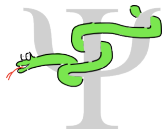
An introduction to cclib

Eric Berquist, August 24th, 2018



Documentation: https://cclib.github.io
Source code: https://github.com/cclib/cclib

1 Installation

```
# All should automatically install NumPy, the only hard dependency.
sudo apt-get install cclib # Ubuntu (out of date)
pip install --user cclib # Python Package Index (preferred)
yay -S python-cclib{,-git} # Arch Linux AUR, release and development versions
```

2 First steps

2.1 Command line using `ccget`

Running

```
molecule water {
0 1
O
H 1 0.99
H 1 0.99 2 106.0
}

set {
    basis sto-3g
}

energy('ccsd(t)')
```

through Psi4 gives

```
$ ccget --list --verbose water_ccsd\(t\).out
Attempting to read water_ccsd(t).out
Identified logfile to be in Psi4 format
cclib can parse the following attributes from water_ccsd(t).out:
  atomcoords
  atommasses
  atomnos
  ccenergies
  charge
  coreelectrons
  homos
  metadata
  moenergies
  mosyms
  mult
  natom
  nbasis
  nmo
  scfenergies
  scftargets
  scfvalues
```

```
$ ccget natom atomnos atommasses atomcoords charge mult nbasis nmo homos scftargets scfvalues scfenergies ccenergies water_ccsd\(t\).out
Attempting to read water_ccsd(t).out
natom
3
atomnos
[8 1 1]
atommasses
[15.99491462  1.00782503  1.00782503]
atomcoords
[[[ 0.          0.         -0.06667853]
  [ 0.         -0.79064915  0.52911834]
  [ 0.          0.79064915  0.52911834]]]
charge
0
mult
1
nbasis
7
nmo
7
homos
[4]
scftargets
[[1.e-08 1.e-08]]
scfvalues
[array([[-7.47145e+01,  3.03943e-01],
       [-2.01407e-01,  5.09857e-02],
       [-4.62920e-02,  9.16936e-03],
       [-1.69766e-03,  3.56660e-03],
       [-4.26333e-04,  3.22652e-04],
       [-2.49391e-06,  3.57847e-05],
       [-2.42460e-08,  1.11355e-06],
       [-2.03499e-11,  5.83644e-10]])]
scfenergies
[-2039.88321508]
ccenergies
[-2041.33710337]
```

2.2 In a Python script (preferred)

```
from cclib.io import ccread

data = ccread("water_ccsd(t).out")
attrs = ["natom", "atomnos", "atommasses", "atomcoords", "charge", "mult", "nbasis", "nmo", "homos", "scftargets", "scfvalues", "scfenergies", "ccenergies"]
for attr in attrs:
    print(attr, type(getattr(data, attr)))
```

```
natom <class 'int'>
atomnos <class 'numpy.ndarray'>
atommasses <class 'numpy.ndarray'>
atomcoords <class 'numpy.ndarray'>
charge <class 'int'>
mult <class 'int'>
nbasis <class 'int'>
nmo <class 'int'>
homos <class 'numpy.ndarray'>
scftargets <class 'numpy.ndarray'>
scfvalues <class 'list'>
scfenergies <class 'numpy.ndarray'>
ccenergies <class 'numpy.ndarray'>
```

A full list of all available attributes is present in `help(cclib.parser.data.ccData)`.

3 Modules

Under `cclib`, the core modules are

- `bridge`: Facilities for moving parsed data to other cheminformatic libraries. Currently available are Open Babel, PyQuante 1, and BioPython.
- `method`: Example analyses and calculations based on data parsed by cclib.
- `io`: Contains all writers (and some readers) for standard chemical representations.
- `parser`: Contains parsers for all available programs. Currently, this is ADF, DALTON, GAMESS-US and Firefly, GAMESS-UK, Gaussian, Jaguar, (Open)Molcas, Molpro, MOPAC, NWChem, ORCA, Psi3 and Psi4, Q-Chem, and Turbomole.
- `scripts`: Contains the command-line scripts `ccframe`, `ccget`, `ccwrite`, and `ccframe`.

3.1 `method`

Most methods are classes that take a `ccData` instance and require a call to `.calculate()`. The base class is `calculationmethod.Method`.

- `cda.CDA`: Charge decomposition analysis, also accessible via the command-line script `cda`.
- `cspa.CSPA`: C-squared population analysis, $\Phi_{\mu i} = \frac{C_{\mu i}^2}{\sum_\nu C_{\nu i}^2}$.
- `density.Density`: Density matrix calculation, $D_{\mu\nu}^{\text{RHF}} = 2\sum_i^{\text{occ MO}} C_{\mu i} C_{\nu i}, D_{\sigma\mu\nu}^{\text{UHF}} = \sum_i^{\text{occ MO}} C_{\mu i}^\sigma C_{\nu i}^\sigma$.
- `electrons.Electrons`: Methods pertaining to electrons (just `.count()` for now).

- `fragments.FragmentAnalysis`: Convert a molecule's basis functions from atomic-based to fragment MO-based.
- `lpa.LPA`: Löwdin population analysis
- `mbo.MBO`: Mayer's bond orders
- `mpa.MPA`: Mulliken population analysis $\Phi_{\mu i} = \sum_{\nu} C_{\mu i} C_{\nu i} S_{\mu\nu}$
- `nuclear.Nuclear`: Methods pertaining to atomic nuclei (stoichiometry strings, nuclear repulsion energies, rotational constants)
- `opa.OPA`: Overlap population analysis, $OP_{AB,i} = 2 \sum_{\mu \in A} \sum_{\nu \in B} C_{\mu i} C_{\nu i} S_{\mu\nu}$
- `orbitals.Orbitals`: Methods containing to orbitals (just `.closed_shell()` for now)
- `volume`: Methods and functions related to volumes (write cube files, form and integrate wavefunctions and electron densities); for more comprehensive functionality, see ORBKIT (https://github.com/orbkit/orbkit), which incorporates cclib.

3.2 io

Accessible via both the command-line scripts (`ccget`, `ccwrite`, `ccframe`)

- Format-specific classes are `CJSONReader`, `CJSONWriter`, `CML` (writer), `MOLDEN` (writer), `WFXWriter`, `XYZReader`, and `XYZWriter`.
- `ccio.ccopen`: Guess the identity of a particular log file and return an instance of it.
- `ccio.ccread`: Attempt to open and read computational chemistry data from a file. The most important function in cclib, it calls `ccopen` then `parse()`. If the file type can't be determined, fall back to another internal reader or Open Babel
- `ccio.ccwrite`: Write the parsed data from an outputfile to a standard chemical representation. The available formats are CJSON (https://github.com/OpenChemistry/chemicaljson), CML, MOLDEN, AIM Extended Wavefunction (*.wfx, http://aim.tkgristmill.com/wfxformat.html), and XYZ.
- `ccio.ccframe`: Returns a `pandas.DataFrame` of data attributes parsed by from one or more logfiles. The command-line version can write CSV, HDF5, JSON, `pickle`, and XLSX formats.

3.3 parser

In addition to all the available parser implementations (which should not be called directly),

- `data.ccData`: The core data structure returned by parsers that contains attributes corresponding to quantum chemical calculation attributes.
- `utils`: Common utilities, such as a simple unit `convertor` and `PeriodicTable`.

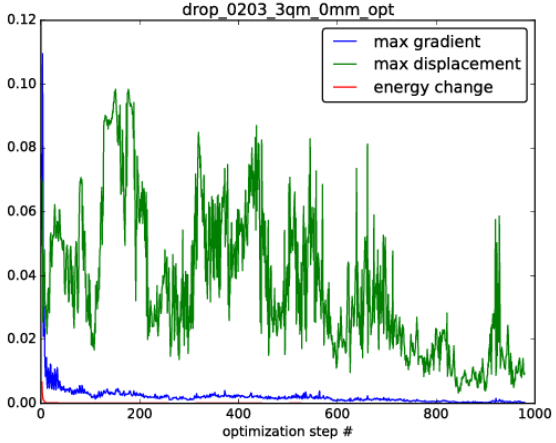
4 Examples

4.1 Calculate the AO-basis density matrix

```
# Need to do this...
C = data.mocoeffs[0].T
# ...so we can write the correct equation (RHF):
# D = 2 \sum_i^{occ MO} C_{\mu i} C_{\nu i}
D = 2 * np.dot(C[:, :nocc], C[:, :nocc].T)
# Compare to the Method:
from cclib.method import Density
m = Density(data)
m.calculate()
np.testing.assert_allclose(m.density[0], D, rtol=0,
↪  atol=1.0e-15)
```

4.2 Plot geometry optimization convergence

```
stub = os.path.splitext(compchemfilename)[0]

data = ccread(compchemfilename)

fig, ax = plt.subplots()

if type(job) == cclib.parser.qchemparser.QChem:

    scfenergies = [utils.convertor(scfenergy, 'eV',
    ↪  'hartree') for scfenergy in data.scfenergies]
    gradients = [geovalue[0] for geovalue in
    ↪  data.geovalues]
    displacements = [geovalue[1] for geovalue in
    ↪  data.geovalues]
    energy_changes = [(geovalue[2] *
    ↪  args.scaling_energy_change) for geovalue in
    ↪  data.geovalues]

    # If this isn't true, something funny happened during
    ↪  the parsing, so fail out.
    assert len(scfenergies) == len(gradients)

    steps = range(1, len(scfenergies) + 1)

    # ax.plot(steps, scfenergies, label='SCF energy')
    ax.plot(steps, gradients, label='max gradient')
    ax.plot(steps, displacements, label='max
    ↪  displacement')
    ax.plot(steps, energy_changes, label='energy change')

    ax.set_title(stub)
    ax.set_xlabel('optimization step #')

ax.legend(loc='best', fancybox=True)

fig.savefig(stub + '.pdf', bbox_inches='tight')
```



5 Development

Inside of a virtualenv,

```
git clone https://github.com/cclib/cclib.git; cd cclib;
↪  pip install -e .
```

which allows making changes without reinstallation. To run the test suite,

```
pip install -r requirements.txt
bash ./travis/run_travis_tests.sh
```

To build the documentation locally,

```
pip install sphinx sphinx_rtd_theme
cd doc; make; firefox sphinx/_build/html/index.html
```