pyrrole Documentation

Felipe S. S. Schneider

Apr 16, 2023
## Contents

1 Usage example 3

2 Installation 5

3 Documentation 7
   3.1 Getting started .......................................................... 7
      3.1.1 Solubility of acetic acid ........................................ 7
   3.2 Using data objects ...................................................... 9
      3.2.1 Reading local files ................................................. 9
      3.2.2 Reading the web ..................................................... 10
   3.3 Systems and equations .................................................. 11
      3.3.1 The ChemicalSystem object revisited ......................... 11
      3.3.2 The ChemicalEquation object ................................... 11

4 Indices and tables 13
WARNING: this project has been retired. Please migrate to geem-lab/overreact.

A Python package for solving chemical problems with computational modeling.
As a usage example, let’s calculate the energy barrier involved in nitrogen inversion in ammonia.

We do this in three simple steps (only eight lines of code):

1. **Get the data**

We first obtain the raw data, which will later be fed to our chemical model. Below we read computational chemistry logfiles of both ground and transition states.\(^1\)

```python
>>> from pyrrole.atoms import read_cclib, create_data
>>> gs = read_cclib("data/ammonia/ammonia.out", name="NH3(g)")
```

\(^1\) Optimizations and frequency calculations of both ammonia and the planar transition state were performed at PBEn-3c using the ORCA electronic structure package (version 4.0.1.2). Logfiles can be found in the project’s repository.
Pyrrole uses `cclib` for reading logfiles, which is compatible with all major computational chemistry packages. You could also want to read tabular data from a file (or even from the web) using `pandas`.

2. **Specify the model**

We now describe our model. This is accomplished through chemical equations:

```python
>>> from pyrrole import ChemicalEquation
>>> equation = ChemicalEquation("NH3(g) -> NH3(g)#", data)
```

While model above consists of a single `ChemicalEquation`, you could create complex models with multiple chemical equations with `ChemicalSystem` objects. You might want to store your complex models in text files too.

3. **Obtain the results**

Simply let pyrrole calculate the energy barrier:

```python
>>> results = equation.to_series()
>>> results["freeenergy"] * 2625.4996382852164  # Hartree to kJ/mol
19.30952589472923
```

(As a side note, the reference value is 21.162 kJ/mol\(^2\).)

Interested? Have another example.

CHAPTER 2

Installation

You can get the library directly from PyPI:

```
$ pip install pyrrole
```
3.1 Getting started

In simple terms, the basic usage of pyrrole can be outlined in three steps:

1. Create a data object (this is actually a pandas.DataFrame).
2. Create a ChemicalSystem object.
3. Manipulate a ChemicalSystem object.

In order to understand each of them, let’s walk through core API concepts as we tackle one everyday use case: the calculation of solvation free energy of acetic acid in water.

3.1.1 Solubility of acetic acid

Let’s say that, after optimization and frequency calculations of acetic acid were done (both in vacuuo and using an implicit solvation method\(^1\)), we wanted to calculate the solvation energy of acetic acid in water. This simple model perfectly exemplifies the usage of pyrrole, starting with the creation of a data object.

**The data object**

The data object consists of a pandas.DataFrame whose records represent chemical species. For our specific problem, we read logfiles (using the read_cclib function, which parses logfiles with the cclib library) and store them in the required tabular form (using create_data):

```python
>>> from pyrrole.atoms import read_cclib, create_data
>>> gas = read_cclib("data/acetate/acetic_acid.out", name="AcOH(g)")
>>> aquo = read_cclib("data/acetate/acetic_acid@water.out", name="AcOH(aq)")
>>> data = create_data(gas, aquo)
```

\(^1\) Calculations were done at PBEh-3c/SMD (water) using the ORCA electronic structure package (version 4.0.1.2). Logfiles can be found in the project’s repository.
Each row of data above contains information found in a single logfile:

```python
>>> columns = ["enthalpy", "entropy", "freeenergy"]
>>> data[columns]  # doctest: +NORMALIZE_WHITESPACE
    enthalpy    entropy    freeenergy
   name
 AcOH (g) -228.533374 0.031135 -228.564509
 AcOH (aq) -228.544332 0.030936 -228.575268
```

The energy values above are in Hartree, which is the convention in the cclib project. Learn more about data objects in Using data objects.

### The ChemicalSystem object

We are now in position to define our chemical system with ChemicalSystem. Our model consists of a single equilibrium between gas phase and aqueous acetic acid:

```python
>>> from pyrrole import ChemicalSystem
>>> system = ChemicalSystem("AcOH(g) <=> AcOH(aq)", data)
>>> system
ChemicalSystem(["AcOH(g) <=> AcOH(aq)"])
```

### Usage of ChemicalSystem

ChemicalSystem objects can be manipulated in a variety of ways. For instance, they can be converted to pandas.DataFrame (with the ChemicalSystem.to_dataframe method):

```python
>>> reactions = system.to_dataframe()
>>> reactions[columns]  # doctest: +NORMALIZE_WHITESPACE
chemical_equation
    enthalpy    entropy    freeenergy
   chemical
 AcOH (g) <=> AcOH (aq) -0.010958 -0.000198 -0.010759
```

Again, energy values are given in Hartree. Conversion factors can be used for handling other units (with the help of the scipy.constants module):

```python
>>> from scipy.constants import kilo, N_A, physical_constants
>>> hartree, _, _ = physical_constants["Hartree energy"]
>>> factor = hartree * N_A / kilo  # Hartree to kJ/mol
>>> factor
2625.4996382852164
```

The calculated factor can be used to convert a whole table if so desired:

```python
>>> reactions[columns] * factor  # doctest: +NORMALIZE_WHITESPACE
chemical_equation
    enthalpy    entropy    freeenergy
   chemical
 AcOH (g) <=> AcOH (aq) -28.76991 -0.521109 -28.248775
```

(By the way, the reported experimental value for the solvation free energy of acetic acid in water is \(-28.0 \text{ kJ/mol}\)).

Now we’re ready to start Using data objects.

---

### 3.2 Using data objects

Any pandas.DataFrame indexed by names of chemical species is a valid data object in pyrrole:

```python
>>> import pandas as pd
>>> data = pd.DataFrame(
...                   [{'name': 'CO3-2(aq)', 'freeenergy': -527.8},
...                   {'name': 'HCO3-(aq)', 'freeenergy': -586.85},
...                   {'name': 'H2CO3(aq)', 'freeenergy': -623.1},
...                   {'name': 'OH-(aq)', 'freeenergy': -157.2},
...                   {'name': 'H2O(l)', 'freeenergy': -237.14}])
>>> data = data.set_index('name')
>>> data
# doctest: +NORMALIZE_WHITESPACE
freeenergy
name
CO3-2(aq)  -527.80
HCO3-(aq)  -586.85
H2CO3(aq)  -623.10
OH-(aq)    -157.20
H2O(l)     -237.14
```

The pandas library, a dependency of pyrrole, can be used to create data objects. Below are examples of creating data objects from different sources.

#### 3.2.1 Reading local files

Pandas can read data sets in various formats, such as comma-separated values (CSV), Google BigQuery, Hierarchical Data Format (HDF), JavaScript Object Notation (JSON), Microsoft Excel, and many other supported format types:

```python
>>> data = pd.read_hdf("data/acetate/data.h5")
>>> data[['jobfilename', 'freeenergy', 'enthalpy']]
   jobfilename freeenergy enthalpy
0  data/acetate/acetate.out -228.000450 -227.969431
1  data/acetate/acetate@water.out -228.120113 -228.089465
2  data/acetate/acetic_acid.out -228.564509 -228.533374
3  data/acetate/acetic_acid@water.out -228.575268 -228.544332
```

Pyrrole requires indices to represent names of chemical species, which is, like above, not always the case. Setting meaningful indices can be accomplished by feeding a custom function to `data.apply`:

```python
>>> def update(series):
...     """Compute a new column 'name' and add it to row.""
...     series['name'] = (series['jobfilename']
...                       .replace('data/acetate/', '')
...                       .replace('.out', ''))
...     series['name'] = (series['name']
...                       .replace('acetate', 'AcO-')
...                       .replace('acetic_acid', 'AcOH'))
...     series['name'] = series['name'].replace('@water', '(aq)')
...     if '(aq)' not in series['name']:
...         series['name'] += "(g)"
...     return series
```

The function above should be applied to the data object, which can then be reindexed:

---

1 Obtained from standard Gibbs free energy of formation.
The data object is now ready to be used:

```python
>>> from pyrrole import ChemicalSystem

... system = ChemicalSystem(['AcO-(g) <=> AcO-(aq)',
...                           'AcOH(g) <=> AcOH(aq)'],
...                           data['freeenergy'])

... system.to_dataframe()  # doctest: +NORMALIZE_WHITESPACE

chemical_equation freeenergy
AcO-(g) => AcO-(aq) -0.119663
AcOH(g) => AcOH(aq) -0.010759
```

In Getting started, we showed how to use create_data to produce a data object by reading output files from computational chemistry programs. Reading lots of logfiles is slow, which is why storing the data in a file translates to faster retrievals later. This can be accomplished with ccframe, a command-line tool that is part of cclib (a dependency of pyrrole). In fact, the file data.h5 used in the example above was produced using ccframe:

```
$ ccframe -O data/acetate/data.h5 data/acetate*out
 data/acetic_acid*out
```

Learn more about ccframe in both its help page (`$ ccframe -h`) and documentation.

### 3.2.2 Reading the web

There’s a lot of freely available data on the internet. For instance, NIST offers enthalpies of formation at 0K (in kJ/mol). Luckily, pandas supports reading HTML tables directly:

```python
>>> url = "https://cccbdb.nist.gov/hf0k.asp"

... data = pd.read_html(url, header=0)[3]  # fourth table in page

... data = data.set_index("Species")

... data = data[['Name', "Hfg 0K", "DOI"]]

... data.head()  # doctest: +NORMALIZE_WHITESPACE

Name Hfg 0K DOI
Species
D Deuterium atom 219.8 NaN
H Hydrogen atom 216.0 10.1002/bbpc.19900940121
H+ Hydrogen atom cation 1528.1 NaN
D2 Deuterium diatomic 0.0 NaN
H2 Hydrogen diatomic 0.0 10.1002/bbpc.19900940121
```

This data allows us to calculate the bond-dissociation enthalpy of the hydrogen molecule at 0K, for instance:

```python
>>> from pyrrole import ChemicalEquation

... equation = ChemicalEquation("H2 -> 2 H", data)

... equation.to_series()

Hfg 0K 432.0
Name: H2 -> 2 H, dtype: float64
```
That’s 432 kJ/mol, or 103.3 kcal/mol.

It’s time to take a deeper look at Systems and equations.

### 3.3 Systems and equations

#### 3.3.1 The ChemicalSystem object revisited

In Getting started, we saw the basics of chemical systems. Drawing.

Internally, a ChemicalSystem object consists of individual ChemicalEquation objects, which can be manipulated on their own.

#### 3.3.2 The ChemicalEquation object

Single chemical equations in pyrrole are handled by ChemicalEquation objects. A special mini-language is used to define chemical equations in a way that makes it easy to simply copy and paste from the web. For instance, the following metal displacement was obtained from a Wikipedia entry:

```python
>>> from pyrrole import ChemicalEquation
>>> half_zinc = ChemicalEquation('Zn(s) -> Zn+2(aq) + 2 e-')
>>> half_copper = ChemicalEquation('Cu+2(aq) + 2 e- <- Cu(s)')
```

ChemicalEquation objects can be manipulated just like vectors, i.e., summed and multiplied by scalar values:

```python
>>> half_zinc - half_copper
ChemicalEquation('Cu+2(aq) + Zn(s) -> Cu(s) + Zn+2(aq)')
```

Stoichiometry coefficients can be obtained individually:

```python
>>> half_zinc.coefficient['e-']
2.0
```

There’s no need to use chemical formulae for chemical species. Any mix of printable characters can be used:

```python
>>> ChemicalEquation('cis-A <=> trans-A')
ChemicalEquation('cis-A <=> trans-A')
```
CHAPTER 4

Indices and tables

- genindex
- modindex
- search