 Development Version

An on-going development build of OctaDist, called `nightly-build` branch. The tarball can be downloaded at Dev-build (zip) or use the following command:

```
wget https://github.com/OctaDist/OctaDist/archive/nightly-build.zip
```

You can also use `pip` to install the latest development build version on your system using the following command:

```
pip install git+https://github.com/octadist/octadist.git@nightly-build
```

**Note:** Python version must be equal or higher than 3.5. See *Development* for more details.

### 4.2.3 Release Archives

The source code and executable of all version and release note can be found at

| Version | Release date | Download link | Stats |
|---|---|---|---|
| 2.6.1 | Aug 24, 2019 | dl-2.6.1 | |
| 2.6.0 | Jun 22, 2019 | dl-2.6.0 | |
| 2.5.4 | Jun 10, 2019 | dl-2.5.4 | |
| 2.5.3 | May 22, 2019 | dl-2.5.3 | |
| 2.5.2 | May 6, 2019 | dl-2.5.2 | |
| 2.5.1 | May 1, 2019 | dl-2.5.1 | |
| 2.5.0 | Apr 25, 2019 | dl-2.5.0 | |
| 2.4 | Apr 21, 2019 | dl-2.4 | |
| 2.3-beta | Apr 14, 2019 | dl-2.3-beta | |
| 2.3-alpha | Mar 6, 2019 | dl-2.3-alpha | |
| 2.2 | Feb 9, 2019 | dl-2.2 | |
| 2.1 | Jan 25, 2019 | dl-2.1 | |
| 2.0 | Jan 24, 2019 | dl-2.0 | |
| 1.3 | Jan 20, 2019 | dl-1.3 | |
| 1.2 | Jan 12, 2019 | dl-1.2 | |
| 1.1 | Jan 8, 2019 | dl-1.1 | |
| 1.0 | Jan 8, 2019 | dl-1.0 | |

Total download:

## 4.3 Install OctaDist

OctaDist is a cross-platform software which is available for Windows, Linux, and macOS; for both 32-bit and 64-bit systems. You can install OctaDist by several ways, depending on your system and purpose.

### 4.3.1 Windows

Most of the Windows end-users do not have Python installed on their OS, so we strongly suggest you download and use a ready-to-use OctaDist executable.

Running OctaDist can be completed in a few steps as follows:

1. Download program executable (*.exe) to your machine:

```
OctaDist-*-Win-x86-64.exe
```

2. Right click on program icon and select:

```
Run as administrator
```

3. Click:

```
Yes
```

---

**Note:** Windows Defender might recognize OctaDist as third-party software. For first time starting OctaDist in Windows, you should run it as an administrator with full rights.

---

If you experience any problems while installing OctaDist on Windows OS, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

### 4.3.2 Linux

OctaDist is available on Python package index library, which can be found at https://pypi.org/project/octadist.

The end-user can use `pip`, a Python package-management system, to find and install OctaDist and other dependencies simultaneously.

Installing OctaDist can be completed in a few steps as follows:

1. Use `pip` command to install OctaDist:

```
pip install --user octadist
```

2. Execute OctaDist GUI, just type:

```
octadist
```

or:

```
octadist_gui
```

3. If you want to run OctaDist with command-line, just type:

```
octadist_cli
```

In case you are not able to install the latest version of OctaDust, you can try the older stable version:

```
pip install --user octadist==2.6.1
```

If you experience any problems while installing OctaDist on Linux, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

### 4.3.3 macOS

Like Linux, installing OctaDist on macOS can be completed in a few steps as follows:

1. Press **Command** - **spacebar** to launch Spotlight and type `Terminal`, then double-click the search result.
2. Use `pip` command to install OctaDist:

```
pip install --user octadist
```

3. Execute OctaDist GUI, just type:

```
octadist
```

or:

```
octadist_gui
```

4. If you want to execute OctaDist with command-line, just type:

```
octadist_cli
```

In case you are not able to install the latest version of OctaDust, you can try the older stable version:

```
pip install --user octadist==2.6.1
```

If you experience any problems while installing OctaDist on macOS, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

### 4.3.4 PyPI

The following commands are also useful for those who want to play with `pip`:

- Show info of package:

```
pip show octadist
```

- Install requirements packages:

```
pip install -r requirements.txt
```

- Install or upgrade to the latest version:

```
pip install --upgrade --user octadist
```

- Install/upgrade/downgrade to a certain version, for example, version 3.0.0:

```
pip install --upgrade --user octadist==3.0.0
```

- Install the package with a specific version of Python. for example:

```
python3.7 -m pip install --upgrade --user octadidst
```

- Uninstall package:

```
pip uninstall octadist
```

More details on installing Python package can be found its official website: https://packaging.python.org/tutorials/installing-packages.

If you experience any problems while installing OctaDist using PyPI, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

### 4.3.5 Anaconda

OctaDist is also available on Anaconda cloud server. The channel of OctaDist is at https://anaconda.org/rangsiman/octadist.

- It can be installed on system using command:

```
conda install -c rangsiman octadist
```

- To update OctaDist to the latest version:

```
conda update -c rangsiman octadist
```

- You can also create a personal environment only for OctaDist. For example, the following commands will create new env called `newenv`, then activate to this new env, and then install OctaDist from conda server:

```
conda create -n newenv python=3.7
activate newenv
conda update --all
conda install -c rangsiman octadist
```

- To clean conda cache:

```
conda clean --all
```

**Note:** OctaDist package on Anaconda server has been imported from PyPI server.

If you experience any problems while installing OctaDist using Conda, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

### 4.3.6 Python Package

OctaDist is a Python package and can be directly implemented into other applications. For example, that OctaDist is a package may be useful for interactive python script.

1. Check if your system has all dependencies for OctaDist:

```
python CheckPyModule.py
```

2. Download the source code (*.tar.gz) to your machine, for example, at **Download** directory:

```
OctaDist-*-src-x86-64.tar.gz
```

3. Uncompress the tarball, using **tar**:

```
tar -xzvf OctaDist-*-src-x86-64.tar.gz
```

4. Move to OctaDist root directory, using **cd**:

```
cd OctaDist-*-src-x86-64
```

5. Execute program like a package (you have to stay outside **octadist** directory):

```
python -m octadist
```

or command-line:

```
python -m octadist_cli
```

**Note:** The PyPI channel of OctaDist is at https://pypi.org/project/octadist/.

**Tip:** PIP-compressed zip files of OctaDist are also available at https://pypi.org/project/octadist/#files.

If you experience any problems while installing OctaDist from the source code, please do not hesitate to post your question at https://groups.google.com/g/octadist-forum.

## 4.4 Build OctaDist

This section will explain how to build OctaDist from source code. If you already have OctaDist installed on your system, this section may be skipped.

### 4.4.1 Prerequisites

This section will explain the dependency requirements for building OctaDist. As OctaDist is written in Python 3, you have to make sure that the version of Python on your system is equal or higher than 3.5. Check it by following command:

```
python --version
```

or

```
python3 --version
```

**Tip:** If you do have Python on the system, I would suggest you to read The Hitchhiker's Guide to Python. It is very useful!

Install Python 3 on:

- Windows
- Linux
- macOS

The following third-party packages are used in OctaDist.

```
numpy
scipy
matplotlib
rmsd
pymatgen
```

Actually, if you use `pip` to install OctaDist, the required dependencies will be installed automatically. However, you can install these packages yourself. This can be done with only one step:

```
pip install -r requirements.txt
```

### 4.4.2 Build the tarball, wheel, and egg

- `.tar.gz` : the tarball (supported by PIP)
- `.whl` : wheel file (supported by PIP)
- `.egg` : cross-platform zip file (supported by easy_install)

1. Build source code:

   ```
   python setup.py sdist bdist_wheel bdist_egg
   ```

2. Install OctaDist:

   ```
   python setup.py install
   ```

   or:

   ```
   pip install dist/*.tar.gz
   ```

3. Run test zip files:

   ```
   python setup.py test
   ```

4. Installed library of OctaDist will be install at `build/lib/octadist` directory.

5. Standalone executable (binary) file will be automatically added to environment variables, you can start OctaDist by calling its names anywhere:

   - To start graphical-interface:

     ```
     octadist
     ```

   - To start command-line:

     ```
     octadist_cli
     ```

**Note:** More details on Python package can be found its official website: https://packaging.python.org/tutorials/installing-packages.

### 4.4.3 Compile OctaDist to EXE

Program source code can be compiled as a standalone executable file (*.exe). Compilation can be completed easily using PyInstaller.

1. Upgrade pip:

```
pip install pip --upgrade
```

2. Install the latest version of PyInstaller:

```
pip install pyinstaller --upgrade
```

3. Check the version of PyInstaller:

```
pyinstaller --version
```

4. Change directory to `octadist` subdirectory, where `main.py` is, for example:

```
cd OctaDist-*-src-x86-64/octadist/
```

5. Compile a standalone, like this:

```
pyinstaller --onefile --windowed -n OctaDist-*-src-x86-64 main.py
```

6. The standalone executable will be build in `dist` directory.

**Note:** Other useful options for building executable can be found at PyInstaller manual.

## 4.5 Run OctaDist

OctaDist supports both a graphical user interface (GUI) and a command line interface (CLI).

### 4.5.1 Run OctaDist GUI using EXE

If you have a standalone executable (.exe) of OctaDist GUI on your system, run OctaDist by double-clicking the `.exe` file as if you open other program.

**Note:** OctaDist can take time to launch the application, usually 5 - 10 seconds. However, if the program does not start, please restart your system and run it again.

### 4.5.2 Run OctaDist GUI on the terminal

Moreover, OctaDist can be called on the terminal such as `CMD`, `PowerShell`, and `Terminal` as long as it is added to environment variable, like this:

```
octadist
```

### 4.5.3 Run OctaDist CLI

You can execute command-line OctaDist interface by typing `octadist_cli` on the terminal. If it is executed without argument, the help docs will show by default.

```
(py37) user@Linux:~$ octadist_cli

# output
usage: octadist_cli [-h] [-v] [-a] [-c] [-g] [-i INPUT] [-o] [-s OUTPUT]
                    [--par PARAMETER [PARAMETER ...]] [--show MOL [MOL ...]]

Octahedral Distortion Calculator:
A tool for computing octahedral distortion parameters in coordination complex.
For more details, please visit https://github.com/OctaDist/OctaDist.

optional arguments:
-h, --help            show this help message and exit
-v, --version         show program's version number and exit
-a, --about           show program info
-c, --cite            show how to cite OctaDist
-g, --gui             launch OctaDist GUI (this option is the same as
                        'octadist' command
-i INPUT, --inp INPUT
                        input structure in .xyz format
-o, --out             show formatted output summary
-s OUTPUT, --save OUTPUT
                        save formatted output to text file, please specify
                        name of OUTPUT file without '.txt' extension
--par PARAMETER [PARAMETER ...]
                        select which the parameter (zeta, delta, sigma, theta)
                        to show
--show MOL [MOL ...]  show atomic symbol (atom) and atomic coordinate
                        (coord) of octahedral structure

Rangsiman Ketkaew       Updated on 2021 E-mail: rangsiman1993@gmail.com
```

Using OctaDist to calculate the distortion of structure can be done as follows:

```
# Compute parameters
octadist_cli -i INPUT.xyz

# Compute parameters and show formatted output
octadist_cli -i INPUT.xyz -o

# Compute parameters and save output as file
octadist_cli -i INPUT.xyz -s OUTPUT
```

**Tip:** On Windows, you can check whether OctaDist is added to environment variables by using `where` command:

```
where octadist
```

For Linux and macOS, use `which` command instead:

```
which octadist
```

or

```
type -P "octadist" && echo "It's in path" || echo "It's not in path"
```

---

## 4.6 Example Calculations

### 4.6.1 Supported File Format

- **CIF file format**

File extension: `.cif` (https://en.wikipedia.org/wiki/Crystallographic_Information_File)

Crystallographic Information File (CIF). Example CIF is below:

```
data_ADH041

###############
## ENTRY ##
###############

_entry.id              ADH041

###############
## ATOM_SITE ##
###############
loop_
_atom_site.id
_atom_site.label_atom_id
_atom_site.label_comp_id
_atom_site.label_asym_id
_atom_site.auth_seq_id
_atom_site.cartn_x
_atom_site.cartn_y
_atom_site.cartn_z
_atom_site.occupancy
_atom_site.B_iso_or_equiv
_atom_site.label_entity_id
_atom_site.label_seq_id
1    O5*   G A   1       7.231  -2.196  -5.399  1.00 22.25   1 1
2    C5*   G A   1       6.950  -3.464  -4.723  1.00 15.86   1 1
3    C4*   G A   1       8.299  -4.018  -4.302  1.00 15.20   1 1
...
```

- **XYZ file format**

File extension: `.xyz` (https://en.wikipedia.org/wiki/XYZ_file_format)

```
<number of atoms>
comment line
<element 1>  <X> <Y> <Z>
<element 2>  <X> <Y> <Z>
<element 3>  <X> <Y> <Z>
...
```

- **Output of computational chemistry programs**

File extension: `.out` and `.log`

1. Gaussian
2. NWChem
3. ORCA
4. Q-Chem

### 4.6.2 Running the tests

#### Example 1

**Example 1 for running the test on OctaDist PyPI**

```
##################################################
# Example 1 for running the test on OctaDist PyPI #
##################################################

import octadist as oc

# The first atom must be metal center atom of octahedral structure.
# If not, please see example_2.py for how to handle this issue.

atom = ["Fe", "O", "O", "N", "N", "N", "N"]

coord = [
    [2.298354000, 5.161785000, 7.971898000],  # <- Metal atom
    [1.885657000, 4.804777000, 6.183726000],
    [1.747515000, 6.960963000, 7.932784000],
    [4.094380000, 5.807257000, 7.588689000],
    [0.539005000, 4.482809000, 8.460004000],
    [2.812425000, 3.266553000, 8.131637000],
    [2.886404000, 5.392925000, 9.848966000],
]

dist = oc.CalcDistortion(coord)
zeta = dist.zeta  # Zeta
delta = dist.delta  # Delta
sigma = dist.sigma  # Sigma
theta = dist.theta  # Theta

print("\nAll computed parameters")
print("-----------------------")
print("Zeta  =", zeta)
print("Delta =", delta)
print("Sigma =", sigma)
```

(continues on next page)

(continued from previous page)

```
print("Theta =", theta)

# All computed parameters
# ------------------------
# Zeta  = 0.22807256171728651
# Delta = 0.0004762517834704151
# Sigma = 47.926528379270124
# Theta = 122.688972774546
```

### Example 2

**Example 2 for running the test on OctaDist PyPI**

```
################################################
# Example 2 for running the test on OctaDist PyPI #
################################################

import octadist as oc

atom = ["O", "O", "Fe", "N", "N", "N", "N"]

coor = [
    [1.885657000, 4.804777000, 6.183726000],
    [1.747515000, 6.960963000, 7.932784000],
    [2.298354000, 5.161785000, 7.971898000],  # <- Metal atom
    [4.094380000, 5.807257000, 7.588689000],
    [0.539005000, 4.482809000, 8.460004000],
    [2.812425000, 3.266553000, 8.131637000],
    [2.886404000, 5.392925000, 9.848966000],
]

# If the first atom is not metal atom, you can rearrange the sequence
# of atom in list using coord.extract_octa method.

atom_octa, coord_octa = oc.io.extract_octa(atom, coor)

dist = oc.CalcDistortion(coord_octa)
zeta = dist.zeta  # Zeta
delta = dist.delta  # Delta
sigma = dist.sigma  # Sigma
theta = dist.theta  # Theta

print("\nAll computed parameters")
print("------------------------")
print("Zeta  =", zeta)
print("Delta =", delta)
print("Sigma =", sigma)
print("Theta =", theta)

# All computed parameters
# ------------------------
# Zeta  = 0.22807256171728651
# Delta = 0.0004762517834704151
# Sigma = 47.926528379270124
# Theta = 122.688972774546
```

### Example 3

**Example 3 for running the test on OctaDist PyPI**

```
#####################################################
# Example 3 for running the test on OctaDist PyPI #
#####################################################

import os
import octadist as oc

# You can also import your input file, like this:

dir_path = os.path.dirname(os.path.realpath(__file__))
input_folder = os.path.join(dir_path, "../example-input/")
file = input_folder + "Multiple-metals.xyz"

# Then use coord.extract_file to extract all atomic symbols and coordinates,
# and then use coord.extract_octa for taking the octahedral structure.

atom_full, coord_full = oc.io.extract_coord(file)
atom, coord = oc.io.extract_octa(atom_full, coord_full)

dist = oc.CalcDistortion(coord)
zeta = dist.zeta  # Zeta
delta = dist.delta  # Delta
sigma = dist.sigma  # Sigma
theta = dist.theta  # Theta

print("\nAll computed parameters")
print("-----------------------")
print("Zeta  =", zeta)
print("Delta =", delta)
print("Sigma =", sigma)
print("Theta =", theta)

# All computed parameters
# -----------------------
# Zeta  = 0.0030146365519487794
# Delta = 1.3695007180404868e-07
# Sigma = 147.3168033970211
# Theta = 520.6407679851042
```

### Example 4

**Example 4 for running the test on OctaDist PyPI**

```
#####################################################
# Example 4 for running the test on OctaDist PyPI #
#####################################################

import os
import octadist as oc

dir_path = os.path.dirname(os.path.realpath(__file__))
input_folder = os.path.join(dir_path, "../example-input/")
```

(continues on next page)

(continued from previous page)

```
file = input_folder + "Multiple-metals.xyz"

atom_full, coor_full = oc.io.extract_coord(file)

# If complex contains metal center more than one, you can specify the index metal
# whose octahedral structure will be computed.
# For example, this complex contains three metal atoms: Fe, Ru, and Rd.
# I add "2" as a second argument for choosing Ru as metal of interest.

atom, coord = oc.io.extract_octa(atom_full, coor_full, 2)

dist = oc.CalcDistortion(coord)
zeta = dist.zeta  # Zeta
delta = dist.delta  # Delta
sigma = dist.sigma  # Sigma
theta = dist.theta  # Theta

print("\nAll computed parameters")
print("-----------------------")
print("Zeta  =", zeta)
print("Delta =", delta)
print("Sigma =", sigma)
print("Theta =", theta)

# All computed parameters
# -----------------------
# Zeta  = 0.001616439510534251
# Delta = 3.5425830613072754e-08
# Sigma = 1.26579367508117
# Theta = 4.177042495798965
```

### Example 5

**Example 5 for running the test on OctaDist PyPI**

```
##################################################
# Example 5 for running the test on OctaDist PyPI #
##################################################

import os
import octadist as oc

dir_path = os.path.dirname(os.path.realpath(__file__))
input_folder = os.path.join(dir_path, "../example-input/")
file = input_folder + "Multiple-metals.xyz"

atom_full, coord_full = oc.io.extract_coord(file)

# Graphical display for octahedral complex
my_plot = oc.draw.DrawComplex_Matplotlib(atom=atom_full, coord=coord_full)
my_plot.add_atom()
my_plot.add_bond()
my_plot.add_legend()
my_plot.show_plot()
```

### Example 6

**Example 6 for running the test on OctaDist PyPI**

```
#####################################################
# Example 6 for running the test on OctaDist PyPI #
#####################################################

import os
import octadist as oc

dir_path = os.path.dirname(os.path.realpath(__file__))
input_folder = os.path.join(dir_path, "../example-input/")
file = input_folder + "Multiple-metals.xyz"

atom_full, coord_full = oc.io.extract_coord(file)

# Display and automatically save image as .png file with user-specified name
my_plot = oc.draw.DrawComplex_Matplotlib(atom=atom_full, coord=coord_full)
my_plot.add_atom()
my_plot.add_bond()
my_plot.add_legend()
my_plot.save_img()
my_plot.show_plot()

# Output image, Complex_saved_by_OctaDist.png, is stored at ../images directory
```

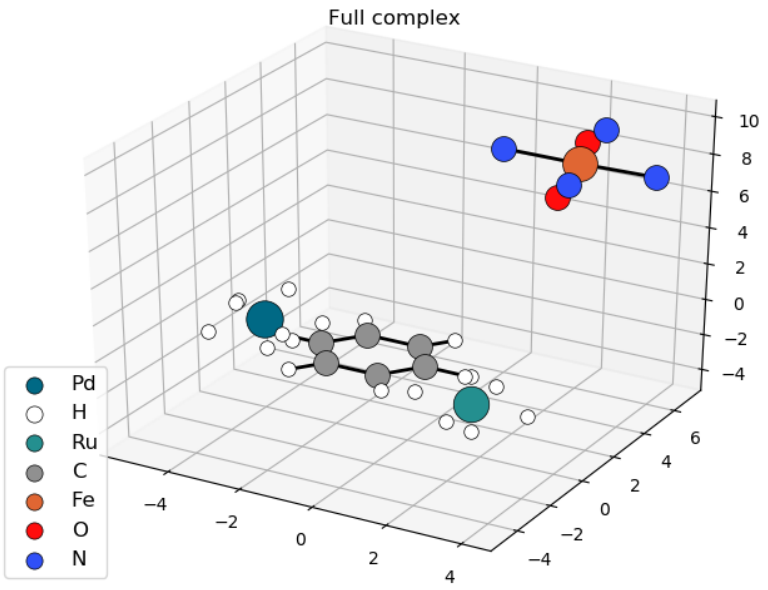


Fig. 1: Snapshot of structure saved by OctaDist.

### Example 7

**Example 7 for running the test on OctaDist PyPI**

```
##################################################
# Example 6 for running the test on OctaDist PyPI #
##################################################

# Display a molecule using Plotly visualizer

import os
import octadist as oc

dir_path = os.path.dirname(os.path.realpath(__file__))
input_folder = os.path.join(dir_path, "../example-input/")
file = input_folder + "Multiple-metals.xyz"

atom_full, coord_full = oc.io.extract_coord(file)

my_plot = oc.draw.DrawComplex_Plotly(atom=atom_full, coord=coord_full)
my_plot.add_atom()
my_plot.add_bond()
my_plot.show_plot()
```

## 4.7 Benchmarks

### 4.7.1 1. Perfect octahedral complex

Perfect iron metal complex:

```
Perfect-octahedron.xyz
Atom                      Cartesian coordinate
Fe          0.200698080     0.706806270     0.000000000
O           1.660698080     0.706806270     0.000000000
O           0.200698080     2.166806270     0.000000000
O           0.200698080     0.706806270     1.460000000
O          -1.259301920     0.706806270     0.000000000
O           0.200698080    -0.753193730     0.000000000
O           0.200698080     0.706806270    -1.460000000
```

- $d_{mean}$ = 1.460000 Angstrom
- $\zeta$ = 0.000000 Angstrom
- $\Delta$ = 0.00000000
- $\Sigma$ = 0.00000000 degree
- $\Theta$ = 0.00000000 degree

### 4.7.2 2. [Fe(1-bpp)2][BF4]2 complex in low-spin state

The XRD structure taken from Malcolm Halcrow's CCDC library:

```
[Fe(1-bpp)2][BF4]2-LS-Full.xyz
Atom                      Cartesian coordinate
Fe          4.067400000     7.204000000    13.611700000
N           4.303300000     7.375000000    11.729200000
```

(continues on next page)

(continued from previous page)

```
N          3.832600000     6.971500000    15.492600000
N          5.882200000     6.446100000    13.431200000
N          3.300200000     5.382800000    13.631600000
N          4.805500000     8.931800000    14.271600000
N          2.318400000     8.016500000    13.115200000
```

- $d_{mean}$ = 1.958109 Angstrom
- $\zeta$ = 0.203199 Angstrom
- $\Delta$ = 0.000348
- $\Sigma$ = 86.081494 degree
- $\Theta$ = 281.231091 degree

### 4.7.3 3. [Fe(1-bpp)2][BF4]2 complex in high-spin state

The XRD structure taken from Malcolm Halcrow's CCDC library:

```
[Fe(1-bpp)2][BF4]2-HS-Full.xyz
Atom                        Cartesian coordinate
Fe          4.904900000     6.913500000    14.248000000
N          4.982200000     6.876500000    12.110900000
N          4.671400000     6.741200000    16.368500000
N          6.853500000     6.086400000    13.701700000
N          5.683000000     8.779200000    15.108200000
N          4.107600000     4.898400000    14.643100000
N          2.957100000     7.673300000    13.543900000
```

- $d_{mean}$ = 2.178519 Angstrom
- $\zeta$ = 0.155914 Angstrom
- $\Delta$ = 0.000168
- $\Sigma$ = 150.814795 degree
- $\Theta$ = 496.648479 degree

### 4.7.4 4. Very distorted structure

Highly distorted structure:

```
Fe-very-distorted-octa.xyz
Atom                        Cartesian coordinate
Fe         18.268051000    11.289120000     2.565804000
O         19.074466000     9.706294000     3.743576000
O         19.823874000    10.436314000     1.381569000
N         18.364987000    13.407634000     2.249608000
N         16.149538000    11.306661000     2.913619000
N         18.599941000    12.116308000     4.528988000
N         17.364238000    10.733354000     0.657318000
```

- $d_{mean}$ = 2.149211 Angstrom
- $\zeta$ = 0.082408 Angstrom

- $\Delta$ = 0.000066
- $\Sigma$ = 182.673342 degree
- $\Theta$ = 673.278321 degree

## 4.8 Error and Fixing

### 4.8.1 1. OctaDist Startup Slow on Windows?

Windows Defender slow down OctaDist by scanning its file. You can fix this annoying issue by excluding OctaDist out of process scan list.

Here are the steps for adding OctaDist to exclusion list:

1. Go to `Start` > `Settings` -> `Update & Security` -> `Virus & threat protection`
2. Under Virus & threat protection settings select `Manage settings`
3. Under Exclusions, select `Add or remove exclusions` and select `Add exclusion`
4. Specify the name of OctaDist executable, for example:

   ```
   OctaDist-3.0.0-Win-x86-64.exe
   ```

5. Close OctaDist and run it again.

### 4.8.2 2. Missing some packages

If error message says *ImportError:* or *ModuleNotFoundError:*, some important packages have not been installed. To install all required packages, stay at top directory of OctaDist and type this command:

```
pip install -r requirements.txt
```

### 4.8.3 3. tkinter is not properly configured to Python

If you are using an old version of Python e.g. 3.7, tkinter somehow could not configured to this Python.

```
raceback (most recent call last):
File "/Users/nadia/Library/Python/3.7/lib/python/site-packages/octadist/octadist_gui.
↪py", line 21, in <module>
    import octadist.main
File "/Users/nadia/Library/Python/3.7/lib/python/site-packages/octadist/__init__.py",␣
↪line 112, in <module>
    from .src import molecule
File "/Users/nadia/Library/Python/3.7/lib/python/site-packages/octadist/src/molecule.
↪py", line 22, in <module>
    from octadist.src import elements, popup
File "/Users/nadia/Library/Python/3.7/lib/python/site-packages/octadist/src/popup.py",
↪ line 17, in <module>
    from tkinter.messagebox import showinfo, showerror, showwarning
File "/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/
↪Versions/3.7/lib/python3.7/tkinter/__init__.py", line 36, in <module>
    import _tkinter # If this fails your Python may not be configured for Tk
```

Solutions: Reinstall Tk and/or install Python 3.8 or newer version

- Linux (Ubuntu): `sudo apt install python3.8`
- macOS: `brew install python@3.8`

### 4.8.4 4. Failed to build PEP517

```
ERROR: Could not build wheels for scipy which use PEP 517 and cannot be installed␣
↪directly
```

Solutions: - `pip3 install --upgrade pip` - `pip3 install --user PEP517`

### 4.8.5 5. MPL error

If program crashes with confusing errors messages, you may need to set *MPLBACKEND* environment variable before running the program, like this:

```
export MPLBACKEND=TkAgg
```

### 4.8.6 6. Cannot connect to X11 server

If you run GUI using *octadist* or *octadist_gui* and then it fails with the following error:

```
(py37) nutt@Ubuntu:~$ octadist

Program Starts >>>
... OctaDist 3.0.0 January 2021 ...
Traceback (most recent call last):
    File "/home/nutt/.local/bin/octadist", line 10, in <module>
    sys.exit(run_gui())
    File "/home/nutt/.local/lib/python3.7/site-packages/octadist/__main__.py", line␣
↪35, in run_gui
    app = octadist.main.OctaDist()
    File "/home/nutt/.local/lib/python3.7/site-packages/octadist/main.py", line 68,␣
↪in __init__
    self.master = tk.Tk()
    File "/usr/lib/python3.7/tkinter/__init__.py", line 2023, in __init__
    self.tk = _tkinter.create(screenName, baseName, className, interactive,␣
↪wantobjects, useTk, sync, use)
_tkinter.TclError: couldn't connect to display ":0"
```

The above message implies that your system cannot connect to X11 server used for displaying the GUI of program. This error usually happens on Debian or Ubuntu (and Windows Subsystem for Linux on Windows). So, you need to install X11 server as follows:

**X11 Client Installation**

To install the *xauth* package, use *apt-get*:

```
sudo apt-get install xauth
```

**X11 Server Installation**

To install a minimal X11 on Ubuntu Server edition:

```
sudo apt-get install xorg
sudo apt-get install openbox
```

**Tip:** If you find any issues, do not hesitate to let us know. Your suggestions would help OctaDist getting improved.

## 4.9 Modules

### 4.9.1 Program structure

OctaDist is composed of the following modules:

| Function | Description |
|---|---|
| main | Main program |
| calc | Calculating distortion parameters |
| draw | Displaying molecule |
| elements | Atomic properties |
| linear | Built-in mathematical functions |
| io | Manipulating atomic coordinates |
| plane | Manipulate projection plane |
| plot | Plotting graph and chart |
| popup | Error, warning, and info messages |
| projection | 2D & 3D vector projections |
| scripting | Interactive code Console |
| structure | All data about structure |
| tools | Analysis tools by 3rd-party libraries |
| util | Frequently-used functions e.g. find atomic bonds |

### 4.9.2 Application Program Interface (API)

| API version | Description |
|---|---|
| octadist_gui | Graphical user interface (__main__.py) |
| octadist_cli | Command line interface (octadist_cli.py) |

### 4.9.3 Source code

#### octadist.main

**class** octadist.main.**OctaDist**

OctaDist class initiates main program UI and create all widgets.

Program interface is structured as follows:

| Program Menu | |
|---|---|
| Frame 1 | |
| Frame 2 | Frame 3 |
| Frame 4 | |

- Frame 1 : Program name and short description
- Frame 2 : Program console
- Frame 3 : Textbox for showing summary output
- Frame 4 : textbox for showing detailed output

### Examples

```
>>> my_app = OctaDist()
>>> my_app.start_app()
```

**create_logo**()
Create icon file from Base64 raw code.

This will be used only for Windows OS.

Other OS like Linux and macOS use default logo of Tkinter.

### Examples

```
>>> if self.octadist_icon is True:
>>>     self.create_logo()
>>> else:
>>>     pass
```

**start_master**()
Start application with UI settings.

**add_menu**()
Add menu bar to master windows.

**add_widgets**()
Add all widgets and components to master windows.

GUI style of widgets in master windows use ttk style.

**show_text**(*text*)
Insert text to result box

**Parameters** **text** (*str*) – Text to show in result box.

**Returns** **None** (*None*)

**welcome_msg**()
Show welcome message in result box:

1. Program name, version, and release.
2. Full author names.
3. Official website: https://octadist.github.io.

**open_file**()
Open file dialog where the user will browse input files.

**search_coord**()
Search and extract atomic symbols and coordinates from input file.

**See also:**

*octadist.src.io.extract_coord()* Extract atomic symbols and atomic coordinates from input file.

*octadist.src.io.extract_octa()* Extract octahedral structure from complex.

**show_coord**()
Show coordinates in box.

**save_results**()
Save results as output file. Default file extension is .txt.

**save_coord**()
Save atomic coordinates (Cartesian coordinate) of octahedral structure. Default file extension is .xyz.

**calc_distortion**()
Calculate all distortion parameters:

- D_mean
- Zeta
- Delta
- Sigma
- Theta

**See also:**

*octadist.src.calc.CalcDistortion.calc_d_mean()* Calculate mean metal-ligand bond length.

*octadist.src.calc.CalcDistortion.calc_zeta()* Calculate Zeta parameter.

*octadist.src.calc.CalcDistortion.calc_delta()* Calculate Delta parameter.

*octadist.src.calc.CalcDistortion.calc_sigma()* Calculate Sigma parameter.

*octadist.src.calc.CalcDistortion.calc_theta()* Calculate Theta parameter.

**settings**()
Program settings allows the user to configure the values of variables that used in molecular display function.

For example, cutoff distance for screening bond distance between atoms.

**copy_name**()
Copy input file name to clipboard.

**See also:**

*copy_path()* Copy absolute path of input file to clibboard.

*copy_results()* Copy results to clibboard.

*copy_octa()* Copy octahedral structure coordinates to clibboard.

**copy_path**()
Copy absolute path of input file to clipboard.

**See also:**

*copy_name()* Copy input file name to clibboard.

*copy_results()* Copy results to clibboard.

*copy_octa()* Copy octahedral structure coordinates to clibboard.

**copy_results**()
Copy the results and computed distortion parameters to clipboard.

**See also:**

*copy_name()* Copy input file name to clibboard.

*copy_path()* Copy absolute path of input file to clibboard.

*copy_octa()* Copy octahedral structure coordinates to clibboard.

**copy_octa**()
Copy atomic coordinates of octahedral structure to clipboard.

**See also:**

*copy_name()* Copy input file name to clibboard.

*copy_path()* Copy absolute path of input file to clibboard.

*copy_results()* Copy results to clibboard.

**edit_file**()
Edit file by specified text editor on Windows.

**See also:**

*settings()*

**scripting_console**()
Start scripting interface for an interactive code.

User can access to class variable (dynamic variable).

| Output box |
| --- |
| Input box |

**See also:**

*settings()* Program settings.

**draw_all_atom**()
Display 3D complex.

**See also:**

**octadist.src.draw.DrawComplex()** Show 3D molecule.

**draw_all_atom_and_face**()
Display 3D complex with the faces.

**See also:**

**octadist.src.draw.DrawComplex()** Show 3D molecule.

**draw_octa**()
Display 3D octahedral structure.

**See also:**

**octadist.src.draw.DrawComplex()** Show 3D molecule.

**draw_octa_and_face**()
Display 3D octahedral structure with the faces.

**See also:**

**octadist.src.draw.DrawComplex()** Show 3D molecule.

**draw_projection**()
Draw projection planes.

**See also:**

*octadist.src.draw.DrawProjection()* Show graphical projections.

**draw_twisting_plane**()
Draw twisting triangular planes.

**See also:**

*octadist.src.draw.DrawTwistingPlane()* Show graphical triangular twisting planes.

**show_data_complex**()
Show info of input complex.

**See also:**

*octadist.src.structure.DataComplex()* Show data summary of complex.

**show_param_octa**()
Show structural parameters of selected octahedral structure.

**See also:**

*octadist.src.structure.StructParam()* Show structural parameter symmary of complex.

**show_surface_area**()
Calculate the area of eight triangular faces of octahedral structure.

**See also:**

*octadist.src.structure.SurfaceArea()* Show the area of the faces of octahedral structure.

**plot_zeta_sigma**()
Plot relationship between zeta and sigma.

**See also:**

*octadist.src.plot.Plot()* Show relationship plot.

**plot_sigma_theta**()
Plot relationship between sigma and theta.

**See also:**

*octadist.src.plot.Plot()* Show relationship plot.

**tool_jahn_teller**()
Calculate Jahn-Teller distortion parameter.

**See also:**

*octadist.src.tools.CalcJahnTeller()* Calculate Jahn-Teller distortion parameter.

**tool_rmsd**()
Calculate root mean squared displacement of atoms in complex, RMSD.

**See also:**

*octadist.src.tools.CalcRMSD()* Calculate RMSD.

**static check_update**()
Check program update by comparing version of program user is using with that of the latest version released on github.

### References

File: https://www.github.com/OctaDist/OctaDist/version_update.txt.

**static callback**(*event*)
On-clink open web browser.

**Parameters** **event** (*object*) – Event object for callback.

**static show_about**()
Show author details on a sub-window.

1. Name of authors
2. Official program website
3. Citation

**static show_license**()
Show license details on a sub-window.

GNU General Public License version 3.0.

### References

Link: https://www.gnu.org/licenses/gpl-3.0.en.html.

**clear_cache**()
Clear program cache by nullifying all default variables and clear both of parameter and result boxes.

**clear_param_box**()
Clear parameter box.

**clear_result_box**()
Clear result box.

**start_app**()
Start application.

octadist.main.**main**()

### octadist.gui

octadist.octadist_gui.**run_gui**()
OctaDist graphical user interface (GUI).

### octadist.cli

octadist.octadist_cli.**check_file**(*file*)
Check if input file is exist or not.

**Parameters** **file** (*str*) – Input file name.

**Returns** **file** (*str*) – Input file name.

octadist.octadist_cli.**find_coord**(*file*)
Find atomic symbols and atomic coordinates of structure.

**Parameters** **file** (*str*) – Input file name.

**Returns**

- **atom** (*list*) – Atomic symbols.
- **coord** (*list*) – Atomic coordinates.

octadist.octadist_cli.**find_octa**(*atom*, *coord*)
Find atomic symbols and atomic coordinates of structure.

**Parameters**

- **atom** (*list*) – Atomic symbols of structure.
- **coord** (*list*) – Atomic coordinates of structure.

**Returns**

- **atom** (*list*) – Atomic symbols of octahedral structure.
- **coord** (*array_like*) – Atomic coordinates of octahedral structure.

octadist.octadist_cli.**calc_param**(*coord*)
Calculate octahedral distortion parameters.

**Parameters** **coord** (*array_like*) – Atomic coordinates of octahedral structure.

**Returns** **computed** (*dict*) – Computed parameters.

octadist.octadist_cli.**run_cli**()
OctaDist command-line interface (CLI). This function has been implemented by entry points function in setuptools package.

### octadist.calc

**class** octadist.src.calc.**CalcDistortion**(*coord*)
Calculate octahedral histortion parameters:

- Bond distance : *calc_d_bond()*

- Mean bond distance : *calc_d_mean()*
- Bond angle around metal center atom : *calc_bond_angle()*
- zeta parameter : *calc_zeta()*
- Delta parameter : *calc_delta()*
- Sigma parameter : *calc_sigma()*
- Minimum Tehta parameter : *calc_theta_min()*
- Maximum Theta parameter : *calc_theta_max()*
- Mean Theta parametes : *calc_theta()*

**Parameters** **coord** (*array_like*) – Atomic coordinates of octahedral structure.

### Examples

```
>>> coord = [[2.298354000, 5.161785000, 7.971898000],  # <- Metal atom
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> test = CalcDistortion(coord)
>>> test.sigma
47.926528379270124
```

**calc_d_bond**()
Calculate metal-ligand bond distance and return value in Angstrom.

**See also:**

***calc_d_mean()*** Calculate mean metal-ligand bond length.

**calc_d_mean**()
Calculate mean distance parameter and return value in Angstrom.

**See also:**

***calc_d_bond()*** Calculate metal-ligand bonds length.

**calc_bond_angle**()
Calculate 12 cis and 3 trans unique angles in octahedral structure.

**See also:**

***calc_sigma()*** Calculate Sigma parameter.

**calc_zeta**()
Calculate zeta parameter[1] and return value in Angstrom.

**See also:**

[1] M. Buron-Le Cointe, J. Hébert, C. Baldé, N. Moisan, L. Toupet, P. Guionneau, J. F. Létard, E. Freysz, H. Cailleau, and E. Collet. - Intermolecular control of thermoswitching and photoswitching phenomena in two spin-crossover polymorphs. Phys. Rev. B 85, 064114.

*calc_d_bond()* Calculate metal-ligand bonds length.

*calc_d_mean()* Calculate mean metal-ligand bond length.

### References

**calc_delta**()
Calculate Delta parameter, also known as Tilting distortion parameter[2].

**See also:**

*calc_d_bond()* Calculate metal-ligand bonds length.

*calc_d_mean()* Calculate mean metal-ligand bond length.

### References

**calc_sigma**()
Calculate Sigma parameter[3] and return value in degree.

**See also:**

*calc_bond_angle()* Calculate bond angles between ligand-metal-ligand.

### References

**determine_faces**()
Refine the order of ligand atoms in order to find the plane for projection.

**Returns**

- **coord_metal** (*array_like*) – Coordinate of metal atom.
- **coord_lig** (*array_like*) – Coordinate of ligand atoms.

**See also:**

*calc_theta()* Calculate mean Theta parameter

### Examples

```
>>> bef = np.array([
            [4.0674, 7.2040, 13.6117]
            [4.3033, 7.3750, 11.7292]
            [3.8326, 6.9715, 15.4926]
            [5.8822, 6.4461, 13.4312]
            [3.3002, 5.3828, 13.6316]
            [4.8055, 8.9318, 14.2716]
            [2.3184, 8.0165, 13.1152]
            ])
```

(continues on next page)

[2] M. W. Lufaso and P. M. Woodward. - Jahn–Teller distortions, cation ordering and octahedral tilting in perovskites. Acta Cryst. (2004). B60, 10-20. DOI: 10.1107/S0108768103026661

[3] James K. McCusker, A. L. Rheingold, D. N. Hendrickson. Variable-Temperature Studies of Laser-Initiated 5T2 → 1A1 Intersystem Crossing in Spin-Crossover Complexes: Empirical Correlations between Activation Parameters and Ligand Structure in a Series of Polypyridyl. Ferrous Complexes. Inorg. Chem. 1996, 35, 2100.

(continued from previous page)

```
>>> metal, coord = self.determine_faces(bef)
>>> metal
[ 4.0674  7.204  13.6117]
>>> coord_lig
[[ 4.3033  7.375  11.7292]      # Front face
 [ 4.8055  8.9318 14.2716]      # Front face
 [ 5.8822  6.4461 13.4312]      # Front face
 [ 2.3184  8.0165 13.1152]      # Back face
 [ 3.8326  6.9715 15.4926]      # Back face
 [ 3.3002  5.3828 13.6316]]     # Back face
```

**calc_theta**()

Calculate Theta parameter[4] and value in degree.

**See also:**

***calc_theta_min()*** Calculate minimum Theta parameter.

***calc_theta_max()*** Calculate maximum Theta parameter.

***octadist.src.linear.angle_btw_vectors()*** Calculate cosine angle between two vectors.

***octadist.src.linear.angle_sign()*** Calculate cosine angle between two vectors sensitive to CW/CCW direction.

***octadist.src.plane.find_eq_of_plane()*** Find the equation of the plane.

***octadist.src.projection.project_atom_onto_plane()*** Orthogonal projection of point onto the plane.

### References

**calc_theta_min**()

Calculate minimum Theta parameter and return value in degree.

**See also:**

***calc_theta()*** Calculate mean Theta parameter

**calc_theta_max**()

Calculate maximum Theta parameter and return value in degree.

**See also:**

***calc_theta()*** Calculate mean Theta parameter

## octadist.draw

**class** octadist.src.draw.**DrawComplex_Matplotlib**(*atom=None*, *coord=None*, *cutoff_global=2.0*, *cutoff_hydrogen=1.2*)

Display 3D structure of octahedral complex with label for each atoms using Matplotlib.

**Parameters**

- **atom** (*list*) – Atomic symbols of octahedral structure. Default is None.

[4] M. Marchivie, P. Guionneau, J.-F. Létard, D. Chasseau. Photo-induced spin-transition: the role of the iron(II) environment distortion. Acta Crystal-logr. Sect. B Struct. Sci. 2005, 61, 25.

- **coord** (*list or array_like or tuple or bool*) – Atomic coordinates of octahedral structure. Default is None.
- **cutoff_global** (*int or float*) – Global cutoff for screening bonds. Default is 2.0.
- **cutoff_hydrogen** (*int or float*) – Cutoff for screening hydrogen bonds. Default is 1.2.

**See also:**

**draw.DrawComplex_Plotly** Use Plotly engine to draw a complex.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> test = DrawComplex_Matplotlib(atom=atom, coord=coord)
>>> test.add_atom()
>>> test.add_bond()
>>> test.add_legend()
>>> test.show_plot()
```

**start_plot**()
Introduce figure to plot.

**plot_title**(*title='Full complex'*, *font_size='12'*)
Add plot title at top position.

**Parameters**

- **title** (*str*) – Top title of the plot. Default is "Full complex".
- **fontsize** (*int or float or str*) – Font size of title. Default is "12".

**add_atom**()
Add all atoms to show in figure.

**add_symbol**()
Add symbol of atoms to show in figure.

**add_bond**()
Calculate bond distance, screen bond, and add them to show in figure.

**See also:**

***octadist.src.util.find_bonds()*** Find atomic bonds.

**add_face**(*coord*)
Find the faces of octahedral structure and add those faces to show in figure.

**See also:**

***octadist.src.util.find_faces_octa()*** Find all faces of octahedron.

**add_legend**()
Add atoms legend to show in figure.

### References

1. **Remove duplicate labels in legend.** Ref: https://stackoverflow.com/a/26550501/6596684.
2. **Fix size of point in legend.** Ref: https://stackoverflow.com/a/24707567/6596684.

**config_plot**(*show_title=True*, *show_axis=True*, *show_grid=True*, ***kwargs*)
Setting configuration for figure.

**Parameters**

- **show_title** (*bool*) – If True, show title of figure. If False, not show title of figure.
- **show_axis** (*bool*) – If True, show axis of figure. If False, not show axis of figure.
- **show_grid** (*bool*) – If True, show grid of figure. If False, not show grid of figure.
- **kwargs** (*dict, optional*) – title_name : title name of figure. title_size : text size of title. label_size : text size of axis labels.

**static save_img**(*save='Complex_saved_by_OctaDist'*, *file='png'*)
Save figure as an image.

**Parameters**

- **save** (*str*) – Name of image file. Default is "Complex_saved_by_OctaDist".
- **file** (*str*) – Image type. Default is "png".

**static show_plot**()
Show plot.

**class** octadist.src.draw.**DrawComplex_Plotly**(*atom=None*, *coord=None*, *cutoff_global=2.0*, *cutoff_hydrogen=1.2*)
Display 3D structure of octahedral complex in web browser using Plotly.

**Parameters**

- **atom** (*list*) – Atomic symbols of octahedral structure. Default is None.
- **coord** (*list or array_like or tuple or bool*) – Atomic coordinates of octahedral structure. Default is None.
- **cutoff_global** (*int or float*) – Global cutoff for screening bonds. Default is 2.0.
- **cutoff_hydrogen** (*int or float*) – Cutoff for screening hydrogen bonds. Default is 1.2.

**See also:**

**draw.DrawComplex_Matplotlib** Use Matplotlib engine to draw a complex.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
```

(continues on next page)

(continued from previous page)

```
            [4.094380000, 5.807257000, 7.588689000],
            [0.539005000, 4.482809000, 8.460004000],
            [2.812425000, 3.266553000, 8.131637000],
            [2.886404000, 5.392925000, 9.848966000]]
>>> test = DrawComplex_Plotly(atom=atom, coord=coord)
>>> test.add_atom()
>>> test.add_bond()
>>> test.show_plot()
```

**start_plot**()
Introduce figure to plot.

**plot_title**(*title='Full complex'*, *font_size='12'*)
Add plot title at top position.

**Parameters**

- **title** (*str*) – Top title of the plot. Default is "Full complex".
- **fontsize** (*int or float or str*) – Font size of title. Default is "12".

**add_atom**()
Add all atoms to show in figure.

**add_bond**()
Calculate bond distance, screen bond, and add them to show in figure.

**See also:**

***octadist.src.util.find_bonds()*** Find atomic bonds.

**save_img**(*save='Complex_saved_by_OctaDist'*, *file='png'*)
Save figure as an image. Note that psutil and plotly-orca are needed for saving Plotly plot as image.

**Parameters**

- **save** (*str*) – Name of image file. Default is "Complex_saved_by_OctaDist".
- **file** (*str*) – Image type. Default is "png".

**show_plot**()
Show plot.

**class** octadist.src.draw.**DrawProjection**(*atom=None*, *coord=None*)
Display the selected 4 faces of octahedral complex.

**Parameters**

- **atom** (*list*) – Atomic symbols of octahedral structure. Default is None.
- **coord** (*list or array_like or tuple*) – Atomic coordinates of octahedral structure. Default is None.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
```

(continues on next page)

(continued from previous page)

```
            [4.094380000, 5.807257000, 7.588689000],
            [0.539005000, 4.482809000, 8.460004000],
            [2.812425000, 3.266553000, 8.131637000],
            [2.886404000, 5.392925000, 9.848966000]]
>>> test = DrawProjection(atom=atom, coord=coord)
>>> test.add_atom()
>>> test.add_symbol()
>>> test.add_plane()
>>> test.show_plot()
```

**start_plot**()
Introduce figure to plot.

**plot_title**(*title='4 pairs of opposite planes'*, *font_size='x-large'*)
Add plot title at top position.

**Parameters**

- **title** (*str*) – Top title of the plot. Default is "Full complex".
- **fontsize** (*int or float or str*) – Font size of title. Default is "x-large".

**shift_plot**()
Shift subplots down. Default is 0.25.

**add_atom**()
Add all atoms to show in figure.

**add_symbol**()
Add all atoms to show in figure.

**add_plane**()
Add the projection planes to show in figure.

**See also:**

***octadist.src.util.find_faces_octa()*** Find all faces of octahedron.

**static save_img**(*save='Complex_saved_by_OctaDist'*, *file='png'*)
Save figure as an image.

**Parameters**

- **save** (*str*) – Name of image file. Default is "Complex_saved_by_OctaDist".
- **file** (*file*) – Image type. Default is "png".

**static show_plot**()
Show plot.

**class** octadist.src.draw.**DrawTwistingPlane**(*atom=None*, *coord=None*, *symbol_fontsize=15*)
Display twisting triangular faces and vector projection.

**Parameters**

- **atom** (*list*) – Atomic symbols of octahedral structure. Default is None.
- **coord** (*list or array or tuple*) – Atomic coordinates of octahedral structure. Default is None.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> test = DrawTwistingPlane(atom=atom, coord=coord)
>>> test.add_plane()
>>> test.add_symbol()
>>> test.add_bond()
>>> test.show_plot()
```

**start_plot**()
Introduce figure to plot.

**plot_title**(*title='Projected twisting triangular faces'*, *font_size='x-large'*)
Add plot title at top position.

**Parameters**

- **title** (*str*) – Top title of the plot. Default is "Projected twisting triangular faces".
- **fontsize** (*int or float or str*) – Font size of title. Default is "x-large".

**shift_plot**()
Shift subplots down. Default is 0.25.

**create_subplots**()
Create subplots.

**add_plane**()
Add the projection planes to show in figure.

**See also:**

***octadist.src.plane.find_eq_of_plane()*** Find the equation of the plane.

***octadist.src.projection.project_atom_onto_plane()*** Orthogonal projection of point onto the plane.

**add_symbol**()
Add all atoms to show in figure.

**add_bond**()
Calculate bond distance, screen bond, and add them to show in figure.

**static save_img**(*save='Complex_saved_by_OctaDist'*, *file='png'*)
Save figure as an image.

**Parameters**

- **save** (*str*) – Name of image file. Default is "Complex_saved_by_OctaDist".
- **file** (*str*) – Image type. Default is "png".

**static show_plot**()
Show plot.

### octadist.elements

octadist.src.elements.**number_to_symbol**(*x*)
Convert atomic number to symbol and vice versa for atom 1-109.

**Parameters** **x** (*str or int*) – symbol or atomic number.

**Returns**

- **atom[x]** (*str*) – If x is atomic number, return symbol.
- **atom.index(i)** (*int*) – If x is symbol, return atomic number.

#### Examples

```
>>> check_atom('He')
2
>>> check_atom(2)
'He'
```

octadist.src.elements.**number_to_radii**(*x*)
Convert atomic number (index) to atom radii in Angstroms: 1-119.

**Parameters** **x** (*int*) – Atomic number.

**Returns** **atom_radii[x]** (*int*) – Atomic radius.

#### Examples

```
>>> check_radii(2) # He
0.93
```

octadist.src.elements.**number_to_color**(*x*)
Convert atomic number to color: 1-109.

**Parameters** **x** (*int*) – Atomic number.

**Returns** **atomic color[x]** (*str*) – Atomic color.

#### References

http://jmol.sourceforge.net/jscolors/

#### Examples

```
>>> check_color(2) # He
'#D9FFFF'
```

### octadist.io

octadist.src.io.**is_cif**(*f*)
Check if the input file is .cif file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is CIF file, return True.

**See also:**

***get_coord_cif()*** Find atomic coordinates of molecule from CIF file.

### Notes

More details about CIF file format are provided at https://en.wikipedia.org/wiki/Crystallographic_Information_File.

### Examples

```
>>> # example.cif
>>> # example
>>> # _audit_creation_date              2012-10-26T21:09:50-0400
>>> # _audit_creation_method            fapswitch 2.2
>>> # _symmetry_space_group_name_H-M    P1
>>> # _symmetry_Int_Tables_number       1
>>> # _space_group_crystal_system       triclinic
>>> # _cell_length_a                    16.012374
>>> # _cell_length_b                    14.740457
>>> # _cell_length_c                    19.436146
>>> # _cell_angle_alpha                 89.939227
>>> # _cell_angle_beta                  90.110039
>>> # _cell_angle_gamma                 90.015104
>>> # _cell_volume                      4587.49671393
>>> #
>>> # loop_
>>> # _atom_site_label
>>> # _atom_site_type_symbol
>>> # _atom_type_description
>>> # _atom_site_fract_x
>>> # _atom_site_fract_y
>>> # _atom_site_fract_z
>>> # _atom_type_partial_charge
>>> # C1    C     C_R   0.340882 0.499989 0.500098 0.541130
>>> # C2    C     C_R   0.528123 0.048033 0.558069 0.232589
>>> # C3    C     C_R   0.499931 0.902862 0.500001 -0.063750
>>> # C4    C     C_R   0.500061 0.097137 0.500001 -0.063745
>>> # C5    C     C_1   0.499958 0.802655 0.499991 0.266033
>>> # ...
>>> is_cif("example.cif")
True
```

octadist.src.io.**get_coord_cif**(*f*)

Get coordinate from .cif file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "example.cif"
>>> atom, coord = get_coord_cif(file)
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**is_xyz**(*f*)

Check if the input file is .xyz file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is XYZ file, return True.

**See also:**

***get_coord_xyz()*** Find atomic coordinates of molecule from XYZ file.

### Examples

```
>>> # example.xyz
>>> # 20
>>> # Comment: From Excel file
>>> # Fe  6.251705    9.063211    5.914842
>>> # N   8.15961     9.066456    5.463087
>>> # N   6.749414    10.457551   7.179682
>>> # N   5.709997    10.492955   4.658257
>>> # N   4.350474    9.106286    6.356091
>>> # O   5.789096    7.796326    4.611355
>>> # O   6.686381    7.763872    7.209699
>>> # ...
>>> is_xyz("example.xyz")
True
```

octadist.src.io.**get_coord_xyz**(*f*)

Get coordinate from .xyz file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "Fe-distorted-complex.xyz"
>>> atom, coord = get_coord_xyz(file)
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**is_gaussian**(*f*)

Check if the input file is Gaussian file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is Gaussian output file, return True.

**See also:**

***get_coord_gaussian()*** Find atomic coordinates of molecule from Gaussian file.

### Examples

```
>>> # gaussian.log
>>> #                             Standard orientation:
>>> # ---------------------------------------------------------------------
>>> # Center     Atomic      Atomic             Coordinates (Angstroms)
>>> # Number     Number       Type              X           Y           Z
>>> # ---------------------------------------------------------------------
>>> #      1         26           0        0.000163    1.364285   -0.000039
>>> #      2          8           0        0.684192    0.084335   -1.192008
>>> #      3          8           0       -0.683180    0.083251    1.191173
>>> #      4          7           0        1.639959    1.353157    1.006941
>>> #      5          7           0       -0.563377    2.891083    1.435925
>>> # ...
>>> is_gaussian("gaussian.log")
True
```

octadist.src.io.**get_coord_gaussian**(*f*)

Extract XYZ coordinate from Gaussian output file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "Gaussian-Fe-distorted-complex.out"
>>> atom, coord = get_coord_gaussian(file)
```

(continues on next page)

(continued from previous page)

```
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**is_nwchem**(*f*)

Check if the input file is NWChem file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is NWChem output file, return True.

**See also:**

***get_coord_nwchem()*** Find atomic coordinates of molecule from NWChem file.

### Examples

```
>>> # nwchem.out
>>> #    ----------------------
>>> #    Optimization converged
>>> #    ----------------------
>>> # ...
>>> # ...
>>> #  No.       Tag          Charge          X              Y              Z
>>> # ---- ---------------- ---------- -------------- -------------- --------------
 ↪-
>>> #    1 Ru(Fragment=1)      44.0000    -3.04059115    -0.08558108    -0.
 ↪07699482
>>> #    2 C(Fragment=1)        6.0000    -1.62704660     2.40971357     0.
 ↪63980357
>>> #    3 C(Fragment=1)        6.0000    -0.61467778     0.59634595     1.
 ↪68841986
>>> #    4 C(Fragment=1)        6.0000     0.31519183     1.41684566     2.
 ↪30745116
>>> #    5 C(Fragment=1)        6.0000     0.28773462     2.80126911     2.
 ↪08006241
>>> # ...
>>> is_nwchem("nwchem.out")
True
```

octadist.src.io.**get_coord_nwchem**(*f*)

Extract XYZ coordinate from NWChem output file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "NWChem-Fe-distorted-complex.out"
>>> atom, coord = get_coord_nwchem(file)
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**is_orca**(*f*)

Check if the input file is ORCA file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is ORCA output file, return True.

**See also:**

***get_coord_orca()*** Find atomic coordinates of molecule from ORCA file.

### Examples

```
>>> # orca.out
>>> # ---------------------------------
>>> # CARTESIAN COORDINATES (ANGSTROEM)
>>> # ---------------------------------
>>> #   C      0.009657    0.000000    0.005576
>>> #   C      0.009657   -0.000000    1.394424
>>> #   C      1.212436   -0.000000    2.088849
>>> #   C      2.415214    0.000000    1.394425
>>> #   C      2.415214   -0.000000    0.005575
>>> # ...
>>> is_orca("orca.out")
True
```

octadist.src.io.**get_coord_orca**(*f*)

Extract XYZ coordinate from ORCA output file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "ORCA-Fe-distorted-complex.out"
>>> atom, coord = get_coord_orca(file)
```

(continues on next page)

(continued from previous page)

```
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**is_qchem**(*f*)

Check if the input file is Q-Chem file format.

**Parameters** **f** (*str*) – User input filename.

**Returns** **bool** (*bool*) – If file is Q-Chem output file, return True.

**See also:**

***get_coord_qchem()*** Find atomic coordinates of molecule from Q-Chem file.

### Examples

```
>>> # qchem.out
>>> # ****************************
>>> # **  OPTIMIZATION CONVERGED  **
>>> # ****************************
>>> #                            Coordinates (Angstroms)
>>> #     ATOM                X                Y                Z
>>> #       1  C         0.2681746845   -0.8206222796   -0.3704019386
>>> #       2  C        -1.1809302341   -0.5901746612   -0.6772716414
>>> #       3  H        -1.6636318262   -1.5373167851   -0.9496501352
>>> #       4  H        -1.2829834971    0.0829227646   -1.5389938241
>>> #       5  C        -1.9678565203    0.0191922768    0.5346693165
>>> # ...
>>> is_qchem("qchem.out")
True
```

octadist.src.io.**get_coord_qchem**(*f*)

Extract XYZ coordinate from Q-Chem output file.

**Parameters** **f** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

### Examples

```
>>> file = "Qchem-Fe-distorted-complex.out"
>>> atom, coord = get_coord_qchem(file)
>>> atom
['Fe', 'O', 'O', 'N', 'N', 'N', 'N']
```

(continues on next page)

(continued from previous page)

```
>>> coord
array([[18.268051, 11.28912 ,  2.565804],
       [19.823874, 10.436314,  1.381569],
       [19.074466,  9.706294,  3.743576],
       [17.364238, 10.733354,  0.657318],
       [16.149538, 11.306661,  2.913619],
       [18.599941, 12.116308,  4.528988],
       [18.364987, 13.407634,  2.249608]])
```

octadist.src.io.**count_line**(*file=None*)

Count lines in an input file.

**Parameters** **file** (*str*) – Absolute or full path of input file.

**Returns** **i + 1** (*int*) – Number of line in file.

### Examples

```
>>> file = "[Fe(1-bpp)2][BF4]2-HS.xyz"
>>> count_line(file)
27
```

octadist.src.io.**extract_coord**(*file=None*)

Check file type, read data, extract atomic symbols and cartesian coordinate from a structure input file provided by the user. This function can efficiently manupulate I/O process. File types currently supported are listed in notes below. Other file formats can also be implemented easily within this module.

**Parameters** **file** (*str*) – User input filename.

**Returns**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.

**See also:**

***octadist.main.OctaDist.open_file()*** Open file dialog and to browse input file.

***octadist.main.OctaDist.search_coord()*** Search octahedral structure in complex.

### Notes

The following are file types supported by the current virsion of OctaDist:

- CIF
- XYZ
- Gaussian
- NWChem
- ORCA
- Q-Chem

### Examples

```
>>> file = "[Fe(1-bpp)2][BF4]2-HS.xyz"
>>> atom, coord = extract_coord(file)
>>> atom
['Fe', 'N', 'N', 'N', 'N', 'N', 'N', 'C', 'C']
>>> coord
array([[-1.95348286e+00,  4.51770478e+00,  1.47855811e+01],
       [-1.87618286e+00,  4.48070478e+00,  1.26484811e+01],
       [-2.18698286e+00,  4.34540478e+00,  1.69060811e+01],
       [-4.88286000e-03,  3.69060478e+00,  1.42392811e+01],
       [-1.17538286e+00,  6.38340478e+00,  1.56457811e+01],
       [-2.75078286e+00,  2.50260478e+00,  1.51806811e+01],
       [-3.90128286e+00,  5.27750478e+00,  1.40814811e+01],
       [-6.14953418e+00,  8.30666180e+00,  2.91361978e+01],
       [-8.59995241e+00,  7.11630815e+00,  4.52898814e+01]])
```

octadist.src.io.**find_metal**(*atom=None*, *coord=None*)

Count the number of metal center atom in complex.

**Parameters**

- **atom** (*list or None*) – Full atomic labels of complex. Default is None.
- **coord** (*array_like or None*) – Full atomic coordinates of complex. Default is None.

**Returns**

- **total_metal** (*int*) – The total number of metal center atom.
- **atom_metal** (*list*) – Atomic labels of metal center atom.
- **coord_metal** (*array_like*) – Atomic coordinates of metal center atom.

**See also:**

**octadist.src.elements.check_atom()** Convert atomic number to atomic symbol and vice versa.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'N', 'N', 'N']
>>> coord = [[-1.95348286e+00,  4.51770478e+00,  1.47855811e+01],
             [-1.87618286e+00,  4.48070478e+00,  1.26484811e+01],
             [-3.90128286e+00,  5.27750478e+00,  1.40814811e+01],
             [-4.88286000e-03,  3.69060478e+00,  1.42392811e+01],
             [-2.18698286e+00,  4.34540478e+00,  1.69060811e+01],
             [-1.17538286e+00,  6.38340478e+00,  1.56457811e+01],
             [-2.75078286e+00,  2.50260478e+00,  1.51806811e+01]]
>>> total_metal, atom_metal, coord_metal = find_metal(atom, coord)
>>> total_metal
1
>>> atom_metal
['Fe']
>>> coord_metal
array([[-1.95348286,  4.51770478, 14.7855811 ]])
```

octadist.src.io.**extract_octa**(*atom=None*, *coord=None*, *metal=1*, *cutoff_metal_ligand=2.8*)

Search the octahedral structure in complex and return atoms and coordinates.

**Parameters**

- **atom** (*list*) – Full atomic labels of complex.
- **coord** (*array_like*) – Full atomic coordinates of complex.
- **metal** (*int*) – Number of metal atom that will be taken as center atom for finding atomic coordinates of octahedral structure of interest. Default is 1.
- **cutoff_metal_ligand** (*float, optional*) – Cutoff distance for screening metal-ligand bond. Default is 2.8.

**Returns**

- **atom_octa** (*list*) – Atomic labels of octahedral structure.
- **coord_octa** (*array_like*) – Atomic coordinates of octahedral structure.

**See also:**

***find_metal()*** Find metals in complex.

***octadist.main.OctaDist.search_coord()*** Search octahedral structure in complex.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'N', 'N', 'N', 'C', 'C']
>>> coord = [[-1.95348286e+00,  4.51770478e+00,  1.47855811e+01],
             [-1.87618286e+00,  4.48070478e+00,  1.26484811e+01],
             [-3.90128286e+00,  5.27750478e+00,  1.40814811e+01],
             [-4.88286000e-03,  3.69060478e+00,  1.42392811e+01],
             [-2.18698286e+00,  4.34540478e+00,  1.69060811e+01],
             [-1.17538286e+00,  6.38340478e+00,  1.56457811e+01],
             [-2.75078286e+00,  2.50260478e+00,  1.51806811e+01],
             [-6.14953418e+00,  8.30666180e+00,  2.91361978e+01],
             [-8.59995241e+00,  7.11630815e+00,  4.52898814e+01]]
>>> atom_octa, coord_octa = extract_octa(atom, coord)
>>> atom_octa
['Fe', 'N', 'N', 'N', 'N', 'N', 'N']
>>> coord_octa
array([[-1.95348286e+00,  4.51770478e+00,  1.47855811e+01],
       [-1.87618286e+00,  4.48070478e+00,  1.26484811e+01],
       [-2.18698286e+00,  4.34540478e+00,  1.69060811e+01],
       [-4.88286000e-03,  3.69060478e+00,  1.42392811e+01],
       [-1.17538286e+00,  6.38340478e+00,  1.56457811e+01],
       [-2.75078286e+00,  2.50260478e+00,  1.51806811e+01],
       [-3.90128286e+00,  5.27750478e+00,  1.40814811e+01]])
```

## octadist.linear

octadist.src.linear.**angle_sign**(*v1*, *v2*, *direct*)

Compute angle between two vectors with sign and return value in degree.

**Parameters**

- **v1** (*array_like*) – Vector in 3D space.
- **v2** (*array_like*) – Vector in 3D space.
- **direct** (*array*) – Vector that refers to orientation of the plane.

**Returns** **angle** (*float64*) – Angle between two vectors in degree unit with sign.

**See also:**

**`calc.calc_theta()`** Calculate theta parameter.

### Examples

```
>>> vector1 = [1.21859514, -0.92569245, -0.51717955]
>>> vector2 = [1.02186387,  0.57480095, -0.95220433]
>>> direction = [1.29280503, 0.69301873, 1.80572438]
>>> angle_sign(vector1, vector2, direction)
60.38697927455357
```

octadist.src.linear.**angle_btw_vectors**(*v1*, *v2*)

Compute angle between two vectors and return value in degree.

**Parameters**

- **v1** (*array_like*) – Vector in 3D space.
- **v2** (*array_like*) – Vector in 3D space.

**Returns** **angle** (*float64*) – Angle between two vectors in degree unit.

### Examples

```
>>> vector1 = [-0.412697, -0.357008, -1.788172]
>>> vector2 = [-0.550839,  1.799178, -0.039114]
>>> angle_btw_vectors(vector1, vector2)
95.62773246517462
```

octadist.src.linear.**angle_btw_planes**(*a1*, *b1*, *c1*, *a2*, *b2*, *c2*)

Find the angle between 2 planes in 3D and return value in degree.

```
General equation of plane:

a*X + b*Y + c*Z + d = 0
```

**Parameters**

- **b1, c1** (*a1,*) – Coefficient of the equation of plane 1.
- **b2, c2** (*a2,*) – Coefficient of the equation of plane 2.

**Returns** **angle** (*float64*) – Angle between 2 planes in degree unit.

### Examples

```
>>> # Plane 1
>>> a1 = -3.231203733528
>>> b1 = -0.9688526458499996
>>> c1 = 0.9391692927779998
>>> # Plane 2
>>> a2 = 1.3904813057000005
```

(continues on next page)

(continued from previous page)

```
>>> b2 = 3.928502357473003
>>> c2 = -4.924114034864001
>>> angle_btw_planes(a1, b1, c1, a2, b2, c2)
124.89920902358416
```

octadist.src.linear.**triangle_area**(*a*, *b*, *c*)

Calculate the area of the triangle using the cross product:

```
Area = abs(ab X ac)/2

where vector ab = b - a and vector ac = c - a.
```

**Parameters**

- **a** (*array_like*) – 3D Coordinate of point.
- **b** (*array_like*) – 3D Coordinate of point.
- **c** (*array_like*) – 3D Coordinate of point.

**Returns** **area** (*float64*) – The triangle area.

### Examples

```
>>> # Three vertices
>>> a = [2.298354000, 5.161785000, 7.971898000]
>>> b = [1.885657000, 4.804777000, 6.183726000]
>>> c = [1.747515000, 6.960963000, 7.932784000]
>>> triangle_area(a, b, c)
1.7508135235821773
```

## octadist.plane

octadist.src.plane.**find_eq_of_plane**(*x*, *y*, *z*)

Find the equation of plane of given three points using cross product:

```
The general form of plane equation:

Ax + By + Cz = D

where A, B, C, and D are coefficient.

XZ  X  XY = (a, b, c)

d = (a, b, c).Z
```

**Parameters**

- **x** (*array_like*) – 3D Coordinate of point.
- **y** (*array_like*) – 3D Coordinate of point.
- **z** (*array_like*) – 3D Coordinate of point.

**Returns**

- **a** (*float64*) – Coefficient of the equation of the plane.
- **b** (*float64*) – Coefficient of the equation of the plane.
- **c** (*float64*) – Coefficient of the equation of the plane.
- **d** (*float64*) – Coefficient of the equation of the plane.

### Examples

```
>>> N1 = [2.298354000, 5.161785000, 7.971898000]
>>> N2 = [1.885657000, 4.804777000, 6.183726000]
>>> N3 = [1.747515000, 6.960963000, 7.932784000]
>>> a, b, c, d = find_eq_of_plane(N1, N2, N3)
>>> a
-3.231203733528
>>> b
-0.9688526458499996
>>> c
0.9391692927779998
>>> d
-4.940497273569501
```

octadist.src.plane.**find_fit_plane**(*coord*)

Find best fit plane to the given data points (atoms).

**Parameters** **coord** (*array_like*) – Coordinates of selected atom chunk.

**Returns**

- **xx** (*float*) – Coefficient of the surface.
- **yy** (*float*) – Coefficient of the surface.
- **z** (*float*) – Coefficient of the surface.
- **abcd** (*tuple*) – Coefficient of the equation of the plane.

**See also:**

**scipy.optimize.minimize()** Used to find the least-square plane.

### Examples

```
>>> points = [(1.1, 2.1, 8.1),
              (3.2, 4.2, 8.0),
              (5.3, 1.3, 8.2),
              (3.4, 2.4, 8.3),
              (1.5, 4.5, 8.0),
              (5.5, 6.7, 4.5)
              ]
>>> # To plot the plane, run following commands:
>>> import matplotlib.pyplot as plt
>>> # map coordinates for scattering plot
>>> xs, ys, zs = zip(*points)
>>> plt.scatter(xs, ys, zs)
>>> plt.show()
```

### octadist.plot

**class** octadist.src.plot.**Plot**(**args*, *name1='Var1'*, *name2='Var2'*)
Relationship plot between Zeta and Sigma parameters.

**Parameters**

- **args[0]** (*list*) – List of data set 1 (data1).
- **= list** (*args[0]*) – List of data set 2 (data2).
- **= str, optional** (*name2*) – Name of data set 1.
- **= str, optional** – Name of data set 2.

#### Examples

```
>>> data1 = [1, 2, 3, 4, 5]
>>> data2 = [1, 2, 3, 4, 5]
>>> test = Plot(data1, data2, name1="Data 1", name2="Data 2")
>>> test.add_point()
>>> test.add_text()
>>> test.add_legend()
>>> test.show_plot()
```

**start_plot**()
Start plot.

**add_point**()
Add all atoms to show in figure.

**add_text**()
Added text to show in figure.

**add_legend**()
Add legend to show in figure.

**config_plot**()
Config structure of figure.

**set_label**()
Set title of figure and axis labels.

**static save_img**(*save='Image_saved_by_OctaDist'*, *file='png'*)
Save figure as an image.

**Parameters**

- **save** (*str*) – Name of image file. Default is "Complex_saved_by_OctaDist".
- **file** (*file*) – Image type. Default is "png".

**static show_plot**()
Show plot.

### octadist.popup

octadist.src.popup.**err_no_file**()
Show this error when no input files uploaded/opened.

octadist.src.popup.**err_invalid_ftype**()
Show this error popup when file type is not supported by the program.

octadist.src.popup.**err_no_coord**(*i*)
Show this error popup when the program cannot read the atomic coordinates of complex inside the file or cannot extract the coordinates from the complex.

This will happen only if the input has no the proper format of atomic coordinates.

**Parameters** **i** (*int*) – Number of file.

octadist.src.popup.**err_less_ligands**(*i*)
Show this error popup when the complex has ligand atoms less that six atoms.

**Parameters** **i** (*int*) – Number of file.

octadist.src.popup.**err_no_metal**()
Show this error popup when the complex has no transition metal atom.

octadist.src.popup.**err_no_calc**()
Show this error popup when the user requests function that the results are required, but the results have not been computed yet.

octadist.src.popup.**err_only_2_files**()
Show this error popup when having not uploaded two complexes for using the RMSD function.

octadist.src.popup.**err_not_equal_atom**()
Show this error popup when the total number of atoms of two complexes are not equal.

octadist.src.popup.**err_atom_not_match**(*line*)
show this error popup when atomic symbol of two similar complexes does not match.

**Parameters** **line** (*int*) – The line number that atomic symbol does not match.

octadist.src.popup.**err_many_files**()
Show this error popup when user has loaded too many files.

octadist.src.popup.**err_wrong_format**()
Show this error popup when user has loaded the file that is not supported by OctaDist.

octadist.src.popup.**err_no_editor**()
Show this error popup if text editor path is empty.

octadist.src.popup.**err_visualizer_not_found**()
Show this error popup if user-defined visualizer is not available.

octadist.src.popup.**err_cannot_update**()
Show this error popup when the program cannot detect the operating system that the user is using.

octadist.src.popup.**info_save_results**(*file*)
show this info popup when an output file has been saved successfully.

**Parameters** **file** (*str*) – Absolute or full path of saved output file.

octadist.src.popup.**info_new_update**()
Show this info popup when new version is available for update.

octadist.src.popup.**info_using_dev**()
Show this info popup if user is using a development build version.

octadist.src.popup.**info_no_update**()
Show this info popup if program is the latest version.

octadist.src.popup.**warn_no_metal**(*i*)
Show this warning popup if no transition metal was found.

**Parameters** **i** (*int*) – Number of file.

octadist.src.popup.**warn_not_octa**()
Show this warning popup if the complex is non-octahedral structure.

### octadist.projection

octadist.src.projection.**project_atom_onto_line**(*p*, *a*, *b*)
Find the point projection on the line, which defined by two distinct end points.

```
a <----> b

P(x) = x1 + (p - x1).(x2 - x1)/(x2-x1).(x2-x1) * (x2-x1)
```

**Parameters**

- **p** (*array_like*) – Coordinate of point to project.
- **a** (*array_like*) – Coordinate of head atom of the line.
- **b** (*array_like*) – Coordinate of tail atom of the line.

**Returns** **projected_point** (*array_like*) – The projected point on the orthogonal line.

#### Examples

```
>>> # point to project
>>> p = [10.1873, 5.7463, 5.615]
>>> # head and end points of line
>>> a = [8.494, 5.9735, 4.8091]
>>> b = [9.6526, 6.4229, 7.3079]
>>> project_atom_onto_line(p, a, b)
[9.07023235 6.19701012 6.05188388]
```

octadist.src.projection.**project_atom_onto_plane**(*p*, *a*, *b*, *c*, *d*)
Find the orthogonal vector of point onto the given plane. The equation of plane is `Ax + By + Cz = D` and point is `(L, M, N)`, then the location on the plane that is closest to the point `(P, Q, R)` is

```
(P, Q, R) = (L, M, N) + λ * (A, B, C)

where λ = (D - ( A*L + B*M + C*N)) / (A^2 + B^2 + C^2).
```

**Parameters**

- **p** (*array_like*) – Point to project.
- **a** (*int or float*) – Coefficient of the equation of the plane.
- **b** (*int or float*) – Coefficient of the equation of the plane.
- **c** (*int or float*) – Coefficient of the equation of the plane.
- **d** (*int or float*) – Coefficient of the equation of the plane.

**Returns** **projected_point** (*array_like*) – The projected point on the orthogonal plane.

### Examples

```
>>> # point to project
>>> p = [10.1873, 5.7463, 5.615]
>>> # coefficient of the equation of the plane
>>> a = -3.231203733528
>>> b = -0.9688526458499996
>>> c = 0.9391692927779998
>>> d = -4.940497273569501
>>> project_atom_onto_plane(p, a, b, c, d)
[2.73723598 3.51245316 7.78040705]
```

## octadist.scripting

**class** octadist.src.scripting.**ScriptingConsole**(*root*)

Start scripting interface for an interactive code.

User can access to class variable (dynamic variable).

| Output box |
|---|
| Input box |

**Parameters** **root** (*object*) – Passing self object from another class to this class as root argument.

**See also:**

**settings** Program settings.

### Examples

```
>>> import tkinter as tk
>>> master = tk.Tk()
>>> console = ScriptingConsole(master)
>>> console.scripting_start()
```

**scripting_start**()

Start scripting console.

**script_run_help**()

Show help messages.

**script_run_list**()

Show list of commands in scripting run.

**script_run_info**()

Show info of program.

**script_run_doc**()

Show document of program.

**script_run_show**(*args*)

Show value of variable that user requests.

**Parameters** **args** (*str*) – Arbitrary argument.

**script_run_type**(*args*)
Show data type of variable.

**Parameters** **args** (*str*) – Arbitrary argument.

**script_run_set**(*args*)
Set new value to variable.

**Parameters** **args** (*str*) – Arbitrary argument.

**script_run_clear**()
Clear output box.

**script_run_clean**(*args*)
Clear output box and clean variable.

**script_run_restore**()
Restore all default settings.

**script_run_history**()
Show history of command.

**script_no_command**(*command*)
Show statement if command not found.

**Parameters** **command** (*str*) – Command that user submits.

**script_execute**(*event*)
Execute input command scripting.

**Parameters** **event** (*object*) – Object for button interaction

### octadist.structure

**class** octadist.src.structure.**DataComplex**(*master=None*, *icon=None*)
Show info of input complex.

**Parameters**

- **master** (*object, optional*) – If None, use tk.Tk(). If not None, use tk.Toplevel(master).
- **icon** (*str, optional*) – If None, use tkinter default icon. If not None, use user-defined icon.

#### Examples

```
>>> file = "File_1"
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> my_app = DataComplex()
>>> my_app.add_name(file)
>>> my_app.add_coord(atom, coord)
```

**start_app**()
Start application.

**add_name**(*file_name*)
Add file name to box.

**Parameters** **file_name** (*array_like*) – List containing the names of all input files.

**add_coord**(*atom*, *coord*)
Add atomic symbols and coordinates to box.

**Parameters**

- **atom** (*array_like*) – Atomic labels of full complex.
- **coord** (*array_like*) – Atomic coordinates of full complex.

**class** octadist.src.structure.**StructParam**(*master=None*, *icon=None*)
Show structural parameters of structure.

**Parameters**

- **master** (*object, optional*) – If None, use tk.Tk(). If not None, use tk.Toplevel(master).
- **icon** (*str, optional*) – If None, use tkinter default icon. If not None, use user-defined icon.

### Examples

```
>>> metal = 'Fe'
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> my_app = StructParam()
>>> my_app.add_metal(metal)
>>> my_app.add_coord(atom, coord)
```

**start_app**()
Start application.

**add_number**(*number*)
Add file number to box.

**Parameters** **number** (*int*) – File number.

**add_metal**(*metal*)
Add metal atom to box:

**Parameters** **metal** (*str*) – Metal atom.

**add_coord**(*atom*, *coord*)
Add atomic symbols and coordinates to box.

**Parameters**

- **atom** (*array_like*) – Atomic labels of full complex.

- **coord** (*array_like*) – Atomic coordinates of full complex.

**class** octadist.src.structure.**SurfaceArea**(*master=None*, *icon=None*)

Find the area of the faces of octahedral structure.

Three ligand atoms are vertices of triangular face

**Parameters**

- **master** (*object, optional*) – If None, use tk.Tk(). If not None, use tk.Toplevel(master).
- **icon** (*str, optional*) – If None, use tkinter default icon. If not None, use user-defined icon.

### Examples

```
>>> metal = 'Fe'
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> my_app = SurfaceArea()
>>> my_app.add_metal(metal)
>>> my_app.add_octa(coord)
```

**start_app**()

Start application.

**add_number**(*number*)

Add file number to box.

**Parameters** **number** (*int*) – File number.

**add_metal**(*metal*)

Add metal atom to box:

**Parameters** **metal** (*str*) – Metal atom.

**add_octa**(*coord*)

Add atomic coordinates of octahedron and find triangle area of the faces.

**Parameters** **coord** (*array_like*) – Atomic coordinates of octahedral structure.

**See also:**

***octadist.src.util.find_faces_octa()*** Find all faces of octahedron.

## octadist.tools

**class** octadist.src.tools.**CalcJahnTeller**(*atom*, *coord*, *cutoff_global=2.0*, *cutoff_hydrogen=1.2*, *master=None*, *icon=None*)

Calculate angular Jahn-Teller distortion parameter[1].

[1] J. M. Holland, J. A. McAllister, C. A. Kilner, M. Thornton-Pett, A. J. Bridgeman, M. A. Halcrow. Stereochemical effects on the spin-state transition shown by salts of [FeL2]2+ [L = 2,6-di(pyrazol-1-yl)pyridine]. J. Chem. Soc., Dalton Trans., 2002, 548-554. DOI: 10.1039/B108468M.

**Parameters**

- **atom** (*array_like*) – Atomic labels of full complex.
- **coord** (*array_like*) – Atomic coordinates of full complex.
- **master** (*None, object*) – If None, use tk.Tk(). If not None, use tk.Toplevel(master).
- **cutoff_global** (*int or float*) – Global cutoff for screening bonds. Default is 2.0.
- **cutoff_hydrogen** (*int or float*) – Cutoff for screening hydrogen bonds. Default is 1.2.
- **icon** (*str, optional*) – If None, use tkinter default icon. If not None, use user-defined icon.

### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> test = CalcJahnTeller(atom=atom, coord=coord)
>>> test.start_app()
>>> test.find_bond()
>>> test.show_app()
```

### References

**start_app**()
Start application.

**find_bond**()
Find bonds.

**See also:**

*octadist.src.util.find_bonds()* Find atomic bonds.

**pick_atom**(*group*)
On-mouse pick atom and get XYZ coordinate.

**Parameters** **group** (*str*) – Group A or B.

**plot_fit_plane**()
Display complex and two fit planes of two sets of ligand in molecule.

**clear_text**()
Clear text in box A & B.

**show_app**()
Show application.

**class** octadist.src.tools.**CalcRMSD**(*coord_1*, *coord_2*, *atom_1=None*, *atom_2=None*, *master=None*, *icon=None*)

Calculate root mean squared displacement of atoms in complex, RMSD[2].

**Parameters**

- **coord_1** (*array_like*) – Atomic coordinates of structure 1.
- **coord_2** (*array_like*) – Atomic coordinates of structure 2.
- **atom_1** (*list or tuple, optional*) – Atomic symbols of structure 1.
- **atom_2** (*list or tuple, optional*) – Atomic symbols of structure 2. If no atom_2 specified, assign it with None.

**Returns**

- **rmsd_normal** (*float*) – Normal RMSD.
- **rmsd_translate** (*float*) – Translate RMSD (re-centered).
- **rmsd_rotate** (*float*) – Kabsch RMSD (rotated).

### References

### Examples

```
>>> # Example of structure 1
>>> comp1 = [[10.1873, 5.7463, 5.615],
             [8.494, 5.9735, 4.8091],
             [9.6526, 6.4229, 7.3079],
             [10.8038, 7.5319, 5.1762],
             [9.6229, 3.9221, 6.0083],
             [12.0065, 5.5562, 6.3497],
             [10.8046, 4.9471, 3.9219]]
```

```
>>> # Example of structure 1
>>> comp2 = [[12.0937, 2.4505, 3.4207],
             [12.9603, 2.2952, 1.7286],
             [13.4876, 1.6182, 4.4230],
             [12.8522, 4.3174, 3.9894],
             [10.9307, 0.7697, 2.9315],
             [10.7878, 2.2987, 5.1071],
             [10.6773, 3.7960, 2.5424]]
```

```
>>> test = CalcRMSD(coord_1=comp1, coord_2=comp2)
>>> test.calc_rmsd()
>>> test.rmsd_normal
6.758144
>>> test.rmsd_translate
0.305792
>>> test.rmsd_rotate
0.277988
```

**start_app**()

[2] J. C. Kromann. https://github.com/charnley/rmsd.

**show_coord**()
Show atomic coordinates in box.

**calc_rmsd**()
Calculate normal, translated, and rotated RMSD.

**calc_and_show**()
Execute calc_rmsd function to calculate RMSD and show results in box.

**show_app**()
Show application.

### octadist.util

octadist.src.util.**find_bonds**(*atom*, *coord*, *cutoff_global=2.0*, *cutoff_hydrogen=1.2*)
Find all bond distance and filter the possible bonds.

- Compute distance of all bonds
- Screen bonds out based on global cutoff distance
- Screen H bonds out based on local cutoff distance

**Parameters**

- **atom** (*list*) – List of atomic labels of molecule.
- **coord** (*list*) – List of atomic coordinates of molecule.
- **cutoff_global** (*int or float*) – Global cutoff for screening bonds. Default is 2.0.
- **cutoff_hydrogen** (*int or float*) – Cutoff for screening hydrogen bonds. Default is 1.2.

**Returns**

- **filtered_pair_2** (*list*) – List of pair of atoms of selected bonds in molecule after screening
- **filtered_bond_2** (*array_like*) – Array of bond distances of selected bonds in molecule after screening.

#### Examples

```
>>> atom = ['Fe', 'N', 'N', 'N', 'O', 'O', 'O']
>>> coord = [[2.298354000, 5.161785000, 7.971898000],
             [1.885657000, 4.804777000, 6.183726000],
             [1.747515000, 6.960963000, 7.932784000],
             [4.094380000, 5.807257000, 7.588689000],
             [0.539005000, 4.482809000, 8.460004000],
             [2.812425000, 3.266553000, 8.131637000],
             [2.886404000, 5.392925000, 9.848966000]]
>>> pair_bond, bond_dist = find_bonds(atom, coord)
>>> pair_bond
[['Fe', 'N'], ['Fe', 'N'], ['Fe', 'N'], ['Fe', 'O'], ['Fe', 'O'], ['Fe', 'O']]
>>> bond_dist
[[[2.298354 5.161785 7.971898]
  [1.885657 4.804777 6.183726]]
 [[2.298354 5.161785 7.971898]
  [1.747515 6.960963 7.932784]]
```

(continues on next page)

(continued from previous page)

```
 [[2.298354 5.161785 7.971898]
  [4.09438  5.807257 7.588689]]
 [[2.298354 5.161785 7.971898]
  [0.539005 4.482809 8.460004]]
 [[2.298354 5.161785 7.971898]
  [2.812425 3.266553 8.131637]]
 [[2.298354 5.161785 7.971898]
  [2.886404 5.392925 9.848966]]]
```

octadist.src.util.**find_faces_octa**(*c_octa*)

Find the eight faces of octahedral structure.

```
1. Choose 3 atoms out of 6 ligand atoms. The total number of combination is 20.
2. Orthogonally project metal center atom onto the face: m ----> m'
3. Calculate the shortest distance between original metal center to its projected␣
↪point.
4. Sort the 20 faces in ascending order of the shortest distance.
5. Delete 12 faces that closest to metal center atom (first 12 faces).
6. The remaining 8 faces are the (reference) face of octahedral structure.
7. Find 8 opposite faces.

Reference plane              Opposite plane
   [[1 2 3],                    [[4 5 6],
    [1 2 4],        --->         [3 5 6],
      ...                          ...
    [2 3 5]]                     [1 4 6]]
```

**Parameters** **c_octa** (*array_like*) – Atomic coordinates of octahedral structure.

**Returns**

- **a_ref_f** (*list*) – Atomic labels of reference face.
- **c_ref_f** (*array_like*) – Atomic coordinates of reference face.
- **a_oppo_f** (*list*) – Atomic labels of opposite face.
- **c_oppo_f** (*array_like*) – Atomic coordinates of opposite face.

**See also:**

***octadist.src.plane.find_eq_of_plane()*** Find the equation of the plane.

***octadist.src.projection.project_atom_onto_plane()*** Orthogonal projection of point onto the plane.

### Examples

```
>>> coord = [[14.68572, 18.49228, 6.66716],
             [14.86476, 16.48821, 7.43379],
             [14.44181, 20.59400, 6.21555],
             [13.37473, 17.23453, 5.45099],
             [16.26114, 18.54903, 8.20527],
             [13.04897, 19.25464, 7.93122],
             [16.09157, 18.96170, 5.02956]]
>>> a_ref, c_ref, a_oppo, c_oppo = find_faces_octa(coord)
```

(continues on next page)

(continued from previous page)

```
>>> a_ref
[[1, 3, 6], [1, 4, 6], [2, 3, 6], [2, 3, 5],
 [2, 4, 5], [1, 4, 5], [1, 3, 5], [2, 4, 6]]
>>> c_ref
[[[14.86476 16.48821  7.43379]
  [13.37473 17.23453  5.45099]
  [16.09157 18.9617   5.02956]],
 ...,
 ...,
 [[14.44181 20.594    6.21555]
  [16.26114 18.54903  8.20527]
  [16.09157 18.9617   5.02956]]]
>>> a_octa
[[2, 4, 5], [2, 3, 5], [1, 4, 5], [1, 4, 6],
 [1, 3, 6], [2, 3, 6], [2, 4, 6], [1, 3, 5]]
>>> c_octa
[[[14.44181 20.594    6.21555]
  [16.26114 18.54903  8.20527]
  [13.04897 19.25464  7.93122]],
 ...,
 ...,
 [[14.86476 16.48821  7.43379]
  [13.37473 17.23453  5.45099]
  [13.04897 19.25464  7.93122]]]
```

## 4.10 Development

OctaDist is written entirely in Python 3 binding to Tkinter toolkit. We have been developing OctaDist with the ease of use and flexibility. In the current version, it supports both of a graphical user interface (GUI) and a command line interface (CLI) version. The first one is mainly developed for the general end-users who are not familiar with command line, while the latter is primarily developed as a package which is appropriate for those who works with CLI. Having designed as a third party package, the command-line OctaDist version is an smart assistant helping with a wide range of your problems.

### 4.10.1 Contribution

To give a contribution on program development, please pull request on the OctaDist Github.

```
git clone https://github.com/OctaDist/OctaDist.git
git checkout nightly-build
git pull origin nightly-build
```

### 4.10.2 OctaDist Testing

When you have finished editing the source code of the program, you can use `setuptools` for testing OctaDist such as build and install. A `setup.py` file in top-level directory provides software testing as follows:

```
pip setup.py build
pip setup.py install
pip setup.py test
```

### 4.10.3 Bug report

If you found a bug in OctaDist, please submit it on issues page. We appreciate all help and contribution in getting program development.

### 4.10.4 Code maintenance

The source code of OctaDist is maintained on Github version control system. Both master revision and nightly development build have been being tested and deployed on Travis CI, a continuous integration service.

Source code on Github:

- Master (stable) version : github.com/OctaDist/OctaDist
- Nightly build version : github.com/OctaDist/OctaDist/tree/nightly-build

**Tip:** For OctaDist download stats, please go to https://octadist.github.io/stats.html.

## 4.11 Authors

The program is actively developed in international collaboration between the members of the Computational Chemistry Research Unit at Thammasat University, the Functional Materials & Nanotechnology CoE at Walailak University, Thailand, and the Switchable Molecules and Materials group at University of Bordeaux, France.


- **Rangsiman Ketkaew (Thammasat University, Thailand)** E-mail: rangsiman1993@gmail.com
- **Yuthana Tantirungrotechai (Thammasat University, Thailand)** E-mail: yt203y@gmail.com
- **David J. Harding (Walailak University, Thailand)** E-mail: hdavid@mail.wu.ac.th
- **Phimphaka Harding (Walailak University, Thailand)** E-mail: kphimpha@mail.wu.ac.th
- **Guillaume Chastanet (University of Bordeaux, France)** E-mail: Guillaume.Chastanet@icmcb.cnrs.fr
- **Philippe Guionneau (University of Bordeaux, France)** E-mail: Philippe.Guionneau@icmcb.cnrs.fr
- **Mathieu Marchivie (University of Bordeaux, France)** E-mail: mathieu.marchivie@icmcb.cnrs.fr


## 4.12 License



**Authors** Rangsiman Ketkaew, Yuthana Tantirungrotechai, David J. Harding, Phimphaka Harding, Mathieu Marchivie

**Version** 3.0.0 of 2021

# Python Module Index

## o

# Index

## A



## C

## D



## E



## F



## G



## I

## M



## N



## O



## P



## R



## S

## T



## W