AChemKit Documentation

Release 0.1

Adam Faulconbridge

September 09, 2011

CONTENTS

README

1.1 Description

PyAChemKit is a collection of Artificial Chemistry software written in Python - a library and collection of tools.

Artificial Chemistry (AChem) is a spin-off topic of Artificial Life. AChem is aimed at emergence of life from non-living environment - primordial soup etc.

1.2 Installation

To install on Ubuntu Linux, run

```
sudo easy_install -U AChemKit
```

This package should work on other Linux distributions and versions of Windows, but is untested.

This package requires the following:

- Python >= 2.6 http://www.python.org/

Some features use the following:

- NetworkX

- GraphViz http://www.graphviz.org/

Optionally, the following can be installed to improve performance:

- Psyco http://psyco.sourceforge.net

- PyPy http://codespeak.net/pypy

1.3 Source

The latest version of the source code is available from https://github.com/afaulconbridge/PyAChemKit

The source code additionally requires the following:

- Sphinx >= 1.0 http://sphinx.pocoo.org/

- Graphviz http://www.graphviz.org/

- Make http://www.gnu.org/software/make/

- LaTeX http://www.latex-project.org/
- PyLint >=0.13.0 http://www.logilab.org/project/pylint/
- Coverage http://nedbatchelder.com/code/coverage/

For a Debian-based Linux distrbution — e.g. Debian, Ubuntu — these can be installed / updated with:

```
make setup
```

(Note, LaTeX is not installed via this method because it is very large. Run `sudo apt-get install texlive-full` if you want to be able to compile the PDF documentation.)

There is a makefile that will run some useful tasks for you (generate documentation, test, benchmark). This can be accessed by running the following command:

```
make help
```

1.4 Copyright

This project is licensed under a modified-BSD license. See the fie `COPYRIGHT` for details.

FILE FORMATS

AChemKit uses a number of text file formats for data storage and as intermediates between various components.

2.1 *.chem* Format

The *.chem* format is used to abstractly describe a chemistry in terms of reactions. Each reaction is represented as a single line, separated into reactants and products by the symbol ->. Blank lines and lines starting with # are ignored - this is useful to remove reactions temporarily or to add documentation. Additional white-space within a line is also ignored.

Reactants and/or products can contain multiple molecular species, separated by +. No explicit restrictions are placed on the names of molecular species, but it is recommended to be restricted to alpha-numeric characters (upper-case and lower-case) as well as *()* and *[]*. In particular, the following should be avoided: +, -, >. At this time, escaping or quoting is not supported.

The reactant/product separator -> is also used to represent the rate constant of the reaction, if specified. This is done by a floating-point number between the - and > characters. For example, *-2.0>* represents a reaction with a rate constant of 2.0. Integers can also be used, but will be converted to floating-point, e.g. *-2>* is a rate constant of 2.0. If a rate constant is not specified, it defaults to 1.0. Rate constants <=0.0 are not valid, and in some situations rate constants >1.0 may not be either.

It is not valid to have more than one reaction with the same combination of reactants and products - in any order and with any rate. For example, *A + B -> C* cannot appear in the same file as *B + A -2.0> C*.

It is not valid to have the same molecular species in both reactants and products. For example *A + B -> B + A* is not valid.

Either the reactants or products can be omitted from a reaction, e.g. *A ->*. This corresponds to material being added or removed from the system.

When a *.chem* file is generated, reactants and products of all reactions are sorted by their string representations, and then reactions are sorted before being output.

Below is a valid example *.chem* file:

```
#This is a comment

#Here are some blank lines

#Now for some reactions
A + B -> AB

#The same molecular species can be used again
A -> C
```

```
#A reversible reaction can be specified by
#stating the inverse of an existing reaction
C -> A

#Some reactions have rate constants other than 1.0
D -10.0> E
F -0.01> G

#Any number of molecular species can be in the reactants and/or products
#The same molecular species can be in the reactants and/or products multiple times
Spam + Spam + Spam + Eggs + Spam -> Breakfast
Catalyst + Input -> Catalyst + Output

#Things can be created and destroyed
-> TheUniverse
TheUniverse ->
```

PROJECT MODULES

3.1 AChemKit Package

3.1.1 `AChemKit` Package

Detailed documentation for all the components of AChemKit. It is generated directly from the source code, so should be up-to-date.

3.1.2 `bucket` Module

Library for working with `Bucket` objects; instances of a simulation of an Artificial Chemistry.

Various tools for going between `ReactionNetwork` and `Bucket` objects, as well an analysing the data within `Bucket` objects.

class `AChemKit.bucket.`**`Bucket`**(*events*)

Bases: `object`

Event history of a simulation of an Artificial Chemistry.

classmethod **`from_file`**(*file*)

Alternative constructor that accepts a `file()` object (or equivalent).

Source must be formatted as a bucket log file, see *bucket_file_format*.

classmethod **`from_filename`**(*infilename*)

Wrapper around `from_file()` that opens the provided filename as a file to read.

classmethod **`from_string`**(*instr*)

Wrapper around `from_file()` that uses `StringIO`.

`reactionnet`

A property that generates and caches a `ReactionNetwork` object representing the reaction network exhibited by this bucket.

Rates are calculated based on repeats of events in this bucket. This may have a large sampling error depending on how many repeates there were.

class `AChemKit.bucket.`**`Event`**(*time*, *reactants*, *products*)

Bases: `object`

Mini-class for tracking events (instances of a reaction) within a `Bucket`.

Supports comparisons, is hashable, is immutable.

> **products**
>
> **reactants**
>
> **time**

3.1.3 `properties` Module

Functions for testing properties of particular reaction networks.

These functions expect an object of class `AChemKit.reactionnet.ReactionNetwork` or subclass or something with an equivalent API. It does not enforce this however, so you may use custom classes with the same API.

Some of these properties have logical prerequisites, but these are not tested for explicitly.

AChemKit.properties.**get_autocatalysis_direct**(*rn*)
> Tests for reactions where some species is both consumed and produced by the same reaction, and is produced more than it is consumed.
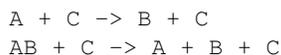>
> For example:
>
> ```
> A + C -> A + A
> ```
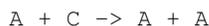
AChemKit.properties.**get_catalysis_direct**(*rn*)
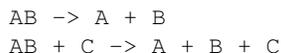> Tests for reactions where some species is both consumed and produced by the same reaction.
>
> For example:
>
> ```
> A + C -> B + C
> AB + C -> A + B + C
> ```

AChemKit.properties.**get_decomposition**(*rn*)
> Returns all reactions where reactants combine to produce more products.
>
> For example:
>
> ```
> AB -> A + B
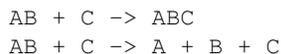> AB + C -> A + B + C
> ```
>
> Works as an iterator filter.

AChemKit.properties.**get_divergence**(*rn*)
> Tests for divergent reactions.
>
> Divergent reactions are where the same reactants have multiple possible collections of products.
>
> For example:
>
> ```
> AB + C -> ABC
> AB + C -> A + B + C
> ```
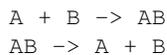
AChemKit.properties.**get_reversible**(*rn*)
> Tests for a pair of reactions where the products of one is the reactants of the other and visa vera.
>
> For example:
>
> ```
> A + B -> AB
> AB -> A + B
> ```

AChemKit.properties.**get_synthesis**(*rn*)
> Returns all reactions where reactants combine to produce fewer products.
>
> May also be called a *combination reaction*.

For example:

```
A + B -> AB
A + B + C -> AB + D
```

Works as an iterator filter.

AChemKit.properties.**has_autocatalysis_direct**(*rn*)
> Tests if any reactions are direct autocatalysis. See get_autocatalysis_direct() for definition.

AChemKit.properties.**has_catalysis_direct**(*rn*)
> Tests if any reactions are direct catalysis. See get_catalysis_direct() for definition.

AChemKit.properties.**has_decomposition**(*rn*)
> Tests if any reactions are decomposition. See get_decomposition() for definition.

AChemKit.properties.**has_divergence**(*rn*)
> Tests if any reactions are divergent. See get_divergence() for definition.

AChemKit.properties.**has_reversible**(*rn*)
> Tests if any reactions are reversible. See get_reversible() for definition.

AChemKit.properties.**has_synthesis**(*rn*)
> Tests if any reactions are synthesis. See get_synthesis() for definition.

AChemKit.properties.**has_varying_rates**(*rn*)
> Tests that the reaction network has different rates for different reactions.
>
> To get the range of rates a reaction network spans, use:
>
> ```
> span = max(rn.rates.values())-min(rn.rates.values())
> ```

AChemKit.properties.**not_conservation_mass**(*rn*)
> Tests for violation of conservation of mass.
>
> Note, we can show that a reaction network breaks conservation of mass, but not prove that it obeys it.
>
> This is done by arranging all the molecules into a partially ordered set - a tree from largest to smallest. If this cannot be done, then conservation of mass must be violated. If this can be done, then conservation of mass may or may not apply - it cannot be said for certain.
>
> This requires that each molecular species can be separated into individual atoms. If this is not possible, then this function will not work.
>
> NEEDS TO BE WRITTEN
>
> (Credit to Adam Nellis for algorithm)

3.1.4 `properties_motif` Module

Functions for testing properties of particular reaction networks.

These functions expect an object of class AChemKit.reactionnet.ReactionNetwork or subclass or something with an equivalent API. It does not enforce this however, so you may use custom classes with the same API.

Some of these properties have logical prerequisites, but these are not tested for explicitly.

Unlike AChemKit.reactionnet.properties, the functions here are not boolean, but rather return grops of things

AChemKit.properties_motif.**find_loops**(*rn*)
> Loops are collections of molecules where a path exists to themselves through linkages of shared products and reactants.

For example:

```
A -> B
B -> C
C -> A
```

3.1.5 `properties_wnx` Module

Various functions that interact with NetworkX (http://networkx.lanl.gov/)

This is used because its a fast graph library with lots of nice algorithms in it.

AChemKit.properties_wnx.**make_catalysis_graph**(*rn*, *min_shared=3*)
> For each reactant and each product in each reaction, they are linked if they share at minimum subsequence length and if the reactant is a catalsyst.

AChemKit.properties_wnx.**make_linkage_graph**(*rn*, *min_shared=3*)
> For each reactant and each product in each reaction, they are linked if they share at minimum subsequence length.

3.1.6 `randomnet` Module

Functions that construct random `ReactionNetwork` instances by various methods.

This is designed to provide null hypothesis data for various situations and metrics. As with random graphs, there is no single best way to generate a random reaction network.

AChemKit.randomnet.**Linear**(*natoms*, *maxlength*, *pform*, *pbreak*, *directed=True*, *rates=1.0*,
            cls=<class 'AChemKit.reactionnet.ReactionNetwork'>, *rng=None*)
> Generates a random `ReactionNetwork` from molecules that are strings of atoms and can join together or break apart.

> Based on the paper Autocatalytic sets of proteins. 1986. Journal of Theoretical Biology 119:1-24 by Kauffman, Stuart A. but without the explicit catalytic activity.

> Arguments:

> **natoms** Number of atoms to use. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution), or a dict of value:weight which will be sampled from.

> ---

> **Note:** `AChemKit.reactionnet.ReactionNetwork` tracks molecules by their reactions, so if a molecule is not part of any reaction it will not appear at all e.g. in `seen`.

> ---

> **maxlength** Maximum number of atoms in a molecule. If this is None, then they are unbounded; this might cause problems with a computational explosion. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution), or a dict of value:weight which will be sampled from.

> **pform** Probability that a pair of molecules will join together per orientation. Must be between 0 and 1. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution), or a dict of value:weight which will be sampled from.

> **pbreak** Probability that any pair of atoms will break. Must be between 0 and 1.Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution), or a dict of value:weight which will be sampled from.

directed If false, molecules have no intrinsic direction so AlphBeta is equivlanet to BetaAlpha.

rates Rate of each reaction in the reaction network. Can be a single value, or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution), or a dict of value:weight which will be sampled from.

cls Alternative class to use for constructing the return rather than `AChemKit.reactionnet.ReactionNetwork`.

rng Random number generator to use. If not specifed, one will be generated at random.

`AChemKit.randomnet.`**`Uniform`**(*nmols*, *nreactions*, *nreactants*, *nproducts*, *rates=1.0*, *cls=<class 'AChemKit.reactionnet.ReactionNetwork'>*, *rng=None*)
  Generates a random `ReactionNetwork` by assigning reaction randomly between all molecular species.

  Arguments:

  nmols Number of molecules in the reaction network.

  Note: `AChemKit.reactionnet.ReactionNetwork` tracks molecules by their reactions, so if a molecule is not part of any reaction it will not appear at all e.g. in `seen`. This could lead to differences from *nmols*.

  ---

  nreactions Number of reaction in the reaction network.

  Note: The value of *nreactions* is the number of times a reaction will be added to the `ReactionNetwork`. If it is already in the `ReactionNetwork`, it will be replaced. This can lead to `AChemKit.reactionnet.ReactionNetwork` with less than *nreactions* reactions.

  ---

  nreactants Number of reactants for each reaction in the reaction network. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution).

  Note: If this is a tuple/list it will be sampled for each reaction.

  ---

  nproducts Number of products for each reaction in the reaction network. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution).

  If this is None, then *nreactants* must be a list/tuple of tuples of (*nreactants*, *nproducts*) pairs that will be uniformly sampled from. Or *nreactants* must be a dictionary with keys of (*nreactants*, *nproducts*) and values of weightings, which will be sampled from.

  Note: If this is a tuple/list it will be sampled for each reaction.

  ---

  rates Rate of each reaction in the reaction network. Can be a single value or a tuple/list which will be uniformly sampled from (duplicates can be used to give a non-uniform distribution).

  Note: If this is a tuple/list it will be sampled for each reaction.

  ---

  cls Alternative class to use for constructing the return rather than `AChemKit.reactionnet.ReactionNetwork`.

  rng Random number generator to use. If not specifed, one will be generated at random.

These arguments can be a single value, a tuple/list which will be uniformly sampled from, or a dictionary of value/weighting which will be sampled from

For example:

`Uniform(5,3,2,1)` will generate 5 molecules with 3 reactions between them where each reaction has two reactants and one product.

`Uniform(5,3,(1,2), (1,2))` will generate 5 molecules with 3 reactions between them where each reaction has one or two reactants and one or two products.

`Uniform(5,3,((2,1),(1,2)), None)` will generate 5 molecules with 3 reactions between them where each reaction has either two reactants and one product or one reactant and two products.

AChemKit.randomnet.**combinations_with_replacement**(*iterable*, *r*)

3.1.7 `randomnet_test` Module

This is the test harness for `AChemKit.randomnet`.

Tricky to test well because it is non-deterministic.

class AChemKit.randomnet_test.**TestLinear**(*methodName='runTest'*)
> Bases: `unittest.TestCase`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **setUp**()

> **test_directed**()

> **test_maxlengt_dict**()

> **test_maxlengt_tuple**()

> **test_natoms_dict**()

> **test_natoms_int**()

> **test_natoms_tuple**()

> **test_pbreak_tuple**()

> **test_pform_tuple**()

> **test_undirected**()

class AChemKit.randomnet_test.**TestUniform**(*methodName='runTest'*)
> Bases: `unittest.TestCase`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **setUp**()

> **test_nmols_dict**()

> **test_nmols_int**()

> **test_nmols_list**()

> **test_nmols_tuple**()

> **test_nprods_dict**()

> **test_nprods_int**()

test_nprods_list()

test_nprods_none_dict()

test_nprods_none_tuple()

test_nprods_tuple()

test_nreacts_dict()

test_nreacts_int()

test_nreacts_list()

test_nreacts_tuple()

test_rate_dict()

test_rate_float()

test_rate_int()

test_rate_list()

test_rate_tuple()

3.1.8 `reactionnet` Module

Core for all ReactionNetwork classes

Of particular note are the alternative constructors of `ReactionNetwork`, `from_file()`, `from_filename()` and `from_string()`.

class `AChemKit.reactionnet.`**`ReactionNetwork`**(*rates*)

Bases: `object`

A dictionary of reactions where each key is a reaction composed of (reactants, products) and each value is the rate.

`ReactionNetwork` objects are immutable and hashable.

`ReactionNetwork` objects support `__eq__()` and `__ne__()`, but none of the other rich comparison operators (__lt__, __le__, __gt__, __ge__).

Different subclassess could be implemented to generate reaction networks on demand (artificial chemistries, etc) and provide additional functionallity, such as visualization or metrics.

Can be cast to string to get a *.chem* representation.

classmethod `from_file`(*infile*)

Alternative constructor that accepts a `file()` object (or equivalent).

Source must be formatted as a .chem file, see *.chem Format*.

classmethod `from_filename`(*infilename*)

Wrapper around `AChemKit.reactionnet.ReactionNetwork.from_file()` that opens the provided filename as a file to read.

classmethod `from_reactions`(*allreactions*, *mols*, *maxdepth=None*, *maxmols=None*)

Generates a reaction network using a function that returns all possible products for a given set of reactants and an initial set of molecules.

allreactions is a function that takes two reactants as parameters, (typically via *args) and returns a dictionary of possible outcomes and their weighting.

Inline emphasis start-string without end-string.

maxdepth is the number of times all reactions of seen molecules is evaluated. If it is None, then the process repeats until no more novel molecules are created; this may lead to an infinite loop.

classmethod **from_string**(*instr*)
> Wrapper around `AChemKit.reactionnet.ReactionNetwork.from_file()` that uses `StringIO`.

rate(*reactants*, *products*)
> Gets the rate of a particular reaction.

rates()
> Itterate over all reactions through tuples of the form (reactants, products, rate)

classmethod **reaction_to_string**(*reaction*, *rate=1.0*)
> Produces a human-readable string for a particular reaction.

> Mainly used to convert entire reaction network to a string representation, but can also be used for individual reactions if desired.

reactions
> Sorted tuple of all reactions in the network.

seen
> Sorted tuple of all molecular species in the network

3.1.9 `reactionnet_test` Module

This is the test harness for `AChemKit.reactionnet`.

class AChemKit.reactionnet_test.**TestReactionNetwork**(*methodName='runTest'*)
> Bases: `unittest.TestCase`

> This is the main class to test ReactionNetwork class.

> It relies on setUp to generate a ReactionNetwork instance which is then probed by the other functions

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **setUp**()

> **test_equal**()
>> As ReactionNetwork has a custom __eq__ function, it is tested here.

>> Needs to both pass and fail.

> **test_hash**()
>> As ReactionNetwork has a custom __hash__ function, it is tested here.

>> Needs to both pass and fail.

>> Technically, the fail here could be true and still be a hash but it is supposed to usually be wrong so assume that it will be wrong.

> **test_rate**()

> **test_reaction_to_string**()
>> Check that it convert a reaction to a string correctly

> **test_reactions**()
>> Makes sure the reactions that were specified are in reactions. Also checks that they are in sorted order TODO check sorted order between reactions

test_seen()
>   Makes sure the molecules that were specified are in seen. Also checks that they are in sorted order.

test_to_string()
>   Check that it converts to a string correctly

class AChemKit.reactionnet_test.**TestReactionNetwork_from_string**(*methodName='runTest'*)
>   Bases: AChemKit.reactionnet_test.TestReactionNetwork

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()

3.1.10 `reactionnetdot` Module

class AChemKit.reactionnetdot.**ReactionNetworkDot**(**args*, ***kwargs*)
>   Bases: AChemKit.reactionnet.ReactionNetwork

This is a subclass of AChemKit.reactionnet.ReactionNetwork that adds a *.dot* representation. It is in a subclass so that on machines where dot is not installed, the basic class can still be used.

dot
>   Property wrapper around to_dot() to provide attribute-like access.

to_dot(*names=None*, *rates=None*, *shown=()*, *hidden=()*)
>   Return a *.dot* format string constructed using a AChemKit.utils.simpledot.SimpleDot view of this reaction network.

Molecular species are shown as either full names, identifier numbers, or as blank circles. This is determined by the *names* parameter, which is a string indicating which to use:

"full" This will use the full molecule name specified in the .chem file

"id" This will use a shortened identifier of that molecules index in the seen tuple.

"blank" This will not name any molecules, but will put an empty circle instead.

A sub-set of the reaction network can be drawn using the *shown* and *hidden* parameters.

Reactions involving only molecular species on the *hidden* list are not shown. Molecular species on the *hidden* list are not shown unless they are involved in a reaction with molecular species not on the *hidden* list, in which case the molecular species on the *hidden* list is unlabeled and shown as a point.

The *shown* list in the inverse of the hidden list. If it is used, any molecular species not on the *shown* list is treated as being on the *hidden* list. If a molecular species is on both *shown* and *hidden* lists, the *hidden* lists wins.

Catalysts (defined as molecules that are required for but unchanged by a reaction) are indicated with a grey line.

Reversible reactions (defined as a pair of reactions where reactants of one are the products of the other and visa versa) are combined and indicated with double arrows resembling a diamond shape.

3.1.11 `reactionnetdot_test` Module

This is the test harness for AChemKit.reactionnetdot.

class AChemKit.reactionnetdot_test.**TestReactionNetworkDot**(*methodName='runTest'*)
>   Bases: unittest.TestCase

This is the main class to test ReactionNetwork class.

It relies on setUp to generate a ReactionNetwork instance which is then probed by the other functions

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()

test_to_dot_str()
> Test for dot conversion by string representation

> Assumes that the dot representation itself is valid

3.1.12 `sims_gillespie` Module

class AChemKit.sims_gillespie.**Gillespie**(*achem*, *mols*, *rng*, *intervalscaling=100.0*)
> Bases: `object`

> **next_reaction**()

AChemKit.sims_gillespie.**simulate_gillespie**(*achem*, *mols*, *maxtime*, *rng=None*)

AChemKit.sims_gillespie.**simulate_gillespie_iter**(*achem*, *mols*, *maxtime*, *rng=None*)
> Given an Artificial Chemistry, this simulates a series of simple iterative reactions. It returns events as tuples of (time, reactants, products).

> This is an implementation of the Gillespie algorithm. It is a simulation of a well-mixed container.

> Is an iterator to reduce memory consumption. See simulate_gillespie for this function wrapped in a tuple.

3.1.13 `sims_gillespie_test` Module

This is the test harness for `AChemKit.sims_simple`.

class AChemKit.sims_gillespie_test.**TestGillespie**(*methodName='runTest'*)
> Bases: `unittest.TestCase`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **setUp**()

> **test_basic**()

3.1.14 `sims_simple` Module

class AChemKit.sims_simple.**AChemAbstract**
> Bases: `object`

> This is an abstract base class for an AChem that can be used in a simulation or similar. It defines core attribtues and functions that subclasses must implement in order to follow the API correctly.

> It is not required that an AChem inherits this, as Python follows the principle of "duck typing".

> **react**(**reactants*)
> > Given some number of reactants, return the products.

> > The reactant objects should not be changed. Rather, copies should be returned. This is so events and buckets work correctly.

class AChemKit.sims_simple.**AChemReactionNetwork**(*reactionnetwork*)

    Bases: AChemKit.sims_simple.AChemAbstract

    This is an AChem class that uses a Reaction Network as its base.

    The main point of this is to be able to compare different simuatlion approaches to see which ones best reconstruct the original Reaction Network.

    react(*reactants*)

AChemKit.sims_simple.**simulate_itterative**(*achem*, *mols*, *maxtime*, *rng=None*)

AChemKit.sims_simple.**simulate_itterative_iter**(*achem*, *mols*, *maxtime*, *rng=None*)

    Given an Artificial Chemistry, this simulates a series of simple iterative reactions. It returns events as tuples of (time, reactants, products).

    One reaction occurs each second for a number of seconds up to the maximum time specified.

    Works as an iterator to reduce memory consumption. See simulate_itterative for this function wrapped in a tuple.

AChemKit.sims_simple.**simulate_stepwise**(*achem*, *mols*, *maxtime*, *rng=None*)

AChemKit.sims_simple.**simulate_stepwise_iter**(*achem*, *mols*, *maxtime*, *rng=None*)

    Given an Artificial Chemistry, this simulates a series of simple iterative reactions. It returns events as tuples of (time, reactants, products).

    Each molecule will attempt to react once for each second. If there are any leftover molecules, they will get a free pass to the next second.

    Is an iterator to reduce memory consumption. See simulate_stepwise for this function wrapped in a tuple.

AChemKit.sims_simple.**simulate_stepwise_multiprocessing**(*achem*, *mols*, *maxtime*, *rng=None*)

AChemKit.sims_simple.**simulate_stepwise_multiprocessing_iter**(*achem*, *mols*, *maxtime*, *rng=None*)

    Given an Artificial Chemistry, this simulates a series of simple iterative reactions. It returns events as tuples of (time, reactants, products).

    Each molecule will attempt to react once for each second. If there are any leftover molecules, they will get a free pass to the next second.

    Is an iterator to reduce memory consumption. See simulate_stepwise_multiprocessing for this function wrapped in a tuple.

    Uses multiprocessing to parallelize reactions. This does not work that well because the individual reactions are not that complicated.

    Will only work where achem is a class, not an instance of a class.

3.1.15 `sims_simple_test` Module

This is the test harness for AChemKit.sims_simple.

class AChemKit.sims_simple_test.**TestItterative**(*methodName='runTest'*)

    Bases: unittest.TestCase

    Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

    setUp()

    test_basic()

class AChemKit.sims_simple_test.**TestStepwise**(*methodName='runTest'*)
    Bases: AChemKit.sims_simple_test.TestItterative

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

test_basic()

3.1.16 Subpackages

tools Package

tools Package

AChemKit comes with a number of complete command-line driven tools for working with various file types.

These are intended to be used in automated pipelines, such as Makefiles.

These can be used as stand-alone programs, or examined as usage examples for the AChemKit *API*.

chem_linear Module

This is a command-line tool for generating *.chem* files using the Linear() random reaction network algorithm.

usage: chem_linear.py [options]

Options:

> **-h, --help**   show this help message and exit
>
> **-o OUTFILE, --outfile OUTFILE**   write to OUTFILE in .chem format (if ommited, use stdout)
>
> **-a NATOMS, --natoms NATOMS**   number of atoms
>
> **-l LENGTH, --length LENGTH**   maximum number of atoms per molecule
>
> **-f PFORM, --pform PFORM**   probability of bonding
>
> **-b PBREAK, --pbreak PBREAK**   probability of breaking
>
> **-u, --undirected**   undirected molecules
>
> **-d, --directed**   directed molecules
>
> **-t RATES, --rates RATES**   possible reaction rates
>
> **-s SEED, --seed SEED**   pseudo-random seed

chem_pp Module

A pretty-printer and syntax checker for *.chem* files

For information in *.chem* files see *.chem Format*.

Usage: chem_pp.py [options]

Options:

> **-h, --help**   show a help message and exit

-i INFILE, --infile=INFILE   read from INFILE (if ommited, use stdin)

-o OUTFILE, --outfile=OUTFILE   write to OUTFILE in .chem format (if ommited, use stdout)

`chem_to_dot` Module

Produces a *.dot* output from a provided *.chem* format input

The output files in *.dot* format are suitable for future processing using Graphviz tools. In particular, they are constructed using `AChemKit.utils.simpledot.SimpleDot`.

For information in *.chem* files see *.chem Format*.

Usage: chem_to_dot.py [options]

Options:

-h, --help   Show a help message and exit

-i INFILE, --infile=INFILE   Read from INFILE (if ommited, use stdin)

-o OUTFILE, --outfile=OUTFILE   Write to OUTFILE in .chem format (if ommited, use stdout)

-n NAMES, --names   Style of molecular species naming. One of 'full', 'id', 'blank'

`chem_to_pdf` Module

Produces a *.pdf* output from a provided *.chem* format input

The output files in *.pdf* format produced using Graphviz tools. In particular, they are constructed using `AChemKit.utils.simpledot.SimpleDot`.

For information in *.chem* files see *.chem Format*.

Usage: chem_to_pdf.py [options]

Options:

-h, --help   show a help message and exit

-i INFILE, --infile=INFILE   read from INFILE (if ommited, use stdin)

-o OUTFILE, --outfile=OUTFILE   write to OUTFILE in .pdf format (if ommited, use stdout)

-l LAYOUT, --layout=LAYOUT   Graphviz layout to use (if ommited, use dot)

`chem_uniform` Module

This is a command-line tool for generating *.chem* files using the `Uniform()` random reaction network algorithm.

`log_to_chem` Module

Produces a *.chem* output from a provided *.log* format input

For information in *.chem* files see *.chem Format*.

Usage: chem_to_dot.py [options]

Options:

-h, --help	Show a help message and exit
-i INFILE, --infile=INFILE	Read from INFILE (if ommited, use stdin)
-o OUTFILE, --outfile=OUTFILE	Write to OUTFILE in .chem format (if ommited, use stdout)
-n NAMES, --names	Style of molecular species naming. One of 'full', 'id', 'blank'

utils Package

`utils` Package

Various utility functions and classes.

`bag` Module

A collection of classes providing containers. These use the abstract base classes (ABCs) from `collections` module to satisfy isinstance() criteria for API provision.

class AChemKit.utils.bag.**Bag**(*iterable*)

> Bases: AChemKit.utils.bag.FrozenBag, _abcoll.MutableSet

> A Bag is like a set, but can contain duplicates.

> Also, a Bag is like a list, but is always ordered.

> **add**(*item*)

> **discard**(*item*)

class AChemKit.utils.bag.**FrozenBag**(*iterable*)

> Bases: _abcoll.Set

> A Bag is like a set, but can contain duplicates.

> Also, a Bag is like a list, but is always ordered.

> Note: objects must be both hashable and sortable. By default, python objects are sorted by id(), but this is not consistent. As there is no easy way to test this, if you get wierd results this may be the cause.

> **count**(*item*)

class AChemKit.utils.bag.**OrderedBag**(*iterable*)

> Bases: AChemKit.utils.bag.OrderedFrozenBag, _abcoll.MutableSet

> Like a Bag, but iterating will keep the order things were put in. New items are added to the end of the Ordered-Bag - if you need anything else you can convert it to a tuple or list and make a new bag.

> Comparisons are still as for a Bag so OrderedBag([1,2,1]) == OrderedBag([2,1,1]) will return True.

> **add**(*item*)

> **discard**(*item*)

class AChemKit.utils.bag.**OrderedFrozenBag**(*iterable*)

> Bases: _abcoll.Set

> Like a FrozenBag, but iterating will keep the order things were put in.

Comparisons are still as for a FrozenBag so OrderedFrozenBag([1,2,1]) == OrderedFrozenBag([2,1,1]) will return True.

count (*item*)

index (*item*)

class AChemKit.utils.bag.**OrderedFrozenBagCache**
> Bases: `object`

bag_test Module

This is the test harness for `AChemKit.utils.bag`.

class AChemKit.utils.bag_test.**TestBag** (*methodName='runTest'*)
> Bases: `AChemKit.utils.bag_test.TestFrozenBag`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **cls**
>> alias of `Bag`

> **setUp** ()

> **test_add** ()

> **test_discard** ()

> **test_hash** ()

class AChemKit.utils.bag_test.**TestFrozenBag** (*methodName='runTest'*)
> Bases: `unittest.TestCase`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **cls**
>> alias of `FrozenBag`

> **setUp** ()

> **test_eq** ()

> **test_hash** ()

> **test_iter** ()

> **test_len** ()

> **test_pickle** ()

> **test_repr** ()

> **test_str** ()

class AChemKit.utils.bag_test.**TestOrderedBag** (*methodName='runTest'*)
> Bases: `AChemKit.utils.bag_test.TestOrderedFrozenBag`, `AChemKit.utils.bag_test.TestBag`

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **cls**
>> alias of `OrderedBag`

setUp()

test_hash()

class AChemKit.utils.bag_test.**TestOrderedFrozenBag**(*methodName='runTest'*)
Bases: AChemKit.utils.bag_test.TestFrozenBag

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

cls
alias of OrderedFrozenBag

setUp()

test_iter()

simpledot **Module**

A library to manage *.dot* files

There are existing python libraries for this, but they do not always do it correctly (e.g. PyDot does not respect order within a .dot file).

class AChemKit.utils.simpledot.**SimpleDot**(*name='G'*, *digraph=True*, *strict=False*, *cluster=False*)
Bases: UserDict.DictMixin

Repsents a graphviz *.dot* file (also known as *.gv*).

Uses a dictionary-like interface via the :class:~'UserDict.DictMixin' class.

To produce *.dot* output, cast to string e.g. *str(mysimpledot)*

Contains three types of graph objects:

Nodes Named by strings and are dictionaries of attributes.

Edges Named by tuples of length 2 of strings and are are dictionaries of attributes. Nodes will be implicitly created by graphviz if they do not exist, and therefore do not need to be explicitly created.

Subgraphs Named by strings and are instances of this class. Some graphviz layout engines (e.g. *neato*) will flatten subgraphs.

These can be set and accessed by standard slice notation (e.g. *dot[nodename] = {}*).

Node names and subgraph names are unique but multiple parallel edges are permitted. However, slice notation (e.g. *dot[(from, to)]*) cannot cope with this. Therefore, when there are multiple parallel edges, accessing any of them returns a tuple of all their attributes as dictionaries. To create multiple parallel edges you must use the add() method

Names of nodes and subgraphs, attribute keys, and attribute values should all be graphviz compatible. Some attempt to wrap string attribute values in quotes will be made so that the use of plain python strings is accepted by graphviz.

Very little checking and enforcement is performed. This means that you can use them in ways not originally intended; for example, you can se attributes that no graphviz programme will recognize. But, it also means you can break it by doing odd things to them.

add(*key*, *value=None*)
Guaranteed to add the passed key/value to self, even if it is a duplicate.

---

get (*key*)
> Returns a tuple of all things matching that key.

> Designed for multiple edges, but will also work with single edges, nodes, or subgraphs. This provides a unified interface.

keys ()
> Return a tuple of the keys.

plot (*output='pdf'*, *prog='dot'*, *args=()*)
> Calls the specified drawing program to turn this into an image.

> Use args to pass extra arguments, particularly −o to specify an output filename.

> Follows the same format as subprocess calls.

utils **Module**

Various small functions that can get lumped together into this module.

AChemKit.utils.utils.**get_sample** (*distribution*, *rng=None*)
> Samples a provided distribution at random.

> Distribution can be a single number (int or float), always returns the same value

> Distribution can be a sequence (list or tuple) which will be uniformly sampled from Duplicates can be used to adjust frequencies.

> Distribution can be a mapping (dict) where the keys are things to be returned and values are the relative weightings.

AChemKit.utils.utils.**long_subseq** (*data*)
> Given some sequences — strings, tuples, lists, etc — return the longest subsequence common to all sequences.

AChemKit.utils.utils.**memory_free** ()
> Returns the amount of memory free, in megabytes

AChemKit.utils.utils.**myp** (*p*, *f*, **args*)

AChemKit.utils.utils.**unpck** (*combo*)

utils_test **Module**

This is the test harness for AChemKit.utils.utils.

class AChemKit.utils.utils_test.**TestGetSample** (*methodName='runTest'*)
> Bases: unittest.TestCase

> Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

> **test_dicts** ()

> **test_ints** ()

> **test_lists** ()

TO-DOS

As with any project, there are a number of things that could be done to improve it.

4.1 Features

Faster Any sort of speed up would be nice. Currently, Pysco will by used if avaliable. Shedskin and PyPy are options for futher improvments.

As well as this, algorithmic speed ups and optimizations would be good.

Chemical Organization Theory prob catalysis RAF sets size of F to size of set overlapping F for sample RA set

4.2 Bugs

PYTHON MODULE INDEX

a