Stochastic Simulation Algorithms in Python Documentation

Release 0.8.2

Dileep Kishore, Srikiran Chandrasekaran

May 04, 2019

CONTENTS

CHAPTER

ONE

PYSSA : PYTHON PACKAGE FOR STOCHASTIC SIMULATIONS

1.1 Introduction

`pyssa` is a Python package for stochastic simulations. It offers a simple api to define models, perform stochastic simulations on them and visualize the results in a convenient manner.

1.2 Install

Install with `pip`:

```
$ pip install pyssa
```

1.3 Documentation

- General: https://pyssa.readthedocs.io.

1.4 Usage

```
from pyssa.simulation import Simulation

 V_r = np.array([[1, 0], [0, 1], [0, 0]])  # Reactant matrix
 V_p = np.array([[0, 0], [1, 0], [0, 1]])  # Product matrix
 X0 = np.array([100, 0, 0])  # Initial state
 k = np.array([1.0, 1.0])  # Rate constants
 sim = Simulation(V_r, V_p, X0, k)  # Declare the simulation object
 # Run the simulation
 sim.simulate(max_t=150, max_iter=1000, chem_flag=True, n_rep=10)
```

You can change the algorithm used to perform the simulation by changing the `algorithm` parameter

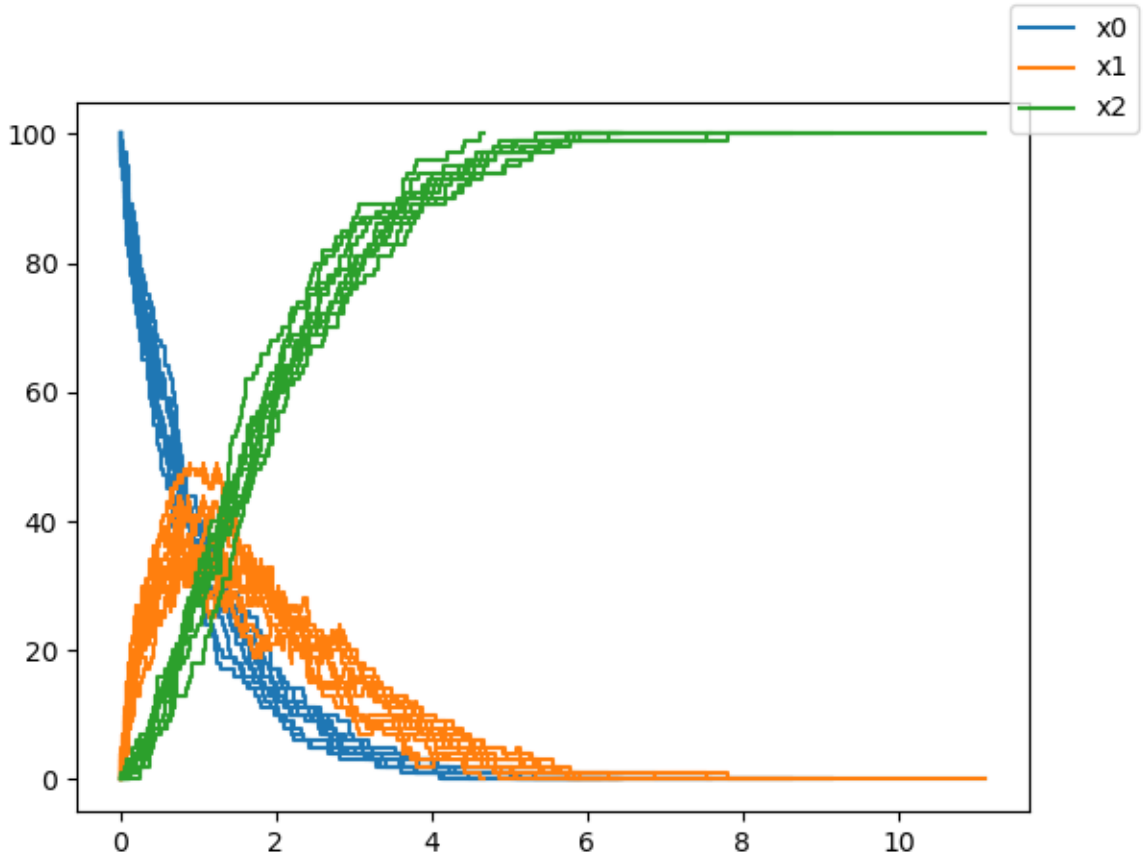
```
sim.simulate(max_t=150, max_iter=1000, chem_flag=True, n_rep=10, algorithm="tau_
↪adaptive")
```

You can run the simulations on multiple cores by specifying the `n_procs` parameter

```
sim.simulate(max_t=150, max_iter=1000, chem_flag=True, n_rep=10, n_procs=4)
```

1.4.1 Plotting

```
sim.plot()
```



Plot of species A, B and C

1.4.2 Accessing the results

```
results = sim.results
```

1.5 Benchmarks

We chose `numba` after extensive testing and benchmarking against `python` and `cython` implementations.

Name (time in ms)	Min	Max	Mean	Std Dev	Median	IQR	Outliers	OPS	Rounds	Iterations
test_numba_benchmark	314.1758 (1.0)	342.9915 (1.0)	322.9318 (1.0)	11.4590 (1.0)	318.7983 (1.0)	9.1533 (1.0)	1;1	3.0966 (1.0)	5	1
test_cy_benchmark	17,345.7698 (55.21)	19,628.3931 (57.23)	18,255.3931 (56.53)	862.4711 (75.27)	18,148.9358 (56.93)	1,030.3676 (112.57)	2;0	0.0548 (0.02)	5	1
test_py_benchmark	27,366.3681 (87.11)	28,417.8333 (82.85)	27,782.2482 (86.03)	387.2758 (33.80)	27,728.4224 (86.98)	347.3891 (37.95)	2;0	0.0360 (0.01)	5	1

1.6 License

1.7 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install Stochastic Simulation Algorithms in Python, run this command in your terminal:

```
$ pip install pyssa
```

This is the preferred method to install Stochastic Simulation Algorithms in Python, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Stochastic Simulation Algorithms in Python can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dileep-kishore/pyssa
```

Or download the tarball:

```
$ curl  -OL https://github.com/dileep-kishore/pyssa/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

TUTORIAL

3.1 Model Building

Consider a simple system of chemical reactions given by:

$$A \xrightarrow{k_1} B$$
$$B \xrightarrow{k_2} C$$

Suppose $k_1 = 1$, k_2 1 and there are initiall 100 units of A. Then we have the following variable definitions

```
k1, k2 = 1.0, 1.0
A0, B0, C0 = 100, 0, 0
```

Then to build the model we have the following variable definitions:

```
import numpy as np
V_r = np.array([[1, 0], [0, 1], [0, 0]])
V_p = np.array([[0, 0], [1, 0], [0, 1]])
X0 = np.array([A0, B0, C0])
k = np.array([k1, k2])
```

3.2 Running Simulations

Suppose we want to run 10 runs of the system for earlier of 1000 time steps / 150 time units each, we have

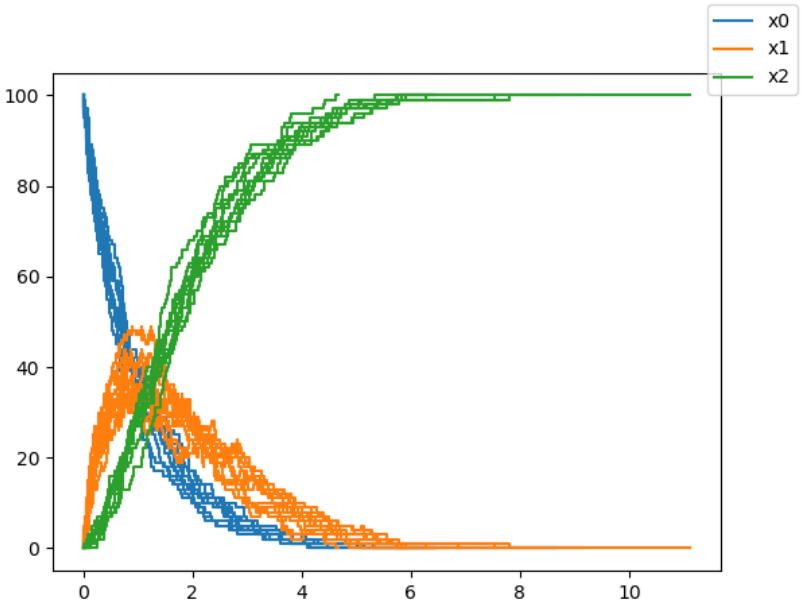
```
from pyssa.simulation import Simulation

sim = Simulation(V_r, V_p, X0, k)
sim.simulate(max_t=150, max_iter=1000, chem_flag=True, n_rep=10)
```

Note that the `chem_flag` is set to `True` since we are dealing with a chemical system.

3.3 Plotting

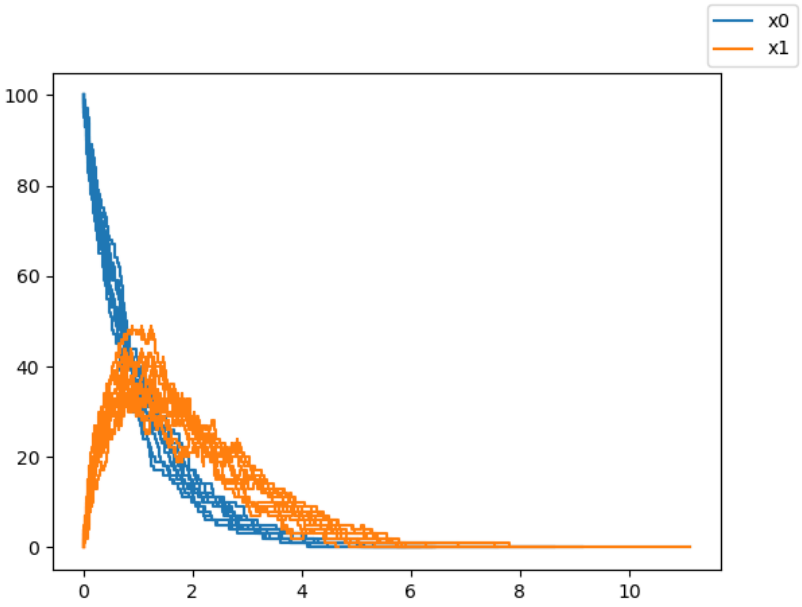
To plot the results on the screen, we simply have

```
sim.plot()
```



To plot only A and B, we use the species indices (`[0,1]`)

```
sim.plot(plot_indices = [0, 1])
```



To not display the plot on the screen and retrieve the figure and axis objects, we have

```
fig, ax = sim.plot(disp = False)
```

3.4 Accessing the results

The results of the simulation can be retrieved by accessing the `Results` object as

```
results = sim.results
```

```
<Results n_rep=10 algorithm=direct seed=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]>
```

The `Results` object provides abstractions for easy retrieval and iteration over the simulation results. For example you can iterate over every run of the simulation using

```
for x, t, status in results:
    pass
```

You can access the results of the `n` th run by

```
nth_result = results[n]
```

You can also access the final states of all the simulation runs by

```
final_times, final_states = results.final
```

```
#final_times
array([ 7.59679567,  6.370443  ,  8.62018373,  6.44826162,  6.42278186,
    4.66472231,  6.15595516,  5.87319502,  9.13955542, 11.12529717])
#final_states
array([[  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.],
   [  0.,   0., 100.]])
```

3.5 Algorithms

The `Simulation` class currently supports the following algorithms:

1. Direct
2. Tau leaping

You can change the algorithm used to perform a simulation using the `simulation` flag

```
sim.simulate(max_t=150, max_iter=1000, chem_flag=True, n_rep=10, algorithm="tau_
↪leaping")
```

CHAPTER

FOUR

PYSSA

4.1 pyssa package

4.1.1 Subpackages

pyssa.benchmark package

Submodules

pyssa.benchmark.benchmark module

Module for benchmarking the performance of various implementations of the Gillespie algorithm

pyssa.benchmark.benchmark.**benchmark**(*setup: str*, *stmt: str*, *number: int = 1000000*) → float

Function the runs the benchmark

Parameters

setup [str] The string containing the initial setup for the profiling

stmt [str] The string containing the code to be profiled

number [int] The number of iterations to be performed

***args** Arguments to be forwarded to the function

****kwargs** Keyword arguments to be forwarded to the function

Returns

float The time taken to run the *stmt*

Module contents

4.1.2 Submodules

4.1.3 pyssa.direct module

4.1.4 pyssa.results module

Module that defines the *Results* class

class pyssa.results.**Results**(*t_list: List[numpy.ndarray], x_list: List[numpy.ndarray], status_list: List[int], algorithm: str, seed: List[int], **kwargs*)

Bases: collections.abc.Collection

A class that stores simulation results and provides methods to access them

Parameters

t_list [List[float]]

x_list [List[np.ndarray]]

status_list [List[int]]

algorithm [str]

seed: List[int]

Attributes

final Returns the final times and states of the system in the simulations

Methods

get_state(t)	Returns the states of the system at time point t.

final

Returns the final times and states of the system in the simulations

Returns

Tuple[np.ndarray, np.ndarray] The final times and states of the sytem

get_state(*t: float*) → List[numpy.ndarray]

Returns the states of the system at time point t.

Parameters

t [float] Time point at which states are wanted.

Returns

List[np.ndarray] The states of the system at *t* for all repetitions.

4.1.5 pyssa.simulation module

The main class for running stochastic simulation

class pyssa.simulation.**Simulation**(*react_stoic: numpy.ndarray, prod_stoic: numpy.ndarray, init_state: numpy.ndarray, k_det: numpy.ndarray, chem_flag: bool = False, volume: float = 1.0*)

Bases: object

A main class for running simulations.

Parameters

react_stoic [(ns, nr) ndarray] A 2D array of the stoichiometric coefficients of the reactants. Reactions are columns and species are rows.

prod_stoic [(ns, nr) ndarray] A 2D array of the stoichiometric coefficients of the products. Reactions are columns and species are rows.

init_state [(ns,) ndarray] A 1D array representing the initial state of the system.

k_det [(nr,) ndarray] A 1D array representing the deterministic rate constants of the system.

volume [float, optional] The volume of the reactor vessel which is important for second and higher order reactions. Defaults to 1 arbitrary units.

chem_flag [bool, optional] If True, divide by Na while calculating stochastic rate constants. Defaults to False.

Raises

ValueError If supplied with order > 3.

Examples

```
>>> V_r = np.array([[1,0],[0,1],[0,0]])
>>> V_p = np.array([[0,0],[1,0],[0,1]])
>>> X0 = np.array([10,0,0])
>>> k = np.array([1,1])
>>> sim = Simulation(V_r, V_p, X0, k)
>>> sim.simulate(max_t=10, max_iter=100, n_rep=n_runs)
```

Attributes

results [Results] The `Results` instance of the simulation

Methods

plot(plot_indices, disp, names)	Plot the simulation
simulate(max_t, max_iter, volume, seed, ...)	Run the simulation

plot (*plot_indices: list = None*, *disp: bool = True*, *names: list = None*)
Plot the simulation

Parameters

plot_indices [list, optional] The indices of the species to be plotted. The default is *[i for i in range(self._ns)]* plots all species.

disp [bool, optional] If *True*, the plot is displayed. The default shows the plot.

names [list, optional] The names of the species to be plotted. The default is *xi* for species *i*.

Returns

fig [class 'matplotlib.figure.Figure'] Figure object of the generated plot.

ax [class 'matplotlib.axes._subplots.AxesSubplot] Axis objected of the generated plot.

results
The `Results` instance of the simulation

Returns

Optional[Results]

simulate (*max_t: float = 10.0*, *max_iter: int = 1000*, *volume: float = 1.0*, *seed: Optional[List[int]] = None*, *n_rep: int = 1*, *n_procs: int = 1*, *algorithm: str = 'direct'*, *debug: bool = False*, ***kwargs*)

Run the simulation

Parameters

max_t [float, optional] The end time of the simulation The default is 10.0

max_iter [int, optional] The maximum number of iterations of the simulation loop The default is 1000 iterations

volume [float, optional] The volume of the system The default value is 1.0

seed [List[int], optional] The list of seeds for the simulations The length of this list should be equal to *n_rep* The default value is None

n_rep [int, optional] The number of repetitions of the simulation required The default value is 1

n_procs [int, optional] The number of cpu cores to use for the simulation The default value is 1

algorithm [str, optional] The algorithm to be used to run the simulation The default value is "direct"

Returns

t [float] End time of the simulation.

Xt [ndarray] System state at time *t* and initial.

status [int] Indicates the status of the simulation at exit. 1 : Succesful completion, terminated when *max_iter* iterations reached. 2 : Succesful completion, terminated when *max_t* crossed. 3 : Succesful completion, terminated when all species went extinct. -1 : Failure, order greater than 3 detected. -2 : Failure, propensity zero without extinction.

pyssa.simulation.**wrapper**(*x*, *func*)

4.1.6 pyssa.tau_adaptive module

4.1.7 pyssa.tau_leaping module

4.1.8 pyssa.utils module

Module containing some utility functions

pyssa.utils.**get_kstoc**

Compute k_stoc from k_det.

Return a vector of the stochastic rate constants (k_stoc) determined from the deterministic rate constants (k_det).

Parameters

react_stoic [(ns, nr) ndarray] A 2D array of the stoichiometric coefficients of the reactants. Reactions are columns and species are rows.

k_det [(nr,) ndarray] A 1D array representing the deterministic rate constants of the system.

volume [float] The volume of the reactor vessel which is important for second and higher order reactions

chem_flag [bool] If True, divide by Na while calculating stochastic rate constants.

Returns

k_stoc [(nr,) ndarray] A 1D array representing the stochastic rate constants of the system.

References

1. Gillespie, D.T., 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. J. Comput. Phys. 22, 403–434. doi:10.1016/0021-9991(76)90041-3.

pyssa.utils.**roulette_selection**

Perform roulette selection on the list of propensities.

Return the index of the selected reaction (*choice*) by performing Roulette selection on the given list of reaction propensities.

Parameters

prop [array_like] A 1D array of the propensities of the reactions.

Returns

choice [int] Index of the chosen reaction.

status [int] Status of the simulation as described in *direct*.

4.1.9 Module contents

Top-level package for Stochastic Simulation Algorithms in Python.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/dileep-kishore/pyssa/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Stochastic Simulation Algorithms in Python could always use more documentation, whether as part of the official Stochastic Simulation Algorithms in Python docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dileep-kishore/pyssa/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pyssa* for local development.

1. Fork the *pyssa* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyssa.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pyssa
$ cd pyssa/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyssa tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/dileep-kishore/pyssa/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_pyssa
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Dileep Kishore <k.dileep1994@gmail.com>
- Srikiran Chandrasekaran <srikiranc@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.8.2 (2019-04-20)

7.1.1 Fixed

- Initialize `algorithms` submodule with `__init__.py`
- Update `setup.py` to allow submodule detection

7.2 0.8.0 (2019-04-13)

7.2.1 Added

- `Results.get_states` method - returns state at time `t`
- Accuracy tests for all algorithms
- Additional consistency checks for `X0` and `k_det`

7.2.2 Changed

- Refactor algorithms into sub module `algorithms`
- Refactor algorithm independent tests

7.2.3 Fixed

- Indexing issue in propensity calculation in `direct` algorithm
- Indexing issue in propensity calculation in `tau_leaping` algorithm
- Address edge case X->2X in `tau_adaptive` algorithm

7.3 0.7.1 (2019-03-23)

7.3.1 Changed

- Refactor `tau_adaptive`

- Rename `direct_naive` to `direct`

7.3.2 Fixed

- SSA part of `tau_adaptive`
- Bug in linux compatibility of `tau_adaptive`

7.4 0.7.0 (2019-02-02)

7.4.1 Added

- Support for the `tau_adaptive` algorithm
- Support for multiprocessing

7.4.2 Fixed

- Transpose stoichiometric matrix
- Update references in docstrings

7.4.3 Changed

- Use `TINY` and `HIGH` for status estimation
- Use `np.int64` and `np.float64` explicitly

7.4.4 Chore

- Update dependencies
- Add azure pipelines for testing on Windows

7.5 0.6.0 (2018-12-16)

7.5.1 Added

- Updated `direct_naive` docstring
- Support for the `tau_leaping` algorithm
- Species name support for plotting

7.5.2 Fixed

- Check for sum propensities uses threshold instead of equality
- Add check for type of `max_iter`

7.5.3 Changed

- Update `roulette_selection` to use *np.searchsorted*
- Minor changes to `numpy` style usage

7.5.4 Chore

- Add `codecov`
- Travis pypi autodepolyment
- Parameterize tests with algorithm name
- Add details about `tau_leaping` to docs and README

7.6 0.5.4 (2018-12-02)

7.6.1 Added

- badge to readme

7.7 0.5.3 (2018-12-02)

7.7.1 Added

- plot to pypi

7.7.2 Changed

- fix bumpversion/black issue
- remove history from package long_description

7.8 0.5.0 (2018-12-01)

First public release!!

7.8.1 Added

- testpypi deployment
- pyup security checking
- readthedocs deployment
- Tutorials and documentation
- Plotting functionality through `Simulation.plot`

7.8.2 Changed

- `Simulation.results` is now a property
- Updated tests to support the new api changes

7.8.3 Chore

- Updated the README

7.9 0.4.0 (2018-11-23)

7.9.1 Added

- `Simulation` class - main class for running simulations
- `Results` class - for storing and acessing simulation results
- `Simulation.simulate` function that returns an instance of the `Results` class

7.9.2 Changed

- Refactor `get_kstoc` and `roulette_selection` into `utils.py`
- Refactor `direct_naive` into `direct_naive.py`
- Delete `pyssa.py` and replace with `Simulation` class

7.9.3 Chore

- Add license and code-style badges
- Use `black` for code-formatting

7.10 0.2.0 (2018-11-10)

7.10.1 Added

- Naive implementation of the Gillepsie algorithm in `numba`
- Tests - sanity checks, bifurcation and long running simulation
- CI on `travis`

7.11 0.1.0 (2018-08-08)

- First commit

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX