pyBio Documentation

Release 0.1.dev

Genadij Razdorov <genadijrazdorov@gmail.com>

Dec 12, 2017

Contents

URL: http://pybio.rtfd.io/

Version: 0.1.dev

Documentation build date: Dec 12, 2017 *pyBio* is a toolkit for biology related computations.

> **Warning:** *pyBio* is in a pre-development phase.
>
> We are designing and prototyping API. Any interface should be considered unstable, any implementation is here just for show case.

pyBio will try to provide common infrastructure useful for any computation related to biology.

CHAPTER 1

Features

- Documented
- Tested
- Reproducible computations
- Biology & chemistry basic computational infrastructure
- Expandable to any biology related application
- Integrated with major biological databases

Note: *Glycobiology based on mass spectrometry* is first application due to authors current field of work

CHAPTER 2

Installation

Install pyBio by running:

```
pip install project
```

CHAPTER 3

Contribute

- Issue Tracker: https://github.com/genadijrazdorov/pybio/issues
- Source Code: https://github.com/genadijrazdorov/pybio

CHAPTER 4

Support

If you are having issues, please let us know.

CHAPTER 5

License

The project is licensed under the MIT license.

CHAPTER 6

Contents

6.1 Glycopeptide example

```
>>> from pybio import Peptide, Glycan, Molecule

>>> # Immunoglobulin heavy constant gamma 1 (Homo sapiens)
>>> # P01857[176 - 184]
>>> peptide = Peptide("EEQYNSTYR")
>>> peptide
Peptide('EEQYNSTYR')
```

```
>>> # Major IgG1 Fc N-glycan
>>> G0F = Glycan(composition="H3N4F")
>>> G0F
Glycan(composition='H3N4F')
```

```
>>> # glycopeptide build
>>> glycopeptide = Molecule()
>>> glycopeptide.bonds[peptide, G0F] = "glycosidic"
```

6.2 Molecule

> Any constitutionally or isotopically distinct atom, molecule, ion, ion pair, radical, radical ion, complex, conformer etc., identifiable as a separately distinguishable entity. (Molecular entity)
>
> —http://goldbook.iupac.org/html/M/M03986.html

Molecular entity is represented as molecular *graph*. Molecular graph is a set of a chemical groups[3] connected by chemical bonds[4].

[3] A defined linked collection of atoms or a single atom within a molecular entity. (http://goldbook.iupac.org/html/G/G02705.html)

[4] ... a chemical bond between two atoms or groups of atoms in the case that the forces acting between them are such as to lead to the formation

```
>>> from pybio import Molecule, Atom
>>> from pybio.molecule import Group
```

6.2.1 Building a molecule

Building a simple molecule:

```
>>> methane = Molecule()

>>> C = methane.add("C")
>>> H = methane.add("H")
>>> methane.bonds[C, H] = 1
>>> # Add hydrogens and bind them to carbon
... for __ in range(3):
...     methane.bonds[C, "H"] = 1
...

>>> methane
<pybio.molecule.Molecule object at 0x...>
```

Groups can be atoms and/or molecules.

Building a ammonium chloride molecule:

```
>>> # NH4 polyatomic ion

>>> NH4 = Molecule()
>>> N = NH4.add("N")

>>> for __ in range(4):
...     NH4.bonds[N, "H"] = True
...

>>> NH4.charge = +1

>>> # complete molecule

>>> NH4Cl = Molecule()

>>> # Bind NH4+ with Cl-
>>> NH4Cl.bonds[NH4, "Cl-"] = True
```

6.2.2 Working with a molecule

Atoms and groups are accessible via groups attribute:

```
>>> sorted([atom() for atom in methane.groups])
[Atom('H'), Atom('H'), Atom('H'), Atom('H'), Atom('C')]

>>> [group() for group in NH4Cl.groups]
[Molecule(), Atom('Cl', charge='-')]
```

of an aggregate with sufficient stability to make it convenient for the chemist to consider it as an independent 'molecular species'. (http://goldbook.iupac.org/html/B/B00697.html)

Bonds are accessible as dictionary:

```
>>> methane.bonds[C, H]
1

>>> methane.bonds[H, C] is methane.bonds[C, H]
True
```

Order of a molecule (number of groups):

```
>>> len(methane)
5

>>> len(NH4Cl)
2
```

Size of a molecule (number of bonds):

```
>>> len(methane.bonds)
4
```

Degree of a group (number of incident bonds):

```
>>> len(methane[C])
4
```

Membership testing:

```
>>> # Concreate group
... C in methane
True
>>> N in NH4Cl
True

>>> # Atom value
... Atom("C") in methane
True
>>> Atom("N") in NH4Cl
True

>>> # faster if C is not in methane
... C in methane.groups
True

>>> # bond testing
... (C, H) in methane
True
>>> # faster
... (C, H) in methane.bonds
True
```

Walking over atoms:

```
>>> list(NH4Cl.walk(N))
[Atom('N+'), Atom('H'), Atom('H'), Atom('H'), Atom('H'), Atom('Cl-')]
```

6.3 Atom

> Smallest particle still characterizing a chemical element. It consists of a nucleus of a positive charge (Z is the proton number and e the elementary charge) carrying almost all its mass (more than 99.9%) and Z electrons determining its size.
>
> —http://goldbook.iupac.org/html/A/A00493.html

Atom API:

```
>>> from pybio import Atom

>>> # Equality
... Atom("C") == Atom("C")
True

>>> # Identity
... Atom("C") is Atom("C")
False

>>> # Membership
... Atom("C") in {Atom("C")}
True
```

6.4 Molecular Formula

https://en.wikipedia.org/wiki/Chemical_formula#Molecular_formula

https://en.wikipedia.org/wiki/Chemical_formula#Hill_system

```
>>> from pybio import Formula, Atom

>>> methane = Formula("CH4")
```

Representing & printing:

```
>>> methane
Formula('CH4')

>>> print(methane)
CH4
```

Individual element testing, counting:

```
>>> Atom("C") in methane
True

>>> Atom("Ca") in methane
False

>>> methane[Atom("H")]
4
```

Ions:

```
>>> Formula("[N+]H4")
Formula('H4N+')
```

Isotopes:

```
>>> Formula("H4[13C]")
Formula('[13C]H4')
```

Hill system:

```
>>> for formula in "IBr Cl4C IH3C C2BrH5 H2O4S".split():
...     print(formula, "->", Formula(formula))
IBr -> BrI
Cl4C -> CCl4
IH3C -> CH3I
C2BrH5 -> C2H5Br
H2O4S -> H2O4S
```

6.5 Graph

Python graph library:

- http://networkx.github.io/

(Python) graph sites:

- https://www.python.org/doc/essays/graphs/
- https://wiki.python.org/moin/PythonGraphApi
- http://www.linux.it/~della/GraphABC/
- https://www.python-course.eu/graphs_python.php
- https://en.wikipedia.org/wiki/Graph_(abstract_data_type)
- https://en.wikipedia.org/wiki/Adjacency_list
- http://www.ics.uci.edu/~eppstein/161/960201.html
- https://pkch.io/2017/03/31/python-graphs-part1/
- https://pkch.io/2017/04/12/python-graphs-part2/

6.5.1 Problem definition

Molecular graph can not be directly defined in python as comparable atoms connected with chemical bonds, because of python invariant that *equal objects have same hash value*.

This can be explained on a **1-1** example:

```
>>> import networkx as nx
>>> graph = nx.Graph()
>>> graph.add_edge(1, 1)

>>> list(graph.nodes())
[1]
```

```
>>> list(graph.edges())
[(1, 1)]
```

What we built instead of **1-1** is a multigraph with a self loop **1]**.

Let us now look at chemical example of ethane: **H3CCH3**. As you can see, we have 2 carbons (C), and 6 hydrogens (H):

```
>>> import networkx as nx
>>> ethane = nx.Graph()
>>> ethane.add_edge("C", "C")
>>> for __ in range(6):
...     ethane.add_edge("H", "C")

>>> S = sorted
>>> S(ethane.nodes())
['C', 'H']

>>> S(S((left, right)) for left, right in ethane.edges())
[['C', 'C'], ['C', 'H']]
```

We got **H-C]**. We can separately track atoms, and use list indices as nodes:

```
>>> import networkx as nx

>>> #        0 2 4 6
>>> atoms = "HHHHHHCC"
>>> ethane = nx.Graph()

>>> #               C  C
>>> ethane.add_edge(6, 7)
>>> for i in range(2):
...     for H in range(3):
... #                          H     C
...         ethane.add_edge(H+i*3, i+6)
...

>>> # nodes from atoms mapping
>>> for i in sorted(ethane.nodes()):
...     print(atoms[i], end=" ")
...
H H H H H H C C

>>> # edges from atoms mapping
>>> for i, j in ethane.edges():
...     print(atoms[i], "-", atoms[j], sep="", end=" ")
...
C-C C-H C-H C-H C-H C-H C-H
```

6.5.2 Needed API

Based on http://www.linux.it/~della/GraphABC/ adding Node instance as wrapper for any object.

```
>>> from pybio.tools.graph import Graph, Node
>>> ethane = Graph()
```

Graph has set of nodes:

```
>>> ethane.nodes == set()
True
```

... and dict of edges:

```
>>> ethane.edges == dict()
True
```

Adding nodes to graph:

```
>>> C1, C2 = C = [ethane.add("C") for __ in range(2)]
```

Graph Node

Node instance wraps any object holding actual node value.

```
>>> # Node type
>>> isinstance(C1, Node)
True

>>> # Node value
>>> C1()
'C'

>>> # Node comparison
>>> C1 is C2, C1 == C2
(False, False)

>>> # Values comparison
>>> C1() is C2(), C1() == C2()
(True, True)
```

Connecting nodes:

```
>>> ethane.edges[C1, C2] = True

>>> for i in range(2):
...     for __ in range(3):
...         ethane.edges[C[i], "H"] = True
...
```

Nodes:

```
>>> {C1, C2} <= ethane.nodes
True
```

Accessing node values:

```
>>> S = sorted
>>> for node in S(node() for node in ethane.nodes):
...     print(node, end=" ")
...
C C H H H H H H
```

Accessing edges:

```
>>> for edge in S(S([left(), right()]) for left, right in ethane.edges):
...     print("{}-{}".format(*edge), end=" ")
...
C-C C-H C-H C-H C-H C-H C-H
```

Membership testing:

```
>>> "C" in ethane
True

>>> C1 in ethane
True
```

6.6 API

6.6.1 pybio package

Subpackages

pybio.tools package

Submodules

pybio.tools.mock module

pybio.tools.mock.**mock** (*function*)

Module contents

Submodules

pybio.atom module

class pybio.atom.**Atom** (*symbol*, *mass_number=None*, *charge=None*)

Bases: object

Smallest particle still characterizing a chemical element

Parameters

- **symbol** (*str*) – Atomic symbol
- **mass_number** (*int, optional*) – Atomic mass number (A)
- **charge** (*int, optional*) – Charge number

atomic_number

int – Atomic number (Z)

charge_regex = '([-+]\\d*)?'

mass_number_regex = '(\\d+)?'

symbol_regex = '([A-Z][a-z]{,2})'

class pybio.atom.**Electron**
Bases: *pybio.atom.Atom*

Subatomic elementary particle with a negative elementary electric charge

References

- http://goldbook.iupac.org/html/E/E01975.html
- https://en.wikipedia.org/wiki/Electron

atomic_number = 0

charge = -1

mass_number = 0

symbol = ''

pybio.formula module

class pybio.formula.**Formula**(*formula=None*)
Bases: collections.OrderedDict

Molecular formula

Parameters **formula** (*str or Mapping*) – formula as a string or Atom-to-count mapping

pybio.formula.**formula**(*composition*)

pybio.glycan module

class pybio.glycan.**Glycan**(*notation=None*, *composition=None*)
Bases: *pybio.molecule.Molecule*

pybio.molecule module

class pybio.molecule.**Group**(*value*)
Bases: pybio.tools.graph.Node

single node in a molecule

A defined linked collection of atoms or a single atom within a molecular entity.

—http://goldbook.iupac.org/html/G/G02705.html

class pybio.molecule.**Molecule**
Bases: pybio.tools.graph.Graph

Molecular entity

Any constitutionally or isotopically distinct atom, molecule, ion, ion pair, radical, radical ion, complex, conformer etc., identifiable as a separately distinguishable entity.

—http://goldbook.iupac.org/html/M/M03986.html

Node
alias of *Group*

add(*group*)

pybio.peptide module

class pybio.peptide.**Peptide**(*sequence*)
Bases: *pybio.molecule.Molecule*

Module contents

6.7 How to contribute

6.7.1 Code contribution

Based on:

- http://nvie.com/posts/a-successful-git-branching-model/
- https://help.github.com/articles/fork-a-repo/
- https://help.github.com/articles/about-pull-requests/
- https://help.github.com/articles/allowing-changes-to-a-pull-request-branch-created-from-a-fork/
- https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow
- https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Note: Following procedure is for Windows platform

If you want to contribute your code to *pyBio*, please follow this steps:

Setup

1. Fork *pyBio*
   (a) Navigate to https://github.com/genadijrazdorov/pybio
   (b) Fork your own copy of *pyBio* by cliking on *Fork* button
   (c) You are navigated to your copy GitHub page
2. Clone your fork locally
   (a) Click on *Clone or download* button
   (b) Copy your fork url by clicking on *Copy to clipboard* button
   (c) Open *Git Bash* console
   (d) Change directory to desired one:

```
$ cd path/to/local/clone/parent
```

   (e) Clone your fork:

```
$ git clone <Shift+Ins>
```

3. Add upstream repo

```
$ cd pybio
$ git remote add upstream https://github.com/genadijrazdorov/pybio.git
```

Feature development

1. Checkout develop branch:

```
$ git checkout develop
```

2. Sync with upstream:

```
$ git pull upstream
```

3. Create and checkout new *feature* branch:

```
$ git checkout -b new-feature-name
```

4. Develop
   (a) Create documentation, unit-tests and implementation for new feature
   (b) Check your implementation by running doctests and pytests
   (c) Add and commit your changes
5. Push your changes to origin:

```
$ git push -u origin
```

6. Create pull request online
   (a) Follow instructions from: https://help.github.com/articles/creating-a-pull-request-from-a-fork/
7. Discuss and modify your code with *pyBio* developers
8. After *feature* branch is merged sync your fork
   (a) Pull from upstream:

```
$ git checkout develop
$ git pull upstream
```

   (b) Push to origin

```
$ git push
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index