PyGauss Documentation

Release 0.6.0

Chris Sewell

September 02, 2015

Contents

1 Contents **3**
- 1.1 System Description . . . 3
- 1.2 Quick Start . . . 4
- 1.3 Example Assessment . . . 5
- 1.4 Project Status . . . 17
- 1.5 Whats New . . . 18
- 1.6 Whats To Come . . . 19
- 1.7 User API . . . 20

2 License **37**

Python Module Index **39**

Author	Chris Sewell
Project Page	https://github.com/chrisjsewell/PyGauss
Conda Distro	https://conda.binstar.org/cjs14
PyPi Distro	https://pypi.python.org/pypi/pygauss
Communication	pygauss@googlegroups.com

PyGauss is intended as an interactive tool for supporting the lifecycle of a computational molecular chemistry investigation. From visual and analytical exploration, through to documentation and publication.

Intitally PyGauss has been designed for the purpose of examining one or more Gaussian quantum chemical computations, both **geometrically** and **electronically**. It is built on top of the cclib/chemview/chemlab suite of packages and python scientific stack though, and so should be extensible to other types of computational chemical analysis. PyGauss is primarily designed to be used interactively in the IPython Notebook.

As shown in the examples, a molecular optimisation can be assesed individually (much like in gaussview), but also as part of a group. The advantages of this package are then:

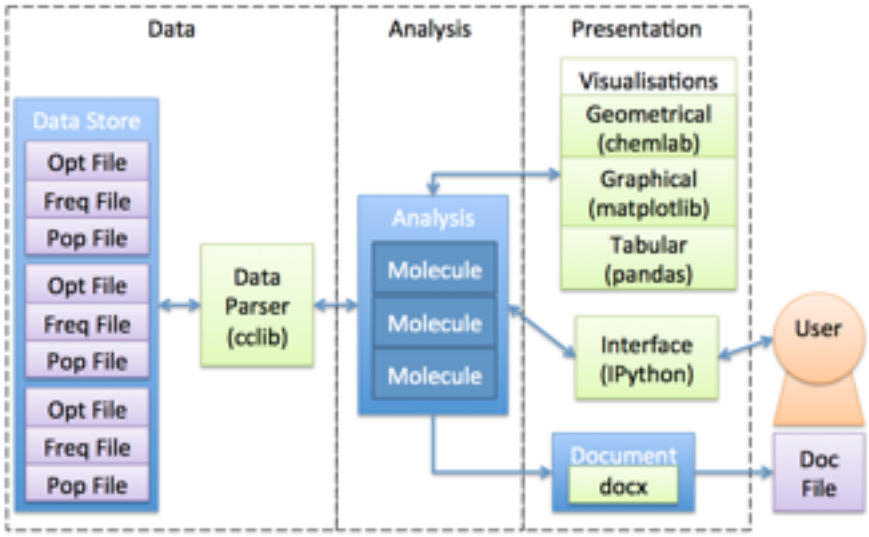
- Faster, more efficient analysis
- Extensible analysis
- Reproducible analysis

CHAPTER 1

Contents

1.1 System Description

PyGauss has been created as a computational tool for visual and data driven analysis of output from DFT computations. The package was created to fill a perceived gap in current analytical programme capabilities, whereby they are closed, Graphical User Interface (GUI) systems that do not allow for autonomous reproducibility or extensibility of analyses. For example, to compare multiple molecular conformers a user must open each data file separately, and then record and analyse the relevant data in an external programme.



PyGauss system diagram (blue=internal class, green=external dependency).

The figure above illustrates the systems model of PyGauss. It was designed in an object-orientated manner to have modularity, with well-defined and open interfaces, between the data parsing, analysis and presentation sub-systems. This allows the system to be flexible to different formats of required input and output data. Each molecular system is abstracted as a separate entity, with the Molecule class containing computational methods for analysing these individual systems. The Analysis class then contains methods for comparison of multiple Molecule systems. The Molecule/Analysis class interface with external sub-systems to format the data as required and present it to the user.

The system was realised with the Python programming language, which is widely used in the scientific community and has a number of existing, well-developed scientific packages. The cclib package has been utilised as a data parser,[1] initially this has focussed on parsing files output from the Gaussian programme (e.g. optimisation, frequency and population computations) but is extensible to many other DFT programmes. The Numpy package is used to carry out statistical computations,[2] whilst the Pandas, Matplotlib and chemlab packages are interfaced with to provide

visualisation of the data.[3-5] Finally the IPython package can be used to provide the user with an interactive interface to the system from where they can create extensible and reusable analysis.[6]

The source code is housed in an open-source repository. The code was written in a test-driven development manner, whereby each computational function is written to pass a number of small defined tests, for example to return a pre-calculated output value for a set of given inputs. The unit tests are automatically run each time code is added to the repository, reasonably ensuring the system is always validated to compute the correct outputs.

Automating the analysis in this manner can allow for additional information to be gained from the data, within the time restrictions of a project. For instance, analysis of the SOPT stabilisation energies to identify covalent aspects of H-bonding is not available in existing programmes.

1. (a) Lanaro. cclib documentation, http://cclib.github.io/

7. (a) van der Walt, S. C. Colbert and G. Varoquaux, Computing in Science Engineering, 2011, 13, 22-30, http://dx.doi.org/10.1109/MCSE.2011.37

19. (a) McKinney, Data Structures for Statistical Computing in Python, 2010, http://conference.scipy.org/proceedings/scipy2010/mckinney.html

23. (a) Hunter, Computing In Science & Engineering, 2007, 9, 90-95, http://dx.doi.org/10.5281/zenodo.15423.

10. (a) Lanaro. chemlab documentation, http://chemlab.readthedocs.org

7. (a) Perez and B. E. Granger, Computing in Science Engineering, 2007, 9, 21-29, http://dx.doi.org/10.1109/MCSE.2007.53

1.2 Quick Start

1.2.1 OSX and Linux

The recommended was to use pygauss is to download the Anaconda Scientific Python Distribution (64-bit). Once downloaded a new environment can be created in terminal and pygauss installed in one simple line:

```
conda create -n pg_env -c https://conda.binstar.org/cjs14 pygauss
```

1.2.2 Windows

There is currently no pygauss Conda distributable for Windows or for chemlab, which has C-extensions that need to be built using a compiler. Therefore chemlab will need to be cloned from GitHub, its extensions built, dependancies installed and finally install pygauss.

```
conda create -n pg_env python=2.7
conda install -n pg_env -c https://conda.binstar.org/cjs14 cclib
conda install -n pg_env -c https://conda.binstar.org/cjs14 chemview
conda install -n pg_env -c https://conda.binstar.org/cjs14 pyopengl
git clone --recursive https://github.com/chemlab/chemlab.git
cd chemlab
python setup.py build_ext --inplace
conda install -n pg_env <pil, pandas, matplotlib, scikit-learn, ...>
activate pg_env
pip install . # or add to PYTHONPATH
pip install pygauss
```

1.2.3 Troubleshooting

If you encounter difficulties it may be useful to look in working_conda_environments at conda environments known to work.

1.2.4 Testing

Pygauss utilises a unit test suite (nose/nose-parameterized) to ensure that computations run, and are correct. Continuous integration testing of the source code is provided by Travis CI and pass completion is an automated condition of the Conda build. These unit tests can also be run manually in the command line;

```
nosetests -v --with-doctest
```

or directly in python;

```
pygauss.run_nose(verbose=True)
```

1.3 Example Assessment

After installing PyGauss you should be able to open this IPython Notebook from; https://github.com/chrisjsewell/PyGauss/blob/master/Example_Assessment.ipynb, and run the following...

```
from IPython.display import display, Image
%matplotlib inline
import pygauss as pg
print 'pygauss version: {}'.format(pg.__version__)
```

```
pygauss version: 0.6.0
```

The test folder has a number of example Gaussian outputs to play around with.

```
folder = pg.get_test_folder()
len(folder.list_files())
```

```
33
```

Note: the *folder* object will act identical whether using a local path or one on a server over ssh (using paramiko):

```
folder = pg.Folder('/path/to/folder',
                   ssh_server='login.server.com',
                   ssh_username='username')
```

1.3.1 Single Molecule Analysis

A *molecule* can be created containg data about the inital geometry, optimisation process and analysis of the final configuration.

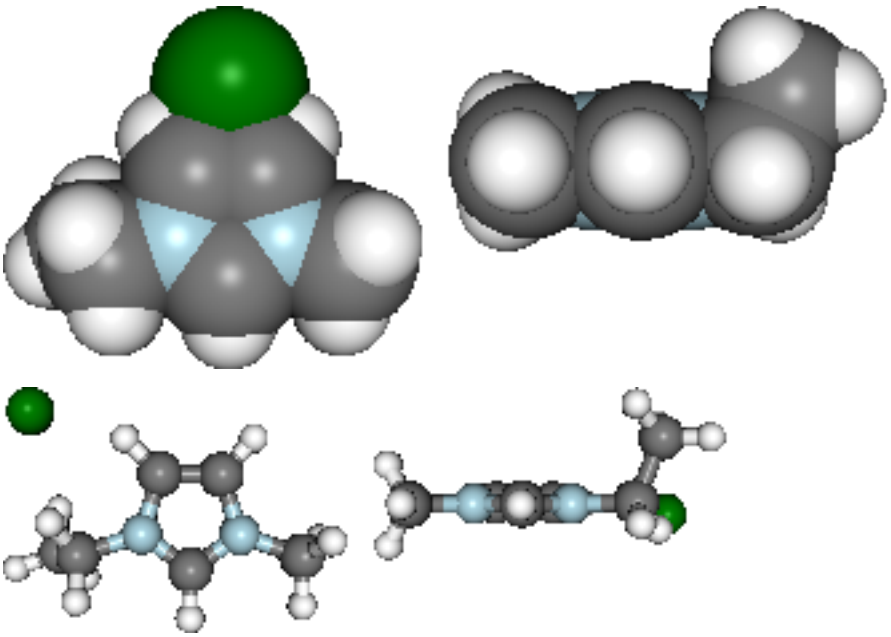
```
mol = pg.molecule.Molecule(folder_obj=folder,
                   init_fname='CJS1_emim-cl_B_init.com',
                   opt_fname=['CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_opt-modredundant_difrz.log',
                              'CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_opt-modredundant_difrz_err.log',
                              'CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_opt-modredundant_unfrz.log'],
                   freq_fname='CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_freq_unfrz.log',
                   nbo_fname='CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_pop-nbo-full-_unfrz.log',
```

```
                atom_groups={'emim':range(20), 'cl':[20]},
                alignto=[3,2,1])
```

Geometric Analysis

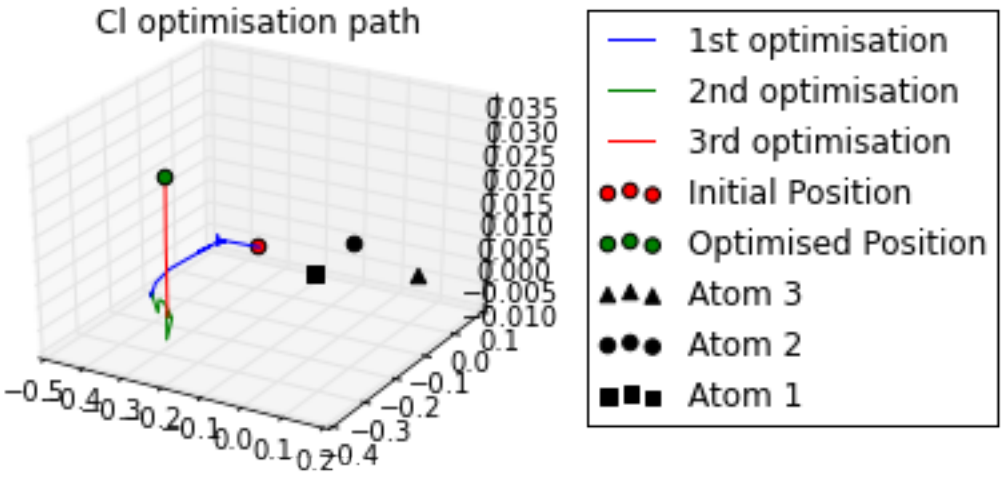
Molecules can be viewed statically or interactively.

```
#mol.show_initial(active=True)
vdw = mol.show_initial(represent='vdw', rotations=[[0,0,90], [-90, 90, 0]])
ball_stick = mol.show_optimisation(represent='ball_stick', rotations=[[0,0,90], [-90, 90, 0]])
display(vdw, ball_stick)
```



```
print 'Cl optimised polar coords from aromatic ring : ({0}, {1},{2})'.format(
    *[round(i, 2) for i in mol.calc_polar_coords_from_plane(20,3,2,1)])
ax = mol.plot_opt_trajectory(20, [3,2,1])
ax.set_title('Cl optimisation path')
ax.get_figure().set_size_inches(4, 3)
```

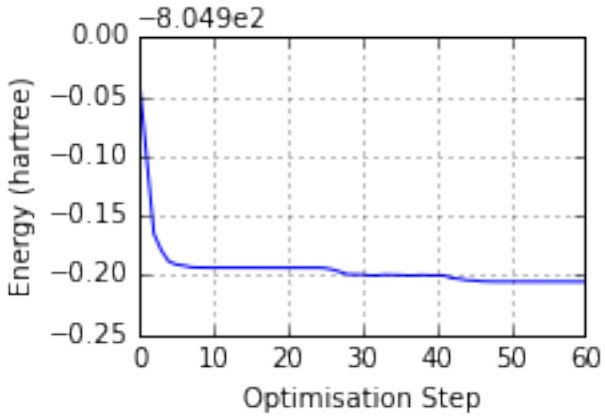
```
Cl optimised polar coords from aromatic ring : (0.11, -116.42,-170.06)
```

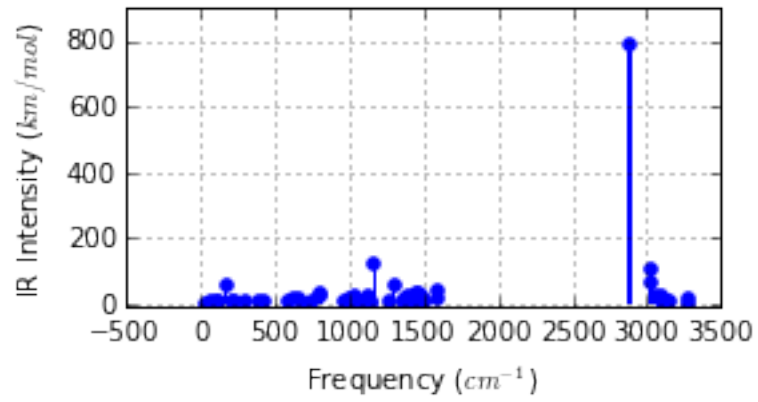


Energetics and Frequency Analysis

```
print('Optimised? {0}, Conformer? {1}, Energy = {2} a.u.'.format(
    mol.is_optimised(), mol.is_conformer(),
    round(mol.get_opt_energy(units='hartree'),3)))
ax = mol.plot_opt_energy(units='hartree')
ax.get_figure().set_size_inches(3, 2)
ax = mol.plot_freq_analysis()
ax.get_figure().set_size_inches(4, 2)
```

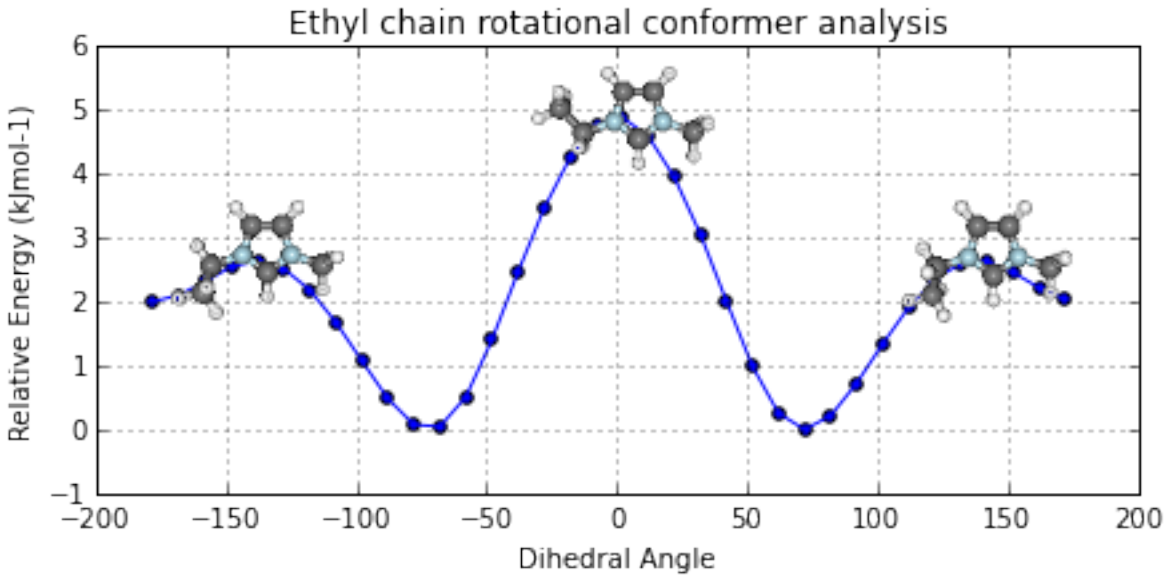
```
Optimised? True, Conformer? True, Energy = -805.105 a.u.
```





Potential Energy Scan analysis of geometric conformers...

```
mol2 = pg.molecule.Molecule(folder_obj=folder, alignto=[3,2,1],
            pes_fname=['CJS_emim_6311_plus_d3_scan.log',
                       'CJS_emim_6311_plus_d3_scan_bck.log'])
ax, data = mol2.plot_pes_scans([1,4,9,10], rotation=[0,0,90], img_pos='local_maxs', zoom=0.5)
ax.set_title('Ethyl chain rotational conformer analysis')
ax.get_figure().set_size_inches(7, 3)
```

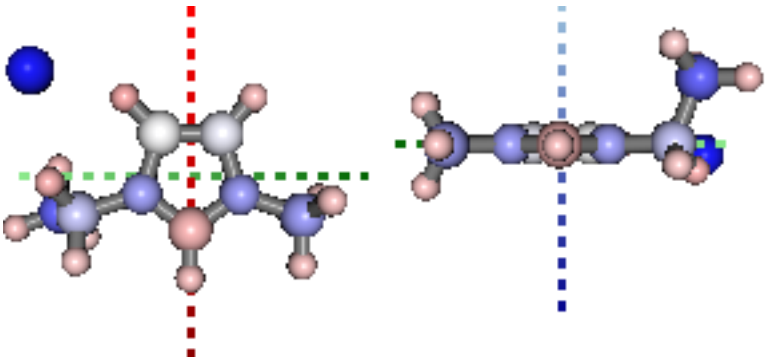


Partial Charge Analysis

using Natural Bond Orbital (NBO) analysis

```
print '+ve charge centre polar coords from aromatic ring: ({0} {1},{2})'.format(
    *[round(i, 2) for i in mol.calc_nbo_charge_center(3, 2, 1)])
display(mol.show_nbo_charges(represent='ball_stick', axis_length=0.4,
                             rotations=[[0,0,90], [-90, 90, 0]]))
```

```
+ve charge centre polar coords from aromatic ring: (0.02 -51.77,-33.15)
```

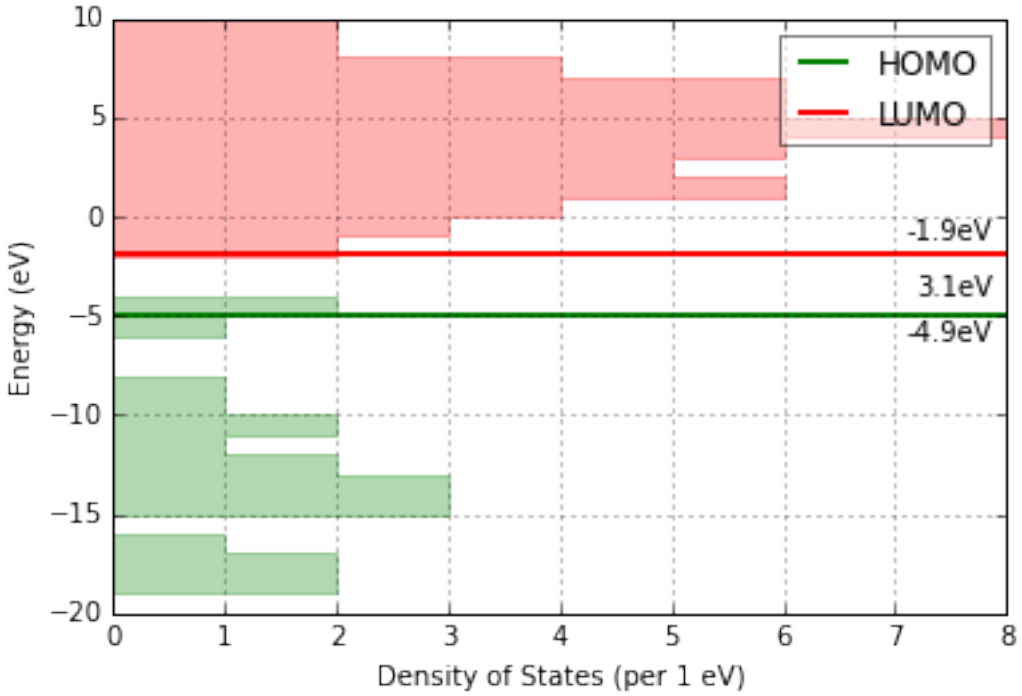


Density of States Analysis

```
print 'Number of Orbitals: {}'.format(mol.get_orbital_count())
homo, lumo = mol.get_orbital_homo_lumo()
homoe, lumoe = mol.get_orbital_energies([homo, lumo])
print 'HOMO at {} eV'.format(homoe)
print 'LUMO at {} eV'.format(lumoe)
```

```
Number of Orbitals: 272
HOMO at -4.91492036773 eV
LUMO at -1.85989816817 eV
```

```
ax = mol.plot_dos(per_energy=1, lbound=-20, ubound=10, legend_size=12)
```

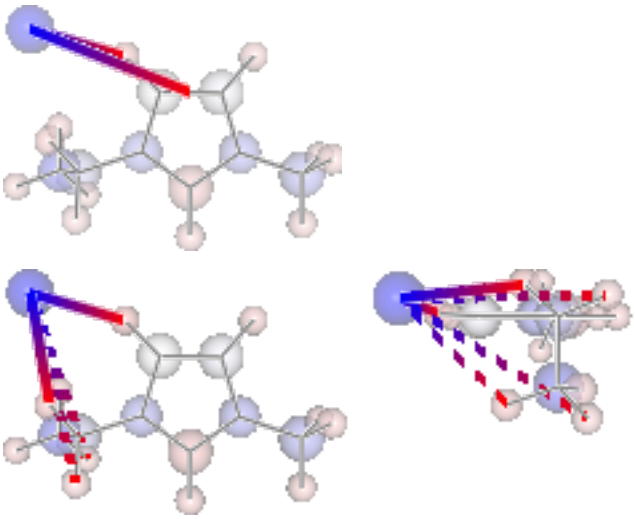


Bonding Analysis

Using Second Order Perturbation Theory.

```
print 'H inter-bond energy = {} kJmol-1'.format(
        mol.calc_hbond_energy(eunits='kJmol-1', atom_groups=['emim', 'cl']))
print 'Other inter-bond energy = {} kJmol-1'.format(
    mol.calc_sopt_energy(eunits='kJmol-1', no_hbonds=True, atom_groups=['emim', 'cl']))
display(mol.show_sopt_bonds(min_energy=1, eunits='kJmol-1',
                            atom_groups=['emim', 'cl'],
                            no_hbonds=True,
                            rotations=[[0, 0, 90]]))
display(mol.show_hbond_analysis(cutoff_energy=5.,alpha=0.6,
                                atom_groups=['emim', 'cl'],
                                rotations=[[0, 0, 90], [90, 0, 0]]))
```

```
H inter-bond energy = 111.7128 kJmol-1
Other inter-bond energy = 11.00392 kJmol-1
```



1.3.2 Multiple Computations Analysis

Multiple computations, for instance of different starting conformations, can be grouped into an *Analysis* class and anlaysed collectively.

```
analysis = pg.Analysis(folder_obj=folder)
errors = analysis.add_runs(headers=['Cation', 'Anion', 'Initial'],
                           values=[['emim'], ['cl'],
                                   ['B', 'BE', 'BM', 'F', 'FE']],
            init_pattern='*{0}-{1}_{2}_init.com',
            opt_pattern='*{0}-{1}_{2}_6-311+g-d-p-_gd3bj_opt*unfrz.log',
            freq_pattern='*{0}-{1}_{2}_6-311+g-d-p-_gd3bj_freq*.log',
            nbo_pattern='*{0}-{1}_{2}_6-311+g-d-p-_gd3bj_pop-nbo-full-*.log',
            alignto=[3,2,1], atom_groups={'emim':range(1,20), 'cl':[20]},
            ipython_print=True)
```

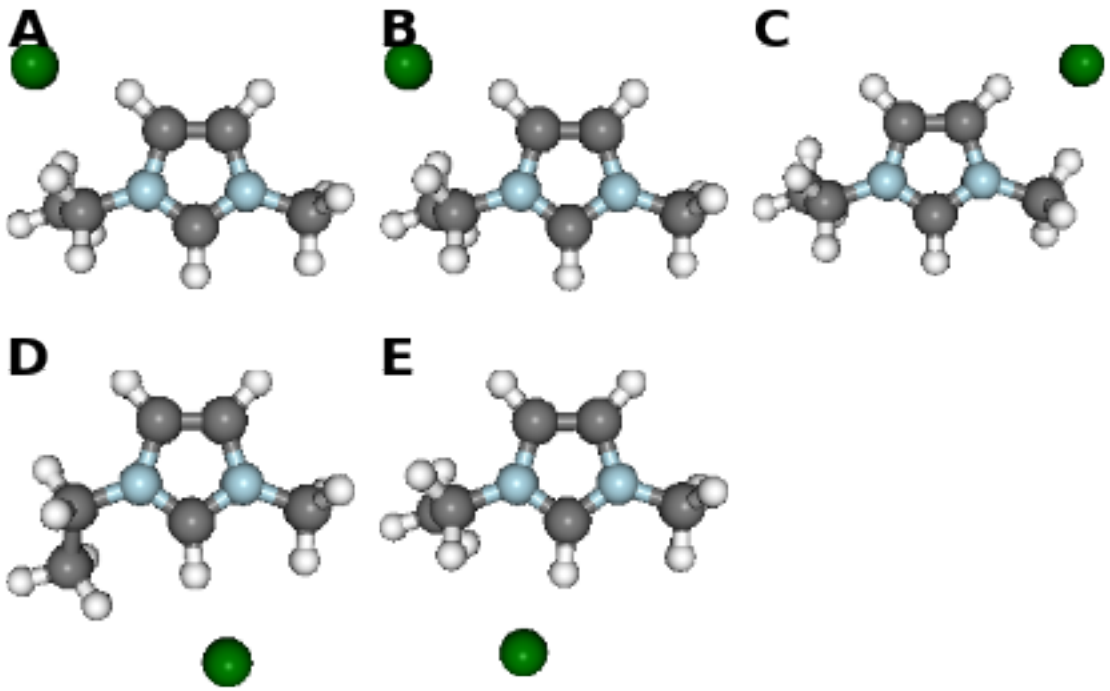
```
Reading data 5 of 5
```

Molecular Comparison

```
fig, caption = analysis.plot_mol_images(mtype='optimised', max_cols=3,
                        info_columns=['Cation', 'Anion', 'Initial'],
```

```
                          rotations=[[0,0,90]])
print caption
```

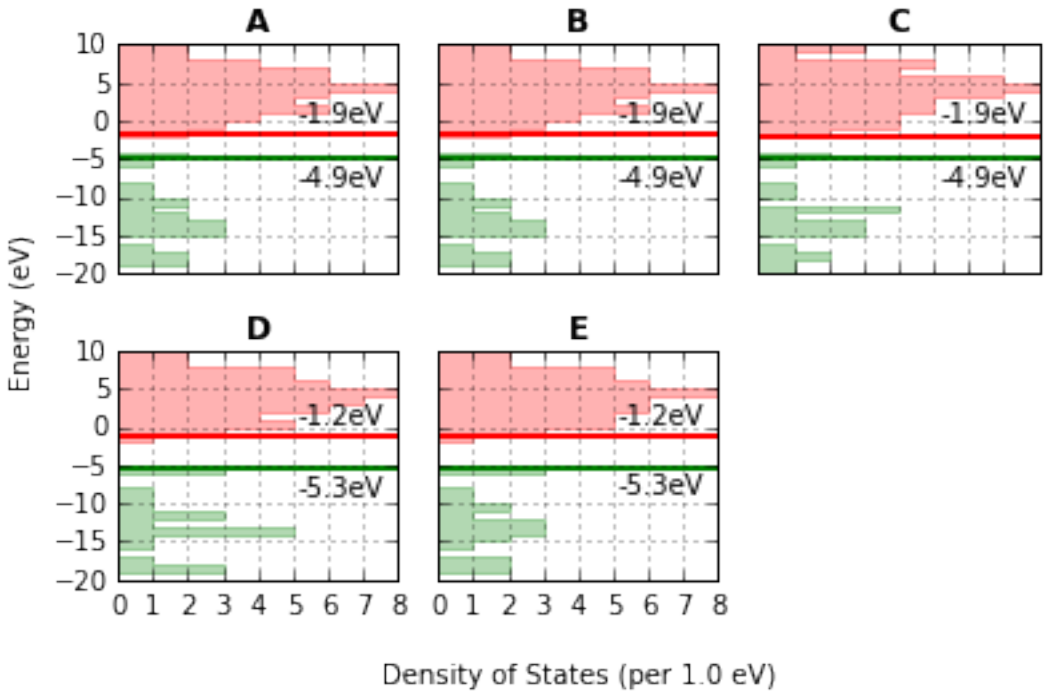
```
(A) emim, cl, B, (B) emim, cl, BE, (C) emim, cl, BM, (D) emim, cl, F, (E) emim, cl, FE
```



Data Comparison

```
fig, caption = analysis.plot_mol_graphs(gtype='dos', max_cols=3,
                                        lbound=-20, ubound=10, legend_size=0,
                                        band_gap_value=False,
                                    info_columns=['Cation', 'Anion', 'Initial'])
print caption
```

```
(A) emim, cl, B, (B) emim, cl, BE, (C) emim, cl, BM, (D) emim, cl, F, (E) emim, cl, FE
```



The methods mentioned for indivdiual molecules can be applied to all or a subset of these computations.

```
analysis.add_mol_property_subset('Opt', 'is_optimised', rows=[2,3])
analysis.add_mol_property('Energy (au)', 'get_opt_energy', units='hartree')
analysis.add_mol_property('Cation chain, $\\psi$', 'calc_dihedral_angle', [1, 4, 9, 10])
analysis.add_mol_property('Cation Charge', 'calc_nbo_charge', 'emim')
analysis.add_mol_property('Anion Charge', 'calc_nbo_charge', 'cl')
analysis.add_mol_property(['Anion-Cation, $r$', 'Anion-Cation, $\\theta$', 'Anion-Cation, $\\phi$'],
                          'calc_polar_coords_from_plane', 3, 2, 1, 20)
analysis.add_mol_property('Anion-Cation h-bond', 'calc_hbond_energy',
                          eunits='kJmol-1', atom_groups=['emim', 'cl'])
analysis.get_table(row_index=['Anion', 'Cation', 'Initial'],
                   column_index=['Cation', 'Anion', 'Anion-Cation'])
```

There is also an option (requiring pdflatex and ghostscript+imagemagik) to output the tables as a latex formatted image.

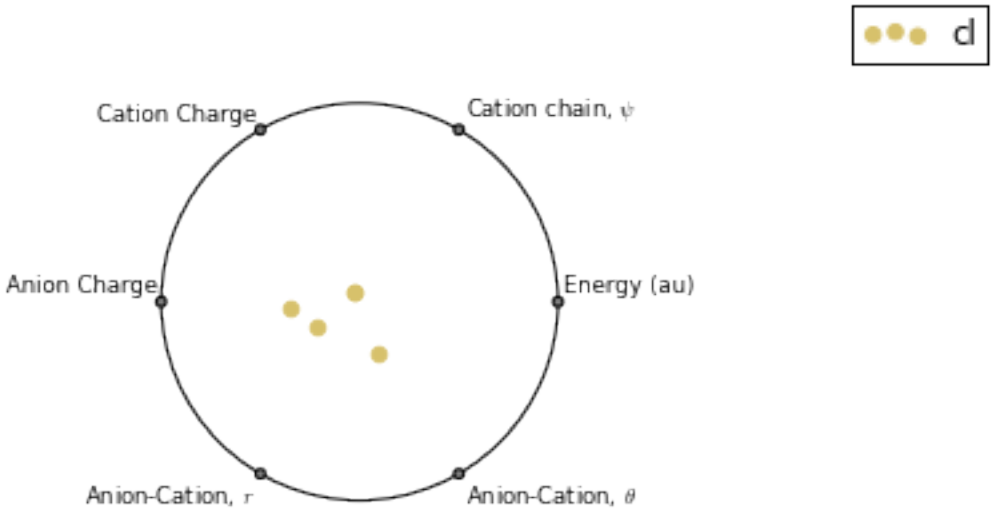
```
analysis.get_table(row_index=['Anion', 'Cation', 'Initial'],
                   column_index=['Cation', 'Anion', 'Anion-Cation'],
                   as_image=True, font_size=12)
```

Anion	Cation	Initial	Opt	Energy (au)	Cation chain, ψ	Charge	Anion Charge	Anion-Cation r	θ	ϕ	h-bond
cl	emim	B	-	-805.105	80.794	0.888	-0.888	0.420	-123.392	172.515	111.713
cl	emim	BE	-	-805.105	80.622	0.887	-0.887	0.420	-123.449	172.806	112.382
cl	emim	BM	True	-805.104	73.103	0.874	-0.874	0.420	124.121	-166.774	130.624
cl	emim	F	True	-805.118	147.026	0.840	-0.840	0.420	10.393	0.728	202.004
cl	emim	FE	-	-805.117	85.310	0.851	-0.851	0.417	-13.254	-4.873	177.360

Multi-Variate Analysis

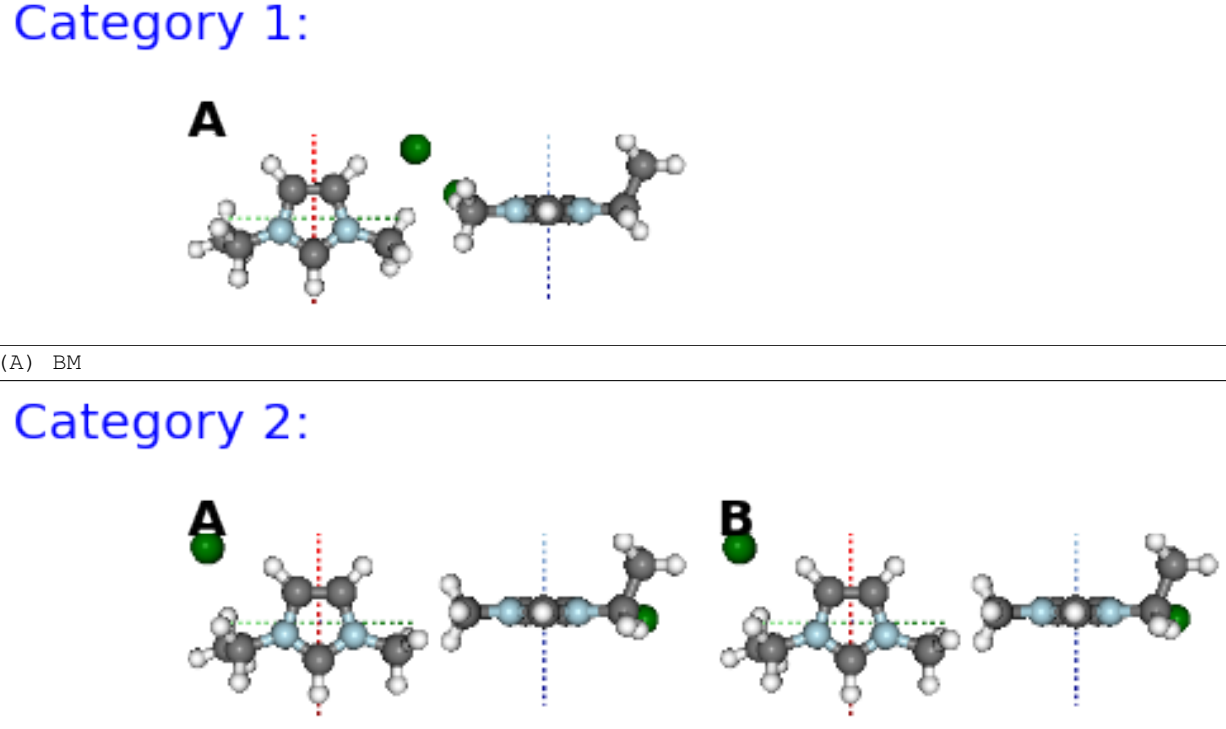
RadViz is a way of visualizing multi-variate data.

```
ax = analysis.plot_radviz_comparison('Anion', columns=range(4, 10))
```



The KMeans algorithm clusters data by trying to separate samples into n groups of equal variance.

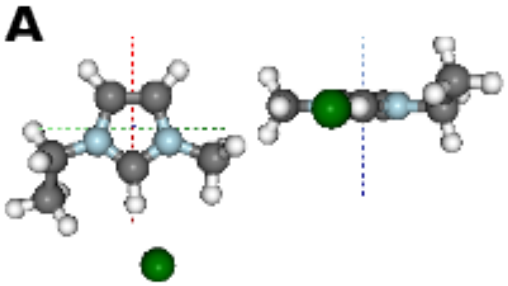
```
pg.utils.imgplot_kmean_groups(
    analysis, 'Anion', 'cl', 4, range(4, 10),
    output=['Initial'], mtype='optimised',
    rotations=[[0, 0, 90], [-90, 90, 0]],
    axis_length=0.3)
```




```
(A) BM
```

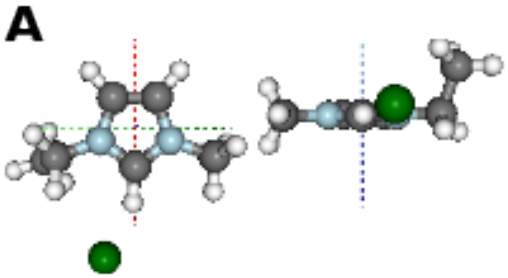
```
(A) B, (B) BE
```

Category 3:




```
(A) F
```

Category 4:

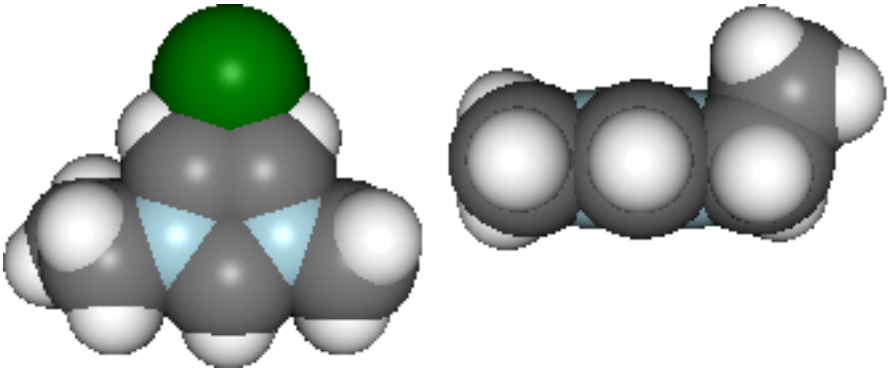



```
(A) FE
```

1.3.3 Documentation (MS Word)

After analysing the computations, it would be reasonable to want to document some of our findings. This can be achieved by outputting individual figure or table images via the folder object.

```
file_path = folder.save_ipyimg(vdw, 'image_of_molecule')
Image(file_path)
```



But you may also want to produce a more full record of your analysis, and this is where python-docx steps in. Building on this package the pygauss MSDocument class can produce a full document of your analysis.

```
import matplotlib.pyplot as plt
d = pg.MSDocument()
d.add_heading('A Pygauss Example Assessment', level=0)

d.add_docstring("""
# Introduction

We have looked at the following aspects
of [EMIM]^{+}[Cl]^{-} (C_{6}H_{11}ClN_{2});

- Geometric conformers
- Electronic structure

# Geometric Conformers
""")

fig, caption = analysis.plot_mol_images(max_cols=2,
                rotations=[[90,0,0], [0,0,90]],
                info_columns=['Anion', 'Cation', 'Initial'])
d.add_mpl(fig, dpi=96, height=9, caption=caption)
plt.close()
d.add_paragraph()
df = analysis.get_table(
        columns=['Anion Charge', 'Cation Charge'],
        row_index=['Anion', 'Cation', 'Initial'])
d.add_dataframe(df, incl_indx=True, style='Medium Shading 1 Accent 1',
                caption='Analysis of Conformer Charge')

d.add_docstring("""
# Molecular Orbital Analysis
## Density of States

It is **important** to *emphasise* that the
computations have only been run in the gas phase.
""")
fig, caption = analysis.plot_mol_graphs(gtype='dos', max_cols=3,
                                        lbound=-20, ubound=10, legend_size=0,
                                        band_gap_value=False,
                                    info_columns=['Cation', 'Anion', 'Initial'])
d.add_mpl(fig, dpi=96, height=9, caption=caption)
plt.close()

d.save('exmpl_assess.docx')
```

Which gives us the following:

A Pygauss Example Assessment

Introduction

We have looked at the following aspects of $[EMIM]^+[Cl]^-$ ($C_6H_{11}ClN_2$);

- Geometric conformers
- Electronic structure

Geometric Conformers

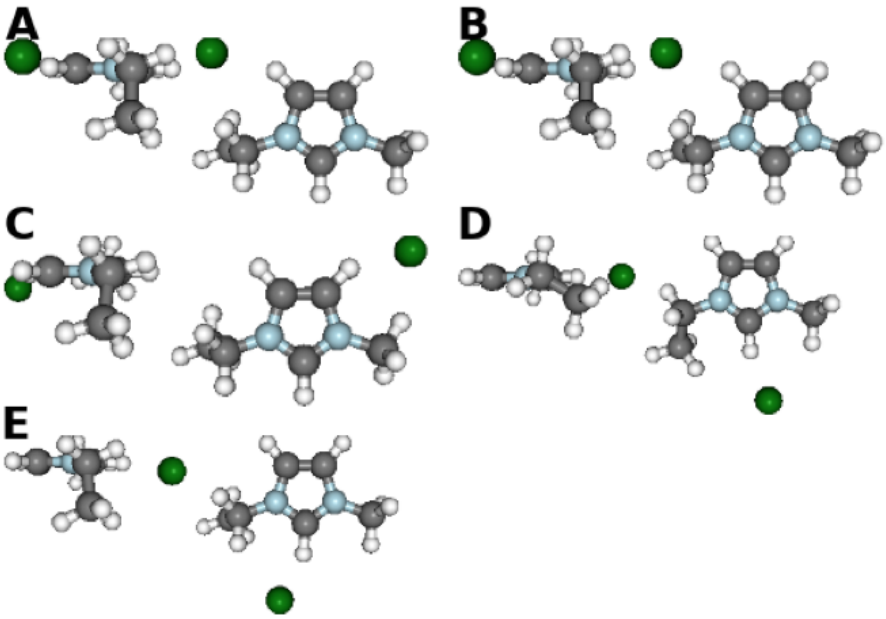


Figure 1: (A) cl, emim, B, (B) cl, emim, BE, (C) cl, emim, BM, (D) cl, emim, F, (E) cl, emim, FE

Anion	Cation	Initial	Anion Charge	Cation Charge
cl	emim	B	-0.88755	0.88756
cl	emim	BE	-0.88724	0.88723
cl	emim	BM	-0.87435	0.87436
cl	emim	F	-0.84037	0.84037
cl	emim	FE	-0.85079	0.8508

Table 1: Analysis of Conformer Charge

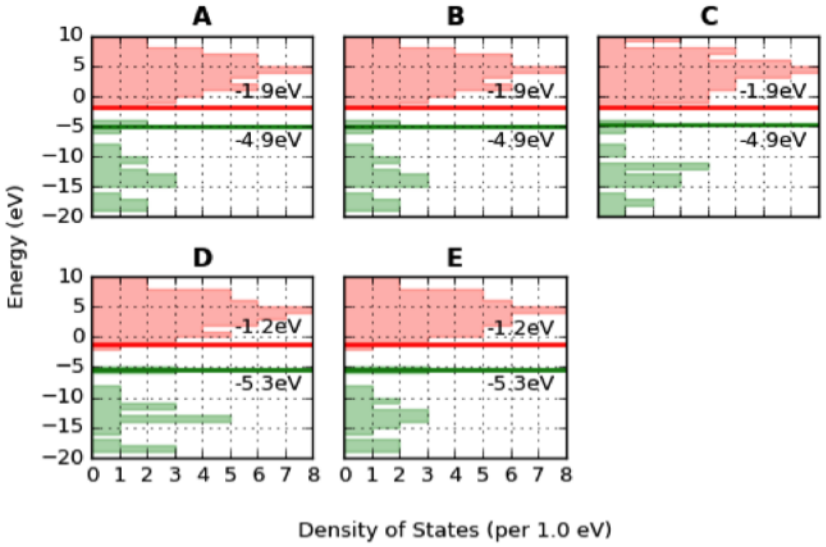


Figure 2: (A) emim, cl, B, (B) emim, cl, BE, (C) emim, cl, BM, (D) emim, cl, F, (E) emim, cl, FE

1.3.4 Further Examples

For more complex examples, you can also find a notebook from a recent project at; https://github.com/chrisjsewell/PyGauss/blob/master/Real_project_example_notebook.ipynb

1.4 Project Status

1.4.1 Distribution

Conda	
PyPi	
Documents	

1.4.2 Development

GitHub	
Unit Testing	
Testing Coverage	
Documents	

1.5 Whats New

1.5.1 v0.6.0 - Functional Improvements

- Added zero-point energy recovery *pygauss.molecule.Molecule.get_zeropt_energy()* and ability to correct in *pygauss.molecule.Molecule.get_opt_energy()*
- Added HOMO-LUMO overlay option in *pygauss.molecule.Molecule.plot_dos()* to visualise and assess band-gaps
- Added top-level functions for saving/loading objects (Molecule, Analysis) to a file; `pygauss.save_object()` and `pygauss.load_object()`
- other minor fixes

1.5.2 v0.5.1 - MSDocument Improvements

Improved the *pygauss.docs.MSDocument* class:

- Improved *pygauss.docs.MSDocument.add_markdown()* to handle more markdown syntax
- Added *pygauss.docs.MSDocument.add_docstring()* to add multiple paragraphs (with markdown)
- Improved *pygauss.docs.MSDocument.add_dataframe()* and *pygauss.docs.MSDocument.add_mpl()*
  - there is now a caption option with built-in figure/table number count

1.5.3 v0.5.0 - DoS Analysis & Multiprocessed I/O

- Density of States (DoS) and Partial Density of States (PDoS) analysis capabilities added.
- Addition of *pygauss.analysis.Analysis.plot_mol_graphs()* for data comparison
- *pygauss.analysis.Analysis.add_runs()* method now utilises the `multiprocessing` module for faster data input*
- Improvement of API documentation

Back-compatibility note: replaced *plot_optimisation_E* and *get_optimisation_E* with *pygauss.molecule.Molecule.get_opt_energy()* and *pygauss.molecule.Molecule.plot_opt_energy()*, to adhere with standard naming conventions.

*multiprocessing not currently available in Windows or over SSH

1.5.4 v0.4.4 - Output to MS Word Documents

- added the *pygauss.docs* module for outputting analysis to documents
  - initially utilising the `docx.document.Document` within *pygauss.docs.MSDocument*

1.5.5 v0.4.3 - Coninuous Integrated Testing

Addition of continuous integrated testing using [Travis](https://travis-ci.org/) and testing coverage analysis using [Coverall](https://coveralls.io/).

1.5.6 v0.4.2 - Addition of Documentation

addition of Sphinx documentation

1.5.7 v0.4.0 - Major Update

update includes:

- refactoring of data io
- improvement of second order perturbation theory analysis
- image output to table
- addition of unit test suite
- improvement of method documentation

breaks some back compatibility

1.5.8 v0.3.0 - File Input Over SSH

main update is the ability setup an ssh connection to a server, using the paramiko library, and parse analysis files over it. Also the ability to use wildcards (*) in input file names.

some minor back compatibility breaks

1.5.9 v0.2.2 - Table Image Improvements

Improvements to Table to Image functionality on OSx

- added some fixes
- re-organised test modules

1.5.10 v0.2.1 - Latex Table Images

addition of functionality to output analysis tables as latex images for input into projects!

1.5.11 v0.2 - Initial working distribution

Working distribution of pygauss to be converted to first conda package

1.5.12 v0.1 - First Version

the first version

1.6 Whats To Come

1.6.1 Natural Bonding Orbital Visualisation

Visualise natural binding orbitals, in particular HOMO and LUMO

1.6.2 Molecular Image Labelling

with atom numbers, atom types, bond lengths, etc...

1.7 User API

1.7.1 pygauss.file_io module

Created on Mon May 18 21:01:25 2015

@author: chris sewell

class pygauss.file_io.**Folder**(*path*, *server=None*, *username=None*, *passwrd=None*)
Bases: object

an object intended to act as an entry point to a folder path

Parameters

- **path** (*str*) – the path to the folder (absolute or relative)
- **server** (*str*) – the server name
- **username** (*str*) – the username to connect to the server
- **passwrd** (*str*) – server password, if not present it will be asked for during initialisation

__enter__()
use with statement to open ssh connection once

__exit__(*type*, *value*, *traceback*)
use with statement to open ssh connection once

active()
if folder is active

get_path()
get folder path

islocal()
if folder is local

list_files(*pattern=None*, *one_file=False*)
list files in folder

Parameters **pattern** (*str*) – a pattern the file must match that can include * wildcards

read_file(*file_name*)
return an open file ready for reading

save_ipyimg(*img*, *img_name*)
a function for outputing an IPython Image to a file

Parameters

- **img** (*IPython.display.Image*) – an IPyton image
- **img_name** (*str*) – the desired name of the file

save_mplfig(*fig*, *fig_name*, *dpi=256*, *format='png'*)
a function for outputing a matplotlib figure to a file

Parameters

- **fig** (*matplotlib.figure.Figure*) – a Matplotlib figure
- **fig_name** (*str*) – the desired name of the file

save_pilimg(*img*, *img_name*)

write_file(*file_name*, *overwrite=False*)
return an open file ready for writing to

class pygauss.file_io.**NoOutputFolder**(**args*, ***kwargs*)
Bases: *pygauss.file_io.Folder*

a folder object which will not output any data

save_ipyimg(**arg*, ***kwargs*)

save_mplfig(**arg*, ***kwargs*)

save_pilimg(**arg*, ***kwargs*)

write_file(**arg*, ***kwargs*)

1.7.2 pygauss.molecule module

Created on Fri May 01 21:24:31 2015

@author: chris

class pygauss.molecule.**Molecule**(*folderpath=''*, *init_fname=False*, *opt_fname=False*, *freq_fname=False*, *nbo_fname=False*, *pes_fname=False*, *fail_silently=False*, *atom_groups={}*, *alignto=[]*, *server=None*, *username=None*, *passwrd=None*, *folder_obj=None*)
Bases: object

a class to analyse gaussian input/output of a single molecular geometry

Parameters

- **folderpath** (*str*) – the folder path
- **init_fname** (*str*) – the intial geometry (.com) file
- **opt_fname** (*str or list of str*) – the optimisation log file
- **freq_fname** (*str*) – the frequency analysis log file
- **nbo_fname** (*str*) – the population analysis logfile
- **pes_fname** (*str*) – the potential energy scan logfile
- **fail_silently** (*bool*) – whether to raise an error if a file read fails (if True can use get_init_read_errors to see errors)
- **atom_groups** (*{str:[int, ...]}*) – groups of atoms that can be selected as a subset
- **alignto** (*[int, int, int]*) – the atom numbers to align the geometry to

Notes

any of the file names can have wildcards (e.g. 'filename*.log) in them, as long as this resolves to a single path in the directory

NB: nbo population analysis must be run with the GFInput flag to ensure data is output to the log file

add_frequency(*file_name*)

add_initialgeom(*file_name*)

add_nbo_analysis(*file_name*)

add_optimisation(*file_name*)

add_pes_analysis(*file_names*)

calc_2plane_angle(*p1*, *p2*, *optimisation=True*)
return angle of planes

calc_bond_angle(*indxs*, *optimisation=True*, *mol=None*)
Returns the angle in degrees between three points

calc_dihedral_angle(*indxs*, *optimisation=True*, *mol=None*)
Returns the angle in degrees between four points

calc_hbond_energy(*atom_groups=[]*, *eunits='kJmol-1'*)

calc_min_dist(*idx_list1*, *idx_list2*, *optimisation=True*, *units='nm'*, *ignore_missing=True*)
indexes start at 1

calc_nbo_charge(*atoms=[]*)
returns total charge of the atoms

calc_nbo_charge_center(*p1*, *p2*, *p3*, *positive=True*, *units='nm'*, *atoms=[]*)
returns the distance r amd angles theta, phi of the positive/negative charge center to the circumcenter of the plane formed by [p1, p2, p3]

the plane formed will have; x-axis along p1, y-axis anticlock-wise towards p2, z-axis normal to the plane

theta (azimuth) is the in-plane angle from the x-axis towards the y-axis phi (inclination) is the out-of-plane angle from the x-axis towards the z-axis

calc_opt_trajectory(*atom*, *plane=[]*)
calculate the trajectory of an atom as it is optimised, relative to a plane of three atoms

calc_polar_coords_from_plane(*p1*, *p2*, *p3*, *c*, *optimisation=True*, *units='nm'*)
returns the distance r and angles theta, phi of atom c to the circumcenter of the plane formed by [p1, p2, p3]

the plane formed will have; x-axis along p1, y-axis anticlock-wise towards p2, z-axis normal to the plane

theta (azimuth) is the in-plane angle from the x-axis towards the y-axis phi (inclination) is the out-of-plane angle from the x-axis towards the z-axis

calc_sopt_energy(*atom_groups=[]*, *eunits='kJmol-1'*, *no_hbonds=False*)
calculate total energy of interactions between "filled" (donor) Lewis-type Natural Bonding Orbitals (NBOs) and "empty" (acceptor) non-Lewis NBOs, using Second Order Perturbation Theory

Parameters

- **eunits** (*str*) – the units of energy to return
- **atom_groups** (*[list or str, list or str]*) – restrict interactions to between two lists (or identifiers) of atom indexes
- **no_hbonds** (*bool*) – whether to ignore H-Bonds in the calculation

Returns **analysis** – a table of interactions

Return type pandas.DataFrame

combine_molecules(*other_mol, self_atoms=False, other_atoms=False, self_rotation=[0, 0, 0], other_rotation=[0, 0, 0], self_transpose=[0, 0, 0], other_transpose=[0, 0, 0], self_opt=True, other_opt=True, charge=None, multiplicity=None, out_name=False, descript='', overwrite=False, active=False, represent='ball_stick', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, ipyimg=True, folder_obj=None*)
transpose in nanometers

get_atom_group(*group*)
return list of atoms in group

get_basis_descript()

get_basis_funcs()

get_folder()
return the Folder instance

get_freq_analysis()
return frequency analysis

Returns **data** – frequency data

Return type pandas.DataFrame

get_hbond_analysis(*min_energy=0.0*, *atom_groups=[]*, *eunits='kJmol-1'*)
EXPERIMENTAL! hydrogen bond analysis (DH—A), using Second Order Bond Perturbation Theiry

Parameters

- **min_energy** (*float*) – the minimum interaction energy to report
- **eunits** (*str*) – the units of energy to return
- **atom_groups** (*[list or str, list or str]*) – restrict interactions to between two lists (or identifiers) of atom indexes

Returns **analysis** – a table of interactions

Return type pandas.DataFrame

Notes

uses a strict definition of a hydrogen bond as: interactions between "filled" (donor) Lewis-type Lone Pair (LP) NBOs and "empty" (acceptor) non-Lewis Bonding (BD) NBOs

get_init_read_errors()
get read errors, recorded if fail_silently was set to True on initialise

get_opt_energy(*units='eV'*, *final=True*, *zpe_correct=False*)
return the SCF optimisation energy(s)

Parameters

- **units** (*str*) – the unit type of the energy
- **final** (*bool*) – return only the final optimised energy if True, else for all steps
- **zpe_correct** (*bool*) – apply zero-point energy correction (found in frequency log) to final optimised energy

Returns **energy** – dependant on final

Return type float or list of floats

get_orbital_count()
return number of orbitals

get_orbital_energies(*orbitals*, *eunits='eV'*)
the orbital energies for listed orbitals

Parameters

- **orbitals** (*int or iterable of ints*) – the orbital(s) to return energies for (starting at 1)
- **eunits** (*str*) – the units of energy

Returns **moenergies** – energy for each orbital

Return type numpy.array

get_orbital_homo_lumo()
return orbital numbers of homo and lumo

get_run_error(*rtype='opt'*)
True if there were errors in the computation, else False

get_sopt_analysis(*eunits='kJmol-1'*, *atom_groups=[]*, *charge_info=False*)
interactions between "filled" (donor) Lewis-type Natural Bonding Orbitals (NBOs) and "empty" (acceptor) non-Lewis NBOs, using Second Order Perturbation Theory (SOPT)

Parameters

- **eunits** (*str*) – the units of energy to return
- **atom_groups** (*[list or str, list or str]*) – restrict interactions to between two lists (or identifiers) of atom indexes
- **charge_info** (*bool*) – include charge info for atoms (under headings 'A_Charges' and 'D_Charges')

Returns **analysis** – a table of interactions

Return type pandas.DataFrame

get_zeropt_energy(*units='eV'*)
return the zero-point energy correction

Parameters **units** (*str*) – the unit type of the energy

Returns **energy** – zero-point energy correction

Return type float

is_conformer(*cutoff=0.0*)
False if any frequencies in the frequency analysis were negative

is_optimised()
was the geometry optimised

plot_dos(*eunits='eV'*, *per_energy=1.0*, *lbound=None*, *ubound=None*, *color_homo='g'*, *color_lumo='r'*, *homo_lumo_lines=True*, *homo_lumo_values=True*, *band_gap_value=True*, *legend_size=10*, *ax=None*)
plot Density of States and HOMO/LUMO gap

Parameters

- **eunits** (*str*) – unit of energy
- **per_energy** (*float*) – energy interval to group states by
- **lbound** (*float*) – lower bound energy

- **ubound** (*float*) – upper bound energy
- **color_homo** (*matplotlib.colors*) – color of homo in matplotlib format
- **color_lumo** (*matplotlib.colors*) – color of lumo in matplotlib format
- **homo_lumo_lines** (*bool*) – draw lines at HOMO and LUMO energies
- **homo_lumo_values** (*bool*) – annotate HOMO and LUMO lines with exact energy values
- **band_gap_value** (*bool*) – annotate inbetween HOMO and LUMO lines with band gap value
- **legend_size** (*int*) – the font size (in pts) for the legend
- **ax** (*matplotlib.Axes*) – an existing axes to plot the data on

Returns **plot** – plotted optimisation data

Return type matplotlib.axes.Axes

plot_freq_analysis(*color='blue'*, *alpha=1*, *marker_size=20*, *ax=None*)
plot frequency analysis

Returns **data** – plotted frequency data

Return type matplotlib.axes.Axes

plot_opt_energy(*units='eV'*, *linecolor='blue'*, *ax=None*)
plot SCF optimisation energy

Returns **data** – plotted optimisation data

Return type matplotlib.axes.Axes

plot_opt_trajectory(*atom*, *plane=[]*, *ax_lims=None*, *ax_labels=False*)
plot the trajectory of an atom as it is optimised, relative to a plane of three atoms

plot_pes_scans(*fixed_atoms, eunits='kJmol-1', img_pos='', rotation=[0.0, 0.0, 0.0], zoom=1, order=1*)
plot Potential Energy Scan

Parameters

- **img_pos** (<*''*,*'local_mins'*,*'local_maxs'*,*'global_min'*,*'global_max'*>) – position image(s) of molecule conformation(s) on plot
- **rotation** (*[float, float, float]*) – rotation of molecule image(s)

remove_alignment_atoms()

set_alignment_atoms(*idx1*, *idx2*, *idx3*)

show_active_orbital(*orbital*, *iso_value=0.03*, *alpha=0.5*, *bond_color=(255, 0, 0)*, *antibond_color=(0, 255, 0)*, *gbonds=True*)
get interactive representation of orbital

Parameters

- **orbital** (*int*) – the orbital to show (in range 1 to number of orbitals)
- **iso_value** (*float*) – The value for which the function should be constant.
- **alpha** – alpha value of iso-surface
- **bond_color** – color of bonding orbital surface in RGB format
- **antibond_color** – color of anti-bonding orbital surface in RGB format

- **gbonds** (*bool*) – guess bonds between atoms (via distance)

show_hbond_analysis(*min_energy=0.0, atom_groups=[], cutoff_energy=0.0, eunits='kJmol-1', bondwidth=5, gbonds=True, active=False, represent='ball_stick', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], relative=False, minval=-1, maxval=1, alpha=0.5, transparent=True, ipyimg=True*)

EXPERIMENTAL! hydrogen bond analysis DH—A

For a hydrogen bond to occur there must be both a hydrogen donor and an acceptor present. The donor in a hydrogen bond is the atom to which the hydrogen atom participating in the hydrogen bond is covalently bonded, and is usually a strongly electronegative atom such as N, O, or F. The hydrogen acceptor is the neighboring electronegative ion or molecule, and must posses a lone electron pair in order to form a hydrogen bond.

Since the hydrogen donor is strongly electronegative, it pulls the covalently bonded electron pair closer to its nucleus, and away from the hydrogen atom. The hydrogen atom is then left with a partial positive charge, creating a dipole-dipole attraction between the hydrogen atom bonded to the donor, and the lone electron pair on the acceptor.

show_highlight_atoms(*atomlists, transparent=False, alpha=0.7, gbonds=True, active=False, optimised=True, represent='vdw', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], ipyimg=True*)

show optimised geometry of molecule with certain atoms highlighted

show_initial(*gbonds=True, active=False, represent='vdw', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], ipyimg=True*)

show initial geometry (before optimisation) of molecule coloured by atom type

show_nbo_charges(*gbonds=True, active=False, relative=False, minval=-1, maxval=1, represent='vdw', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], ipyimg=True*)

show optimised geometry coloured by charge from nbo analysis

show_optimisation(*opt_step=False, gbonds=True, active=False, represent='vdw', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], ipyimg=True*)

show optimised geometry of molecule coloured by atom type

show_sopt_bonds(*min_energy=20.0, cutoff_energy=0.0, atom_groups=[], bondwidth=5, eunits='kJmol-1', no_hbonds=False, gbonds=True, active=False, represent='ball_stick', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], relative=False, minval=-1, maxval=1, alpha=0.5, transparent=True, ipyimg=True*)

visualisation of interactions between "filled" (donor) Lewis-type Natural Bonding Orbitals (NBOs) and "empty" (acceptor) non-Lewis NBOs, using Second Order Perturbation Theory

yield_orbital_images(*orbitals, iso_value=0.02, extents=(2, 2, 2), transparent=True, alpha=0.5, wireframe=True, bond_color=(255, 0, 0), antibond_color=(0, 255, 0), resolution=100, gbonds=True, represent='ball_stick', rotations=[[0.0, 0.0, 0.0]], zoom=1.0, width=300, height=300, axis_length=0, lines=[], ipyimg=True*)

yield orbital images

Parameters

- **orbitals** (*int or list of ints*) – the orbitals to show (in range 1 to number of orbitals)
- **iso_value** (*float*) – The value for which the function should be constant.
- **extents** ((*float, float, float*)) – +/- x,y,z to extend the molecule geometrty when constructing the surface

- **transparent** (*bool*) – whether iso-surface should be transparent (based on alpha value)
- **alpha** – alpha value of iso-surface
- **wireframe** – whether iso-surface should be wireframe (or solid)
- **bond_color** – color of bonding orbital surface in RGB format
- **antibond_color** – color of anti-bonding orbital surface in RGB format
- **resolution** (*int*) – The number of grid point to use for the surface. An high value will give better quality but lower performance.
- **gbonds** (*bool*) – guess bonds between atoms (via distance)
- **represent** (*str*) – representation of molecule ('none', 'wire', 'vdw' or 'ball_stick')
- **zoom** (*float*) – zoom level of images
- **width** (*int*) – width of original images
- **height** (*int*) – height of original images (although width takes precedent)
- **axis_length** (*float*) – length of x,y,z axes in negative and positive directions
- **lines** (*[start_coord, end_coord, start_color, end_color, width, dashed]*) – lines to add to image
- **ipyimg** (*bool*) – whether to return an IPython image, PIL image otherwise

Returns **mol** – an image of the molecule in the format specified by ipyimg

Return type IPython.display.Image or PIL.Image

pygauss.molecule.**orbit_z**(*self*, *angle*)

1.7.3 pygauss.analysis module

class pygauss.analysis.**Analysis**(*folderpath=''*, *server=None*, *username=None*, *passwrd=None*, *folder_obj=None*, *headers=[]*)

Bases: object

a class to analyse multiple computations

Parameters

- **folderpath** (*str*) – the folder directory storing the files to be analysed
- **server** (*str*) – the name of the server storing the files to be analysed
- **username** (*str*) – the username to connect to the server
- **passwrd** (*str*) – server password, if not present it will be asked for during initialisation
- **headers** (*list*) – the variable categories for each computation

add_basic_properties(*props=['basis', 'nbasis', 'optimised', 'conformer']*)
adds columns giving info of basic run properties

add_mol_property(*name*, *method*, **args*, ***kwargs*)
compute molecule property for all rows and create a data column

Parameters

- **name** (*str*) – what to name the data column
- **method** (*str*) – what molecule method to call

- ***args** – arguments to pass to the molecule method
- ****kwargs** – keyword arguments to pass to the molecule method

add_mol_property_subset(*name*, *method*, *rows=[]*, *filters={}*, *args=[]*, *kwargs={}*, *relative_to_rows=[]*)
compute molecule property for a subset of rows and create/add-to data column

Parameters

- **name** (*str or list of strings*) – name for output column (multiple if method outputs more than one value)
- **method** (*str*) – what molecule method to call
- **rows** (*list*) – what molecule rows to calculate the property for
- **filters** (*dict*) – filter for selecting molecules to calculate the property for
- **args** (*list*) – the arguments to pass to the molecule method
- **kwargs** (*dict*) – the keyword arguments to pass to the molecule method
- **relative_to_rows** (*list of ints*) – compute values relative to the summated value(s) of molecule at the rows listed

add_run(*identifiers={}*, *init_fname=None*, *opt_fname=None*, *freq_fname=None*, *nbo_fname=None*, *alignto=[]*, *atom_groups={}*, *add_if_error=False*, *folder_obj=None*)
add single Gaussian run input/outputs

add_runs(*headers=[]*, *values=[]*, *init_pattern=None*, *opt_pattern=None*, *freq_pattern=None*, *nbo_pattern=None*, *add_if_error=False*, *alignto=[]*, *atom_groups={}*, *ipython_print=False*, *folder_obj=None*)
add multiple Gaussian run inputs/outputs

calc_kmean_groups(*category_column*, *category_name*, *groups*, *columns=[]*, *rows=[]*, *filters={}*)
calculate the kmeans grouping of rows

The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

copy()

count_runs()
get number of runs held in analysis

folder
The folder for gaussian runs

get_basic_property(*prop*, **args*, ***kwargs*)
returns a series of a basic run property or nan if it is not available

Parameters **prop** (*str*) – can be 'basis', 'nbasis', 'optimised', 'opt_error' or 'conformer'

get_folder()

get_ids(*variable_names*, *variable_lists*)
return ids of a list of unique computations

get_molecule(*row*)
get molecule object coresponding to particular row

get_table(*rows=[]*, *columns=[]*, *filters={}*, *precision=4*, *head=False*, *mol=False*, *row_index=[]*, *column_index=[]*, *as_image=False*, *na_rep='-'*, *font_size=None*, *width=None*, *height=None*, *unconfined=False*)

return pandas table of requested data in requested format

rows [integer or list of integers] select row ids

columns [string/integer or list of strings/integers] select column names/positions

filters [dict] filter for rows with certain value(s) in specific columns

precision [int] decimal precision of displayed values

head [int] return only first n rows

mol [bool] include column containing the molecule objects

row_index [string or list of strings] columns to use as new index

column_index [list of strings] srings to place in to higher order column indexs

as_image [bool] output the table as an image (used pygauss.utils.df_to_img)

na_rep [str] how to represent empty (nan) cells (if outputting image)

width, height, unconfined [int, int, bool] args for IPy Image

Returns **df** – a table of data

Return type pandas.DataFrame

plot_mol_graphs(*gtype='energy'*, *share_plot=False*, *max_cols=1*, *padding=(1, 1)*, *tick_rotation=0*, *rows=[]*, *filters={}*, *sort_columns=[]*, *info_columns=[]*, *info_incl_id=False*, *letter_prefix=''*, *start_letter='A'*, *grid=True*, *sharex=True*, *sharey=True*, *legend_size=10*, *color_scheme='jet'*, *eunits='eV'*, *per_energy=1.0*, *lbound=None*, *ubound=None*, *color_homo='g'*, *color_lumo='r'*, *homo_lumo_lines=True*, *homo_lumo_values=True*, *band_gap_value=True*)

get a set of data plots for each molecule

Parameters

- **gtype** (*str*) – the type of plot, energy = optimisation energies, freq = frequency analsis, dos = Density of States,
- **share_plot** (*bool*) – whether to plot all data on the same or separate axes
- **max_cols** (*int*) – maximum columns on plots (share_plot=False only)
- **padding** (*tuple*) – padding between images (horizontally, vertically)
- **tick_rotation** (*int*) – rotation of x-axis labels
- **rows** (*int or list*) – index for the row of each molecule to plot (all plotted if empty)
- **filters** (*dict*) – {columns:values} to filter by
- **sort_columns** (*list of str*) – columns to sort by
- **info_columns** (*list of str*) – columns to use as info in caption
- **info_incl_id** (*bool*) – include molecule id number in labels
- **letter_prefix** (*str*) – prefix for labelling subplots (share_plot=False only)
- **start_letter** (*str*) – starting (capital) letter for labelling subplots (share_plot=False only)

- **grid** (*bool*) – whether to include a grid in the axes
- **sharex** (*bool*) – whether to align x-axes (share_plot=False only)
- **sharey** (*bool*) – whether to align y-axes (share_plot=False only)
- **legend_size** (*int*) – the font size (in pts) for the legend
- **color_scheme** (*str*) – the scheme to use for each molecule (share_plot=True only) according to http://matplotlib.org/examples/color/colormaps_reference.html
- **eunits** (*str*) – the units of energy to use
- **per_energy** (*float*) – energy interval to group states by (DoS only)
- **lbound** (*float*) – lower bound energy (DoS only)
- **ubound** (*float*) – upper bound energy (DoS only)
- **color_homo** (*matplotlib.colors*) – color of homo in matplotlib format
- **color_lumo** (*matplotlib.colors*) – color of lumo in matplotlib.colors
- **homo_lumo_lines** (*bool*) – draw lines at HOMO and LUMO energies
- **homo_lumo_values** (*bool*) – annotate HOMO and LUMO lines with exact energy values
- **band_gap_value** (*bool*) – annotate inbetween HOMO and LUMO lines with band gap value

Returns

- **data** (*matplotlib.figure.Figure*) – plotted frequency data
- **caption** (*str*) – A caption describing each subplot, given info_columns

plot_mol_images (*mtype='optimised', max_cols=1, padding=(1, 1), sort_columns=[], info_columns=[], info_incl_id=False, label_size=20, letter_prefix='', start_letter='A', rows=[], filters={}, align_to=[], rotations=[[0.0, 0.0, 0.0]], gbonds=True, represent='ball_stick', zoom=1.0, width=500, height=500, axis_length=0, relative=False, minval=-1, maxval=1, highlight=[], frame_on=False, eunits='kJmol-1', sopt_min_energy=20.0, sopt_cutoff_energy=0.0, atom_groups=[], alpha=0.5, transparent=False, hbondwidth=5, no_hbonds=False*)

show molecules in matplotlib table of axes

Parameters

- **mtype** – 'initial', 'optimised', 'nbo', 'highlight', 'highlight-initial', 'sopt' or 'hbond'
- **max_cols** (*int*) – maximum columns in plot
- **padding** (*tuple*) – padding between images (horizontally, vertically)
- **sort_columns** (*list of str*) – columns to sort by
- **info_columns** (*list of str*) – columns to use as info in caption
- **info_incl_id** (*bool*) – include molecule id number in caption
- **label_size** (*int*) – subplot label size (pts)
- **letter_prefix** (*str*) – prefix for labelling subplots
- **start_letter** (*str*) – starting (capital) letter for labelling subplots
- **rows** (*int or list*) – index for the row of each molecule to plot (all plotted if empty)

- **filters** (*dict*) – {columns:values} to filter by
- **align_to** (*[int, int, int]*) – align geometries to the plane containing these atoms
- **rotations** (*list of [float, float, float]*) – for each rotation set [x,y,z] an image will be produced
- **gbonds** (*bool*) – guess bonds between atoms (via distance)
- **represent** (*str*) – representation of molecule ('none', 'wire', 'vdw' or 'ball_stick')
- **zoom** (*float*) – zoom level of images
- **width** (*int*) – width of original images
- **height** (*int*) – height of original images (although width takes precedent)
- **axis_length** (*float*) – length of x,y,z axes in negative and positive directions
- **relative** (*bool*) – coloring of nbo atoms scaled to min/max values in atom set (for nbo mtype)
- **minval** (*float*) – coloring of nbo atoms scaled to absolute min (for nbo mtype)
- **maxval** (*float*) – coloring of nbo atoms scaled to absolute max (for nbo mtype)
- **highlight** (*list of lists*) – atom indxes to highlight (for highlight mtype)
- **eunits** (*str*) – the units of energy to return (for sopt/hbond mtype)
- **sopt_min_energy** (*float*) – minimum energy to show (for sopt/hbond mtype)
- **sopt_cutoff_energy** (*float*) – energy below which bonds will be dashed (for sopt mtype)
- **alpha** (*float*) – alpha color value of geometry (for sopt/hbond mtypes)
- **transparent** (*bool*) – whether atoms should be transparent (for sopt/hbond mtypes)
- **hbondwidth** (*float*) – width of lines depicting interaction (for hbond mtypes)
- **atom_groups** (*[list or str, list or str]*) – restrict interactions to between two lists (or identifiers) of atom indexes (for sopt/hbond mtypes)
- **no_hbonds** (*bool*) – whether to ignore H-Bonds in the calculation (for sopt only)
- **frame_on** (*bool*) – whether to show frame around each image

Returns

- **fig** (*matplotlib.figure.Figure*) – A figure containing subplots for each molecule image
- **caption** (*str*) – A caption describing each subplot, given info_columns

plot_radviz_comparison(*category_column*, *columns=[]*, *rows=[]*, *filters={}*, *point_size=30*, ***kwargs*)

return plot axis of radviz graph

RadViz is a way of visualizing multi-variate data. It is based on a simple spring tension minimization algorithm. Basically you set up a bunch of points in a plane. In our case they are equally spaced on a unit circle. Each point represents a single attribute. You then pretend that each sample in the data set is attached to each of these points by a spring, the stiffness of which is proportional to the numerical value of that attribute (they are normalized to unit interval). The point in the plane, where our sample settles to (where the forces acting on our sample are at an equilibrium) is where a dot representing our sample will be drawn. Depending on which class that sample belongs it will be colored differently.

remove_columns(*columns*)

remove_non_conformers (*cutoff=0.0*)
removes runs with negative frequencies

remove_non_optimised ()
removes runs that were not optimised

remove_rows (*rows*)
remove one or more rows of molecules

Parameters **rows** (*int or list of ints:*) – the rows to remove

set_folder (*folderpath=''*, *server=None*, *username=None*, *passwrd=None*)

yield_mol_images (*rows=[], filters={}, mtype='optimised', sort_columns=[], align_to=[], rotations=[[0.0, 0.0, 0.0]], gbonds=True, represent='ball_stick', zoom=1.0, width=300, height=300, axis_length=0, relative=False, minval=-1, maxval=1, highlight=[], active=False, sopt_min_energy=20.0, sopt_cutoff_energy=0.0, atom_groups=[], alpha=0.5, transparent=False, hbondwidth=5, eunits='kJmol-1', no_hbonds=False, ipyimg=True*)
yields molecules

Parameters

- **mtype** – 'initial', 'optimised', 'nbo', 'highlight', 'highlight-initial', 'sopt' or 'hbond'
- **info_columns** (*list of str*) – columns to use as info in caption
- **max_cols** (*int*) – maximum columns in plot
- **label_size** (*int*) – subplot label size (pts)
- **start_letter** (*str*) – starting (capital) letter for labelling subplots
- **save_fname** (*str*) – name of file, if you wish to save the plot to file
- **rows** (*int or list*) – index for the row of each molecule to plot (all plotted if empty)
- **filters** (*dict*) – {columns:values} to filter by
- **sort_columns** (*list of str*) – columns to sort by
- **align_to** (*[int, int, int]*) – align geometries to the plane containing these atoms
- **rotations** (*list of [float, float, float]*) – for each rotation set [x,y,z] an image will be produced
- **gbonds** (*bool*) – guess bonds between atoms (via distance)
- **represent** (*str*) – representation of molecule ('none', 'wire', 'vdw' or 'ball_stick')
- **zoom** (*float*) – zoom level of images
- **width** (*int*) – width of original images
- **height** (*int*) – height of original images (although width takes precedent)
- **axis_length** (*float*) – length of x,y,z axes in negative and positive directions
- **relative** (*bool*) – coloring of nbo atoms scaled to min/max values in atom set (for nbo mtype)
- **minval** (*float*) – coloring of nbo atoms scaled to absolute min (for nbo mtype)
- **maxval** (*float*) – coloring of nbo atoms scaled to absolute max (for nbo mtype)
- **highlight** (*list of lists*) – atom indxes to highlight (for highlight mtype)
- **eunits** (*str*) – the units of energy to return (for sopt/hbond mtype)

- **sopt_min_energy** (*float*) – minimum energy to show (for sopt/hbond mtype)
- **sopt_cutoff_energy** (*float*) – energy below which bonds will be dashed (for sopt mtype)
- **alpha** (*float*) – alpha color value of geometry (for highlight/sopt/hbond mtypes)
- **transparent** (*bool*) – whether atoms should be transparent (for highlight/sopt/hbond mtypes)
- **hbondwidth** (*float*) – width of lines depicting interaction (for hbond mtypes)
- **atom_groups** (*[list or str, list or str]*) – restrict interactions to between two lists (or identifiers) of atom indexes (for sopt/hbond mtypes)
- **no_hbonds** (*bool*) – whether to ignore H-Bonds in the calculation
- **ipyimg** (*bool*) – whether to return an IPython image, PIL image otherwise
- **Yields** –
- **-------** –
- **indx** (*int*) – the row index of the molecule
- **mol** (*IPython.display.Image or PIL.Image*) – an image of the molecule in the format specified by ipyimg

pygauss.analysis.**unpack_and_make_molecule**(*val_dict*)

1.7.4 pygauss.docs module

Created on Tue Jun 16 15:52:53 2015

@author: chris sewell

class pygauss.docs.**MSDocument**(*docx=None*)

Bases: object

a class to output a Microsoft Word Document

inherited api details for docx.document.Document can be found at; https://python-docx.readthedocs.org/en/latest/api/document.html

the class has an internal state for the number of calls to add_picture and add_table for use in caption numbering

Parameters **docx** (*str or file-like object*) – can be either a path to a .docx file (a string) or a file-like object. If docx is missing or None, the built-in default document "template" is loaded.

__dir__()

required to have docx.document.Document methods in IPython tab completion

__getattr__(*name*)

required to get docx.document.Document methods

add_dataframe(*df*, *incl_indx=True*, *autofit=True*, *sig_figures=5*, *style='Medium List 1 Accent 1'*, *caption=None*)

add dataframe as a table to the document

Parameters

- **df** (*pandas.DataFrame*) – a pandas dataframe
- **incl_indx** (*bool*) – include dataframes index in table
- **autofit** (*bool*) – allow table to autofit content

- **sig_figures** (*int*) – number of significant figures for numbers in table
- **style** (*str*) – MS Word table style
- **caption** (*str*) – add a caption below the table

Returns **pic** – a table added to the document

Return type docx.table.Table

add_docstring(*docstring*, *style='Body Text'*, *markdown=True*)
adds a doctring to the document

this function will split text into paragraphs (denominated by a separating blank line) remove new-line characters and add to document, allowing for markdown style text designated in *pygauss.docs.MSDocument.add_markdown()*

Parameters

- **text** (*str*) – the text to add
- **style** (*str*) – the style to apply for each paragraph
- **markdown** (*bool*) – whether to apply markdown to the text

Returns **paras** – a list of paragraphs added to the document

Return type docx.text.paragraph.Paragraph

add_list(*text_list=[]*, *numbered=False*)
adds a list

add_markdown(*text=''*, *style='Body Text'*, *markup_dict=None*, *para=None*)
adds a paragraph to the document, allowing for paragraph/font styling akin to a stripped down version of markdown text:

paragraph level:

```
# Header (level denoted by number of #'s)
- bullet list
1. numbered list
```

font level:

```
**bold**
*italic*
_{subscript}
^{superscript}
~~strikethrough~~
$mathML$
```

Parameters

- **text** (*str*) – the text to add
- **style** (*str*) – the style to apply (overriden if paragraph level markdown)
- **markup_dict** (*dict*) – if set will override built in font level markup {font_attribute:(start_chars, end_chars)}
- **para** (*docx.text.paragraph.Paragraph*) – a pre-existing paragraph to add the text to if set, will ignore paragraph level markdown

Returns **para** – a paragraph added to the document

Return type docx.text.paragraph.Paragraph

add_mpl(*fig*, *dpi=None*, *width=None*, *height=None*, *pad_inches=0.2*, *caption=None*)
add matplotlib figure to the document

Parameters

- **fig** (*matplotlib.figure.Figure*) – a matplotlib figure
- **dpi** (*int*) – Dots per inch
- **width** (*float*) – width of image in document
- **height** (*float*) – width of image in document
- **pad_inches** (*float*) – amount of padding around the figure
- **caption** (*str*) – a caption below the figure

Returns **pic** – an inline picture added to the document

Return type docx.shape.InlineShape

add_picture(*image_path_or_stream*, *width=None*, *height=None*)
Return a new picture shape added in its own paragraph at the end of the document. The picture contains the image at *image_path_or_stream*, scaled based on *width* and *height*. If neither width nor height is specified, the picture appears at its native size. If only one is specified, it is used to compute a scaling factor that is then applied to the unspecified dimension, preserving the aspect ratio of the image. The native size of the picture is calculated using the dots-per-inch (dpi) value specified in the image file, defaulting to 72 dpi if no value is specified, as is often the case.

add_table(*rows*, *cols*, *style=None*)
Add a table having row and column counts of *rows* and *cols* respectively and table style of *style*. *style* may be a paragraph style object or a paragraph style name. If *style* is |None|, the table inherits the default table style of the document.

1.7.5 pygauss.isosurface module

Created on Mon May 25 15:23:49 2015

@author: chris based on add_isosurface function from chemview

pygauss.isosurface.**calc_normals**(*verts*, *faces*)
from; https://sites.google.com/site/dlampetest/python/calculating-normals-of-a-triangle-mesh-using-numpy

pygauss.isosurface.**get_isosurface**(*coordinates*, *function*, *isolevel=0.03*, *color=(255, 0, 0, 255)*, *extents=(5, 5, 5)*, *resolution=100*)
Add an isosurface to the current scene.

Parameters

- **coordinates** (*numpy.array*) – the coordinates of the system
- **function** (*func*) – A function that takes x, y, z coordinates as input and is broadcastable using numpy. Typically simple functions that involve standard arithmetic operations and functions such as `x**2 + y**2 + z**2` or `np.exp(x**2 + y**2 + z**2)` will work. If not sure, you can first pass the function through `numpy.vectorize`. Example: `mv.add_isosurface(np.vectorize(f))`
- **isolevel** (*float*) – The value for which the function should be constant.
- **color** (*(int, int, int, int)*) – The color given in RGBA format
- **extents** (*(float, float, float)*) – +/- x,y,z to extend the molecule geometrty when constructing the surface

- **resolution** (*int*) – The number of grid point to use for the surface. An high value will give better quality but lower performance.

pygauss.isosurface.**normalize_v3**(*arr*)
Normalize a numpy array of 3 component vectors shape=(n,3)

1.7.6 pygauss.utils module

Created on Thu Apr 30 01:08:30 2015

@author: chris

pygauss.utils.**circumcenter**(*pts*)
Computes the circumcenter and circumradius of M, N-dimensional points (1 <= M <= N + 1 and N >= 1). The points are given by the rows of an (M)x(N) dimensional maatrix pts.

Returns a tuple (center, radius) where center is a column vector of length N and radius is a scalar.

In the case of four points in 3D, pts is a 4x3 matrix arranged as:

pts = [x0 y0 z0] [x1 y1 z1] [x2 y2 z2] [x3 y3 z3]

with return value ([cx cy cz], R)

Uses an extension of the method described here: http://www.ics.uci.edu/~eppstein/junkyard/circumcenter.html

pygauss.utils.**circumcenter_barycoords**(*pts*)
Computes the barycentric coordinates of the circumcenter M, N-dimensional points (1 <= M <= N + 1 and N >= 1). The points are given by the rows of an (M)x(N) dimensional matrix pts.

Uses an extension of the method described here: http://www.ics.uci.edu/~eppstein/junkyard/circumcenter.html

pygauss.utils.**df_to_img**(*df*, *na_rep='-'*, *other_temp=None*, *font_size=None*, *width=None*, *height=None*, *unconfined=False*)
converts a pandas Dataframe to an IPython image

na_rep [str] how to represent empty (nan) cells

other_temp [str] a latex template to use for the table other than the default

The function uses pandas to convert the dataframe to a latex table, applies a template, converts to a PDF, converts to an image, and finally return the image

to use this function you will need the pdflatex executable from tex distribution, the convert executable from imagemagick, which also requires ghostscript; http://www.ghostscript.com/download/gsdnld.html http://www.imagemagick.org/script/binary-releases.php

NB: on Windows some issues were found with convert being an already existing application. To overcome this change its filename and use the im_name variable.

pygauss.utils.**imgplot_kmean_groups**(*analysis*, *category*, *cat_name*, *groups*, *columns*, *filters={}*, *output=[]*, *max_cols=2*, ***kwargs*)

pygauss.utils.**is_wellcentered**(*pts*, *tol=1e-08*)
Determines whether the M points in N dimensions define a well-centered simplex.

pygauss.utils.**set_imagik_exe**(*name*)

CHAPTER 2

License

Pygauss is released under the GNU GPLv3 and its main developer is Chris Sewell.

p

pygauss.analysis, 27
pygauss.docs, 33
pygauss.file_io, 20
pygauss.isosurface, 35
pygauss.molecule, 21
pygauss.utils, 36

Symbols

__dir__() (pygauss.docs.MSDocument method), 33
__enter__() (pygauss.file_io.Folder method), 20
__exit__() (pygauss.file_io.Folder method), 20
__getattr__() (pygauss.docs.MSDocument method), 33

A

active() (pygauss.file_io.Folder method), 20
add_basic_properties() (pygauss.analysis.Analysis method), 27
add_dataframe() (pygauss.docs.MSDocument method), 33
add_docstring() (pygauss.docs.MSDocument method), 34
add_frequency() (pygauss.molecule.Molecule method), 21
add_initialgeom() (pygauss.molecule.Molecule method), 22
add_list() (pygauss.docs.MSDocument method), 34
add_markdown() (pygauss.docs.MSDocument method), 34
add_mol_property() (pygauss.analysis.Analysis method), 27
add_mol_property_subset() (pygauss.analysis.Analysis method), 28
add_mpl() (pygauss.docs.MSDocument method), 35
add_nbo_analysis() (pygauss.molecule.Molecule method), 22
add_optimisation() (pygauss.molecule.Molecule method), 22
add_pes_analysis() (pygauss.molecule.Molecule method), 22
add_picture() (pygauss.docs.MSDocument method), 35
add_run() (pygauss.analysis.Analysis method), 28
add_runs() (pygauss.analysis.Analysis method), 28
add_table() (pygauss.docs.MSDocument method), 35
Analysis (class in pygauss.analysis), 27

C

calc_2plane_angle() (pygauss.molecule.Molecule method), 22
calc_bond_angle() (pygauss.molecule.Molecule method), 22
calc_dihedral_angle() (pygauss.molecule.Molecule method), 22
calc_hbond_energy() (pygauss.molecule.Molecule method), 22
calc_kmean_groups() (pygauss.analysis.Analysis method), 28
calc_min_dist() (pygauss.molecule.Molecule method), 22
calc_nbo_charge() (pygauss.molecule.Molecule method), 22
calc_nbo_charge_center() (pygauss.molecule.Molecule method), 22
calc_normals() (in module pygauss.isosurface), 35
calc_opt_trajectory() (pygauss.molecule.Molecule method), 22
calc_polar_coords_from_plane() (pygauss.molecule.Molecule method), 22
calc_sopt_energy() (pygauss.molecule.Molecule method), 22
circumcenter() (in module pygauss.utils), 36
circumcenter_barycoords() (in module pygauss.utils), 36
combine_molecules() (pygauss.molecule.Molecule method), 22
copy() (pygauss.analysis.Analysis method), 28
count_runs() (pygauss.analysis.Analysis method), 28

D

df_to_img() (in module pygauss.utils), 36

F

Folder (class in pygauss.file_io), 20
folder (pygauss.analysis.Analysis attribute), 28

G

get_atom_group() (pygauss.molecule.Molecule method), 23
get_basic_property() (pygauss.analysis.Analysis method), 28
get_basis_descript() (pygauss.molecule.Molecule method), 23

get_basis_funcs() (pygauss.molecule.Molecule method), 23
get_folder() (pygauss.analysis.Analysis method), 28
get_folder() (pygauss.molecule.Molecule method), 23
get_freq_analysis() (pygauss.molecule.Molecule method), 23
get_hbond_analysis() (pygauss.molecule.Molecule method), 23
get_ids() (pygauss.analysis.Analysis method), 28
get_init_read_errors() (pygauss.molecule.Molecule method), 23
get_isosurface() (in module pygauss.isosurface), 35
get_molecule() (pygauss.analysis.Analysis method), 28
get_opt_energy() (pygauss.molecule.Molecule method), 23
get_orbital_count() (pygauss.molecule.Molecule method), 23
get_orbital_energies() (pygauss.molecule.Molecule method), 24
get_orbital_homo_lumo() (pygauss.molecule.Molecule method), 24
get_path() (pygauss.file_io.Folder method), 20
get_run_error() (pygauss.molecule.Molecule method), 24
get_sopt_analysis() (pygauss.molecule.Molecule method), 24
get_table() (pygauss.analysis.Analysis method), 28
get_zeropt_energy() (pygauss.molecule.Molecule method), 24

I

imgplot_kmean_groups() (in module pygauss.utils), 36
is_conformer() (pygauss.molecule.Molecule method), 24
is_optimised() (pygauss.molecule.Molecule method), 24
is_wellcentered() (in module pygauss.utils), 36
islocal() (pygauss.file_io.Folder method), 20

L

list_files() (pygauss.file_io.Folder method), 20

M

Molecule (class in pygauss.molecule), 21
MSDocument (class in pygauss.docs), 33

N

NoOutputFolder (class in pygauss.file_io), 21
normalize_v3() (in module pygauss.isosurface), 36

O

orbit_z() (in module pygauss.molecule), 27

P

plot_dos() (pygauss.molecule.Molecule method), 24
plot_freq_analysis() (pygauss.molecule.Molecule method), 25
plot_mol_graphs() (pygauss.analysis.Analysis method), 29
plot_mol_images() (pygauss.analysis.Analysis method), 30
plot_opt_energy() (pygauss.molecule.Molecule method), 25
plot_opt_trajectory() (pygauss.molecule.Molecule method), 25
plot_pes_scans() (pygauss.molecule.Molecule method), 25
plot_radviz_comparison() (pygauss.analysis.Analysis method), 31
pygauss.analysis (module), 27
pygauss.docs (module), 33
pygauss.file_io (module), 20
pygauss.isosurface (module), 35
pygauss.molecule (module), 21
pygauss.utils (module), 36

R

read_file() (pygauss.file_io.Folder method), 20
remove_alignment_atoms() (pygauss.molecule.Molecule method), 25
remove_columns() (pygauss.analysis.Analysis method), 31
remove_non_conformers() (pygauss.analysis.Analysis method), 31
remove_non_optimised() (pygauss.analysis.Analysis method), 32
remove_rows() (pygauss.analysis.Analysis method), 32

S

save_ipyimg() (pygauss.file_io.Folder method), 20
save_ipyimg() (pygauss.file_io.NoOutputFolder method), 21
save_mplfig() (pygauss.file_io.Folder method), 20
save_mplfig() (pygauss.file_io.NoOutputFolder method), 21
save_pilimg() (pygauss.file_io.Folder method), 21
save_pilimg() (pygauss.file_io.NoOutputFolder method), 21
set_alignment_atoms() (pygauss.molecule.Molecule method), 25
set_folder() (pygauss.analysis.Analysis method), 32
set_imagik_exe() (in module pygauss.utils), 36
show_active_orbital() (pygauss.molecule.Molecule method), 25
show_hbond_analysis() (pygauss.molecule.Molecule method), 26
show_highlight_atoms() (pygauss.molecule.Molecule method), 26
show_initial() (pygauss.molecule.Molecule method), 26

show_nbo_charges() (pygauss.molecule.Molecule method), 26
show_optimisation() (pygauss.molecule.Molecule method), 26
show_sopt_bonds() (pygauss.molecule.Molecule method), 26

U

unpack_and_make_molecule() (in module pygauss.analysis), 33

W

write_file() (pygauss.file_io.Folder method), 21
write_file() (pygauss.file_io.NoOutputFolder method), 21

Y

yield_mol_images() (pygauss.analysis.Analysis method), 32
yield_orbital_images() (pygauss.molecule.Molecule method), 26