PyGauss Documentation

Release 0.4.4

Chris Sewell

June 17, 2015

Contents

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 Quick Start

1.1.1 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
```

1.1.2 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
```

1.1.3 Troubleshooting

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

1.1.4 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)
```

1.2 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: 0.4.3
```

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

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

```
37
```

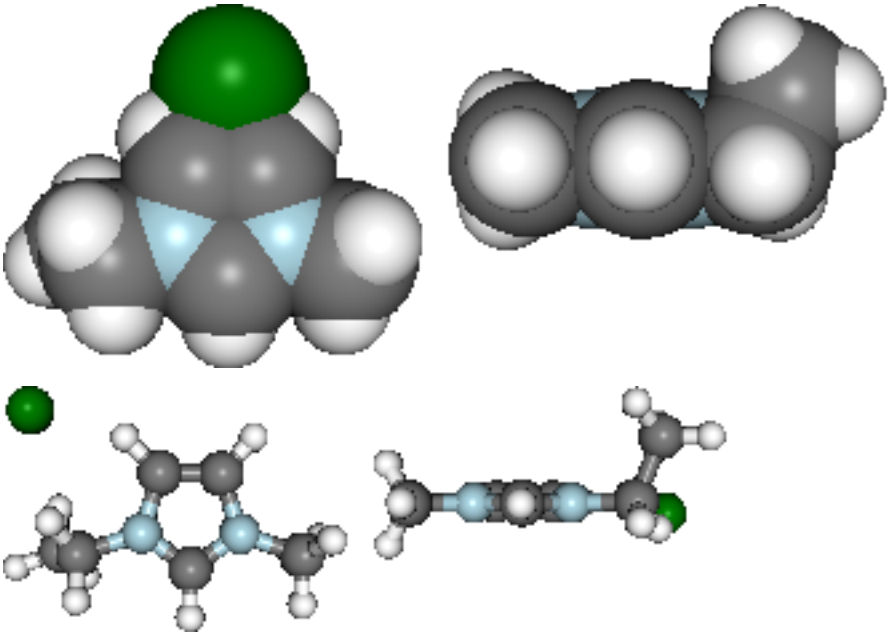
Note: the *folder* object will act identical whether using a local path or one on a server over ssh (using paramiko):

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

1.2.1 Single Molecule Analysis

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

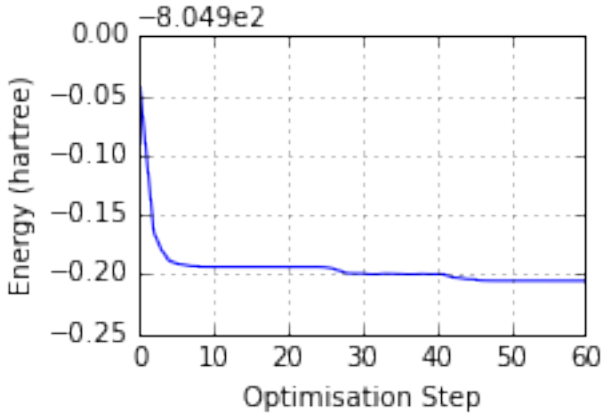
```
mol = pg.molecule.Molecule(folder_obj=folder,
                   init_fname='CJS1_emim-cl_B_init.com',
                   opt_fname=['CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_opt-modredundant_difrz.log',
                              'CJS1_emim-cl_B_6-311+g-d-p-_gd3bj_opt-modredundant_difrz_err.log',
                              '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])

#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)
```

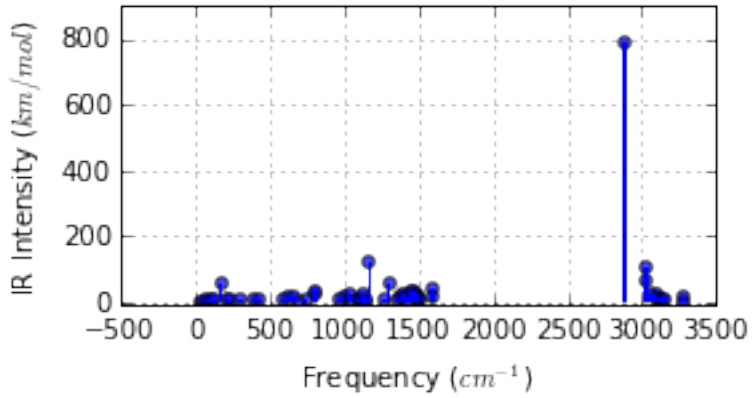


Basic analysis of optimisation...

```
print('Optimised? {0}, Conformer? {1}, Energy = {2} a.u.'.format(
    mol.is_optimised(), mol.is_conformer(),
    round(mol.get_optimisation_E(units='hartree'),3)))
ax = mol.plot_optimisation_E(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.
```

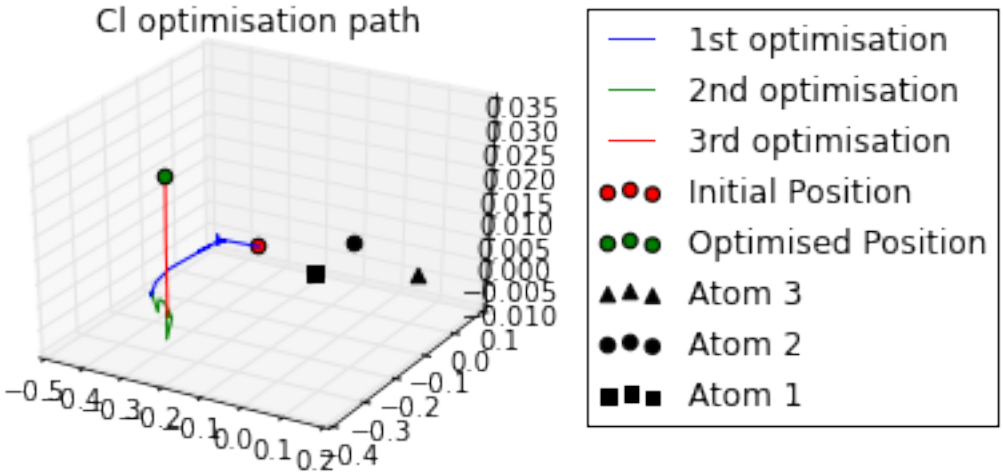




Geometric analysis...

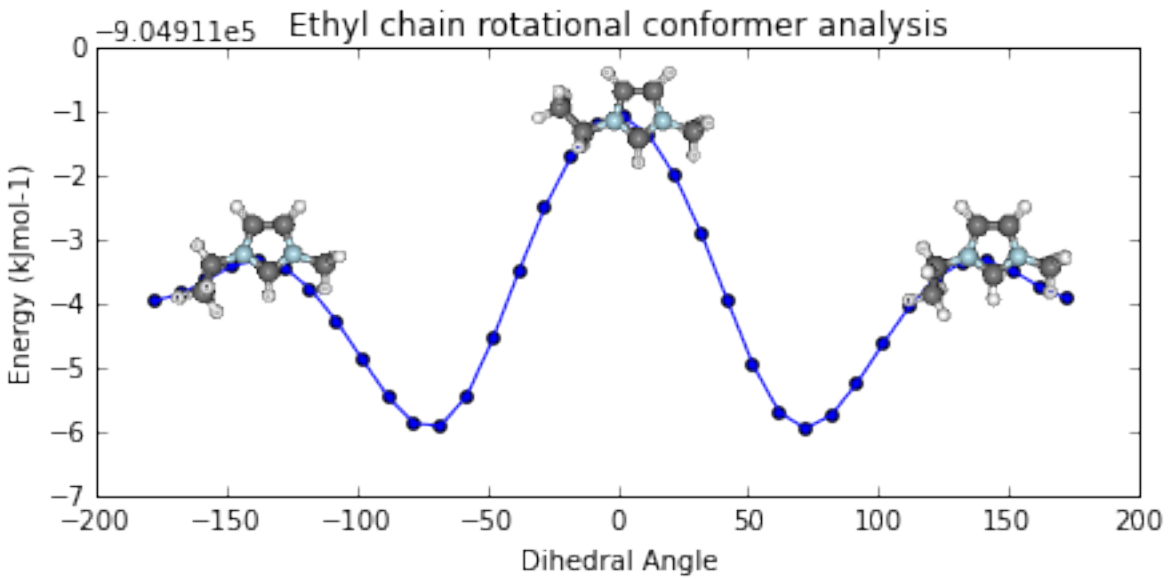
```
print 'Cl optimised polar coords from aromatic ring : ({0}, {1},{2})'.format(
    *[round(i, 2) for i in mol.calc_polar_coords_from_plane(20,3,2,1)])
ax = mol.plot_opt_trajectory(20, [3,2,1])
ax.set_title('Cl optimisation path')
ax.get_figure().set_size_inches(4, 3)
```

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



Potential Energy Scan analysis of geometric conformers...

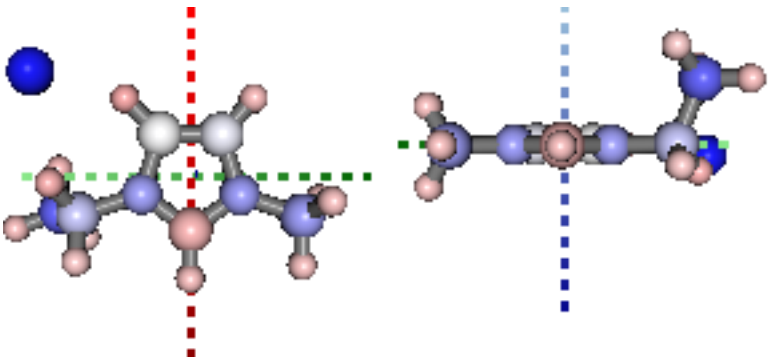
```
mol2 = pg.molecule.Molecule(folder_obj=folder, alignto=[3,2,1],
            pes_fname=['CJS_emim_6311_plus_d3_scan.log',
                       'CJS_emim_6311_plus_d3_scan_bck.log'])
ax = mol2.plot_pes_scans([1,4,9,10], rotation=[0,0,90], img_pos='local_maxs', zoom=0.5)
ax.set_title('Ethyl chain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



Natural Bond Orbital and Second Order Perturbation Theory analysis...

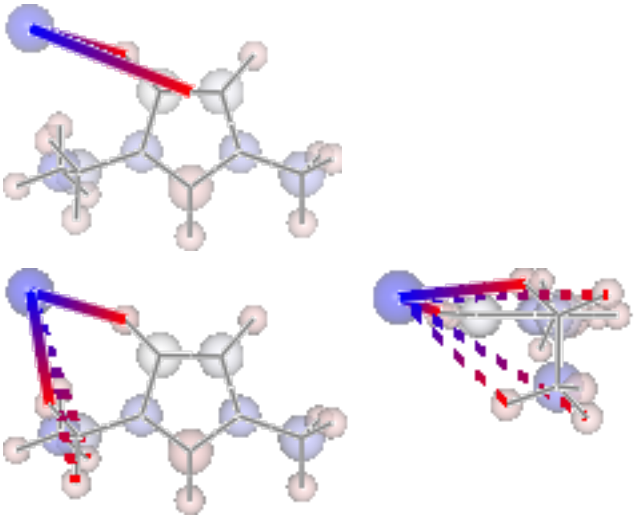
```
print '+ve charge centre polar coords from aromatic ring: ({0} {1},{2})'.format(
    *[round(i, 2) for i in mol.calc_nbo_charge_center(3, 2, 1)])
display(mol.show_nbo_charges(represent='ball_stick', axis_length=0.4,
                             rotations=[[0,0,90], [-90, 90, 0]]))
```

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



```
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
```

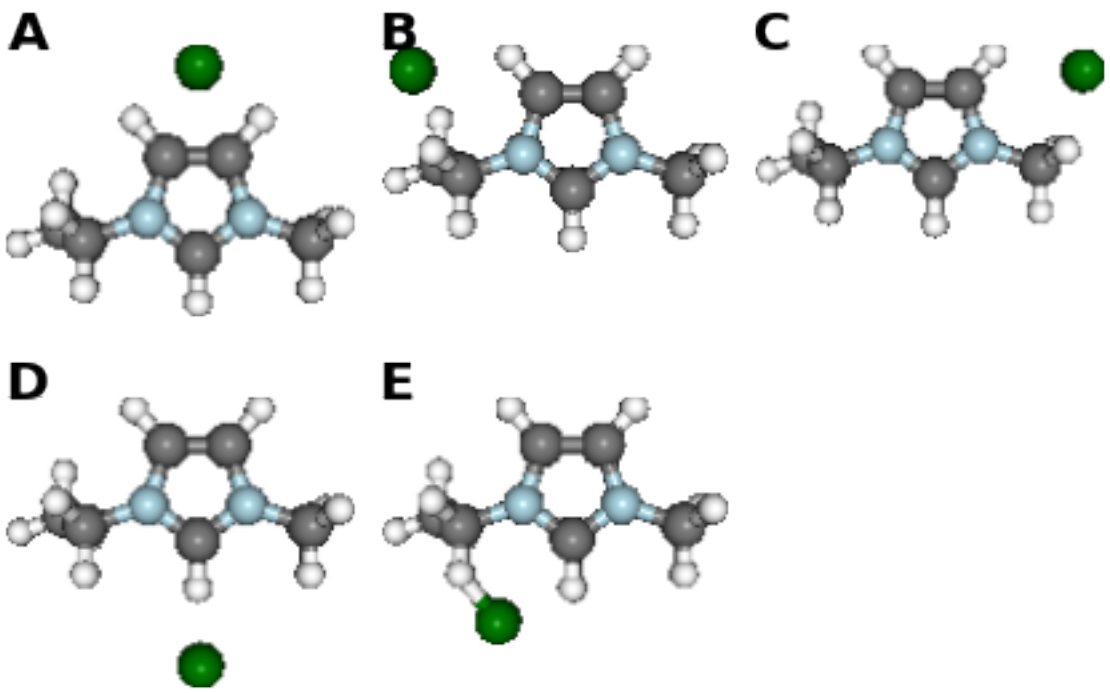


1.2.2 Multiple Computations Analysis

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

```
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]})

fig, caption = analysis.plot_mol_images(mtype='initial', max_cols=3,
                        info_columns=['Cation', 'Anion', 'Initial'],
                        rotations=[[0,0,90]])
print caption
```

```
Figure: (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 then 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_optimisation_E', 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'])
```

 <table border="1" class="dataframe"> <thead> <tr> <th></th> <th></th> <th></th> <th colspan="2" halign="left"></th> <th colspan="2" halign="left">Cation</th> <th>Anion</th> <th colspan="4" halign="left">Anion-Cation</th> </tr> <tr> <th></th> <th></th> <th></th> <th>Opt</th> <th>Energy (au)</th> <th>chain, ψ</th> <th>Charge</th> <th>Charge</th> <th>r</th> <th>θ</th> <th>ϕ</th> <th>h-bond</th> </tr> <tr> <th>Anion</th> <th>Cation</th> <th>Initial</th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> <th></th> </tr> </thead> <tbody> <tr> <th rowspan="5" valign="top">cl</th> <th rowspan="5" valign="top">emim</th> <th>B</th> <td>NaN</td> <td>-805.105</td> <td>80.794</td> <td>0.888</td> <td>-0.888</td> <td>0.420</td> <td>-123.392</td> <td>172.515</td> <td>111.713</td> </tr> <tr> <th>BE</th> <td>NaN</td> <td>-805.105</td> <td>80.622</td> <td>0.887</td> <td>-0.887</td> <td>0.420</td> <td>-123.449</td> <td>172.806</td> <td>112.382</td> </tr> <tr> <th>BM</th> <td>True</td> <td>-805.104</td> <td>73.103</td> <td>0.874</td> <td>-0.874</td> <td>0.420</td> <td>124.121</td> <td>-166.774</td> <td>130.624</td> </tr> <tr> <th>F</th> <td>True</td> <td>-805.118</td> <td>147.026</td> <td>0.840</td> <td>-0.840</td> <td>0.420</td> <td>10.393</td> <td>0.728</td> <td>202.004</td> </tr> <tr> <th>FE</th> <td>NaN</td> <td>-805.117</td> <td>85.310</td> <td>0.851</td> <td>-0.851</td> <td>0.417</td> <td>-13.254</td> <td>-4.873</td> <td>177.360</td> </tr> </tbody> </table> **NEW FEATURE:** there is now an option (requiring pdflatex and ghostscript+imagemagik) to output the

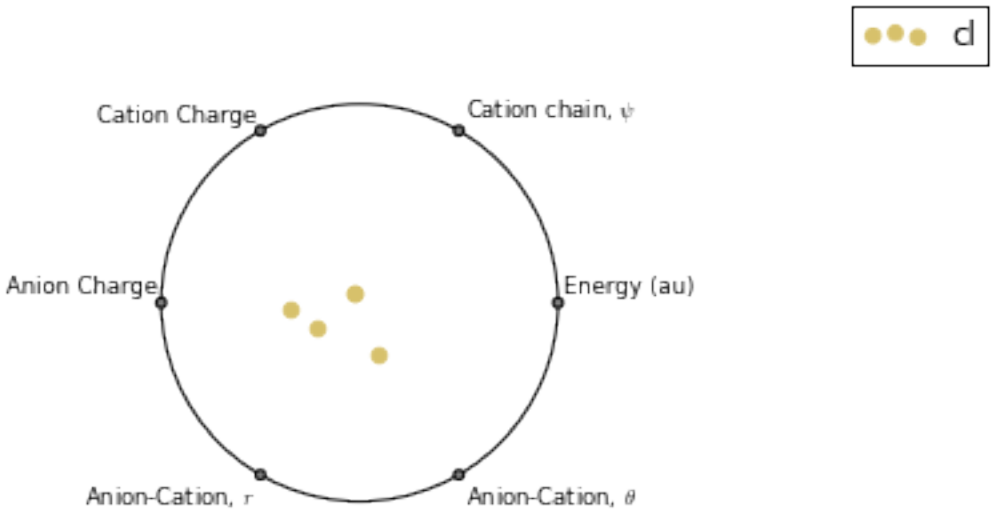
tables as a latex formatted image.

```
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

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)
```

Category 1:

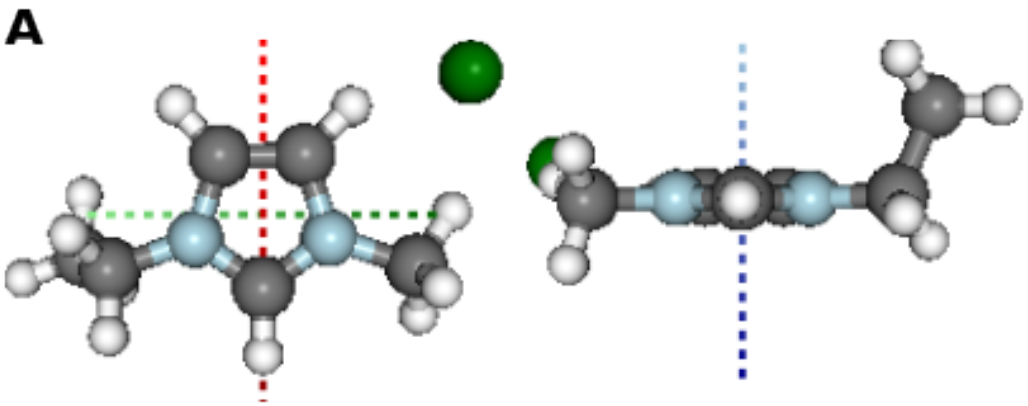


Figure: (A) BM

Category 2:

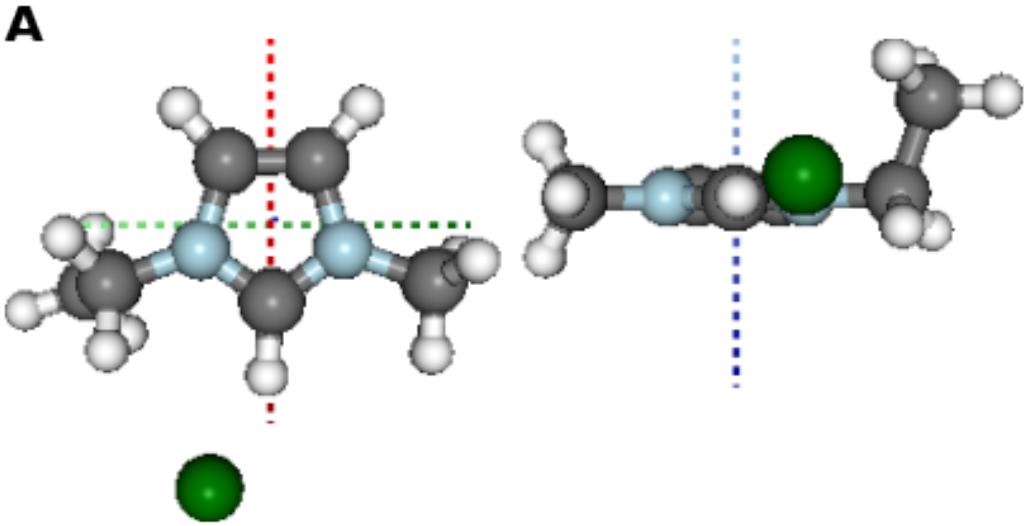
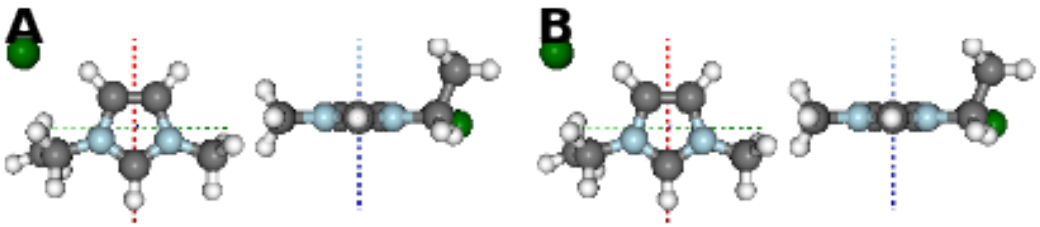


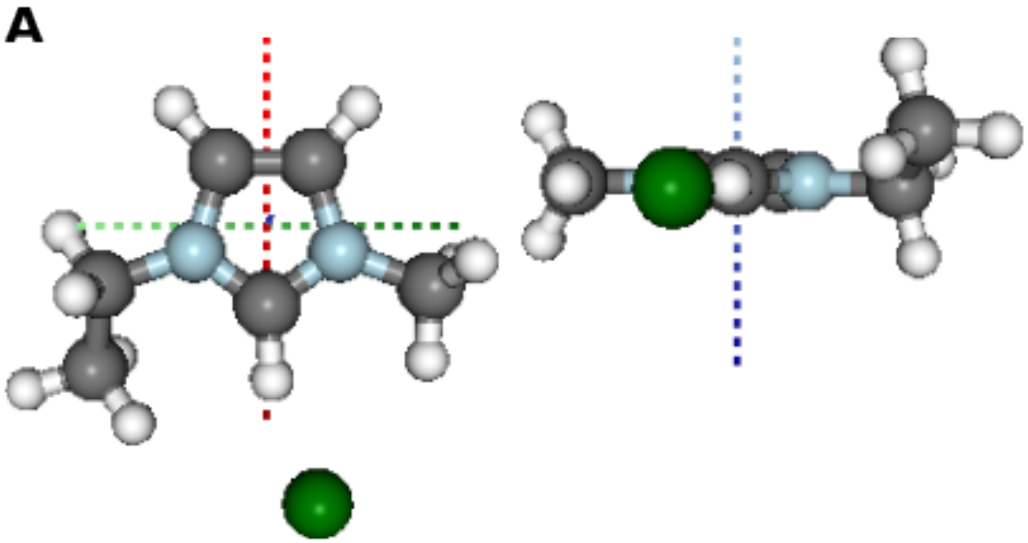
Figure: (A) FE

Category 3:



```
Figure: (A) B, (B) BE
```

Category 4:

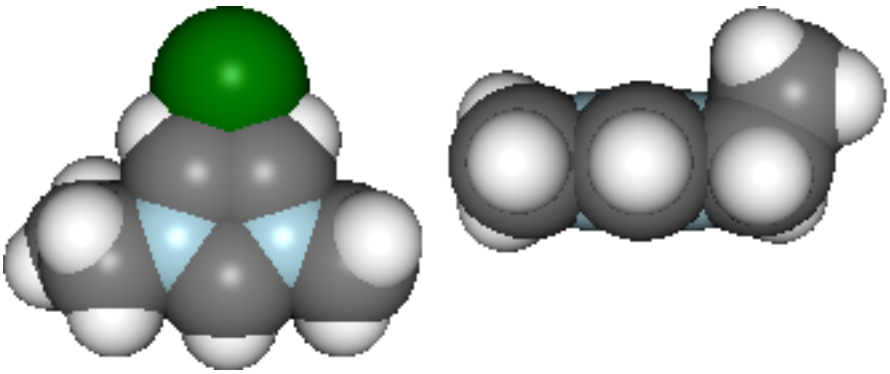



```
Figure: (A) F
```

1.2.3 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.

```
d = pg.MSDocument()
d.add_heading('A Pygauss Example Assessment', level=1)

d.add_paragraph('We have looked at the following aspects;')
d.add_list(['geometric conformers', 'electronic structure'])
```

```
d.add_heading('Geometric Conformers', level=2)
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)
fig.clear()
d.add_markdown(caption.replace('Figure:', '**Figure:**'))
d.add_paragraph()
df = analysis.get_table(columns=['Anion Charge', 'Cation Charge',
                                 'Energy (au)'],
                    row_index=['Anion', 'Cation', 'Initial'])
d.add_dataframe(df, incl_indx=True, style='Medium Shading 1 Accent 1')
d.add_markdown('**Table:** Analysis of Conformer Charge')

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

Which gives us the following:

A Pygauss Example Assessment

We have looked at the following aspects:

- geometric conformers
- electronic structure

Geometric Conformers

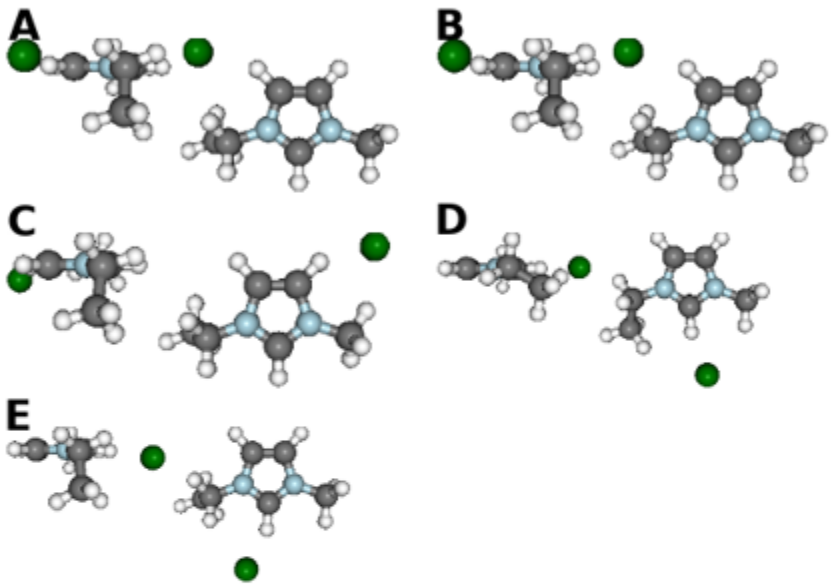


Figure: (A) cl, emim, B, (B) cl, emim, BE, (C) cl, emim, BM, (D) cl, emim, F, (E) cl, emim, FE

Anion	Cation	Initial	Anion Charge	Cation Charge	Energy (au)
cl	emim	B	-0.88755	0.88756	-805.11
cl	emim	BE	-0.88724	0.88723	-805.11
cl	emim	BM	-0.87435	0.87436	-805.1
cl	emim	F	-0.84037	0.84037	-805.12
cl	emim	FE	-0.85079	0.8508	-805.12

Table: Analysis of Conformer Charge

MORE TO COME!!

1.3 Project Status

1.3.1 Distribution

Conda	
PyPi	
Documents	

1.3.2 Development

GitHub	
Unit Testing	
Testing Coverage	
Documents	

1.4 Whats New

1.4.1 v0.4.4 - Output to MS Word Documents

added the pygauss.docs module for outputting analysis to documents, initially utilising *python-docx* with the *MSDocument* class

1.4.2 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/).

1.4.3 v0.4.2 - Addition of Documentation

addition of Sphinx documentation

1.4.4 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

1.4.5 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

1.4.6 v0.2.2 - Table Image Improvements

Improvements to Table to Image functionality on OSx

- added some fixes
- re-organised test modules

1.4.7 v0.2.1 - Latex Table Images

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

1.4.8 v0.2 - Initial working distribution

Working distribution of pygauss to be converted to first conda package

1.4.9 v0.1 - First Version

the first version

1.5 Whats To Come

1.5.1 Natural Bonding Orbital Visualisation

Visualise natural binding orbitals, in particular HOMO and LUMO

1.5.2 Densities of State analysis

1.5.3 Molecular Image Labelling

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

1.6 User API

1.6.1 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()

get_path()

islocal()

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

pattern [str] a pattern the file must match that can include * wildcards

read_file(*file_name*)

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

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

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*)

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*)

1.6.2 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
- **of the file names can have wildcards (e.g. 'filename*.log) in them,** (*any*) –
- **long as this resolves to a single path in the directory** (*as*) –
- **NB** (*nbo population analysis must be run with the GFInput flag to ensure*) –
- **is output to the log file** (*data*) –

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_basis_descript()

get_basis_funcs()

get_folder()
return the Folder instance

get_freq_analysis()
return frequency analysis

Returns **data** – frequency data

Return type pd.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** (*pandas.DataFrame*) – a table of interactions
- *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_optimisation_E(*units='eV'*, *final=True*)
return the SCF optimisation energy

Parameters

- **units** (*str*) – the unit type of the energy
- **final** (*bool*) – return only the final optimised energy if True, else for all steps

Returns **out** – 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 np.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

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

is_optimised()
was the geometry optimised

plot_freq_analysis()
plot frequency analysis

Returns **data** – plotted frequency data

Return type matplotlib.axes._subplots.AxesSubplot

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_optimisation_E(*units='eV'*)
plot SCF optimisation energy

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

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_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

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_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

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=True** – 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*)

1.6.3 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*)
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()

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_freq_analysis(*info_columns=[]*, *rows=[]*, *filters={}*)
return frequency analysis

Parameters

- **info_columns** (*list of str*) – columns to use as info in caption
- **rows** (*int or list*) – index for the row of each molecule to plot (all plotted if empty)
- **filters** (*dict*) – {columns:values} to filter by

Returns **data** – frequency data

Return type pd.DataFrame

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

plot_freq_analysis(*info_columns=[]*, *rows=[]*, *filters={}*, *share_plot=True*, *include_row=False*)
plot frequency analysis

Parameters

- **info_columns** (*list of str*) – columns to use as info in caption
- **rows** (*int or list*) – index for the row of each molecule to plot (all plotted if empty)
- **filters** (*dict*) – {columns:values} to filter by
- **share_plot** (*bool*) – whether to share a single plot or have multiple ones
- **include_row** (*bool*) – include row number in legend labels

Returns **data** – plotted frequency data

Return type matplotlib.figure.Figure

plot_mol_images(*mtype='optimised', info_columns=[], info_incl_id=False, max_cols=1, label_size=20, start_letter='A', save_fname=None, 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', 'sopt' or 'hbond'
- **info_columns** (*list of str*) – columns to use as info in caption
- **info_incl_id** (*bool*) – include molecule id number 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
- **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

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', 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', '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
- **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

1.6.4 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

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 methods in ipython tab completion

__getattr__(*name*)

required to get docx.Document methods

add_dataframe(*df*, *incl_indx=True*, *autofit=True*, *sig_figures=5*, *style='Medium List 1 Accent 1'*)

add dataframe as a table

add_list(*text_list=[]*, *numbered=False*)

adds a list

add_markdown(*text=''*, *style='Body Text'*)

adds a paragraph to the document, allowing for markdown style **bold** and *italic* text

add_mpl(*fig*, *dpi=None*, *width=None*, *height=None*, *pad_inches=0.2*)

add matplotlib figure to the document, width/height in cm Amount of padding around the figure

1.6.5 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.**my_calc_normals**(*verts*, *faces*)
doesn't work

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

1.6.6 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

Symbols

A

C

D

F

G

I

L

M

N

O

P

R

S

W

Y