Biosystempy

Generated by Doxygen 1.8.11

Contents

1 Python BioSystem Framework

The aim of the project was to create simple, easy accessible and editable framework for synthetic biology research, based on freely available libraries and programming languages.

The tool was inspired by MIT university course „20.305x Principles of Synthetic Biology“ provided tool „Part-compositor framework“ which is based on „MatLab“ framework.

Python programming language and its non-standard libraries: Sympy, Numpy, and Scipy were used to implement the goal, due to the similarity of the MatLab functionality required. The created tool can simulate concentrations of substances in time using chemical reaction differential equitations with the specified initial concentration conditions of substances.

The implementation is available on the public github webpage: https://github.com/eglepl/pybiosystem_framwork

Requirements

- Linux (might work with other OS)
- Python 2.7
- Python libraries:
  - SymPy v1.0
  - NumPy v1.11.1
  - SciPy v0.18.1
  - Matplotlib v1.5.3 (optional for data plotting)

Documentation

See the docs/html/index.php file for documentation reference and examples.

The PDF version can be found in docs/latex folder.

Usage

```
from Biosystem import *
from Part import *
from Rate import *
from Pulse import *
import matplotlib.pyplot as plt # optional for plotting

sys = BioSystem()
sys.addConstant('k', 0.05)
dAdt = sys.addCompositor('A', 10)
dBdt = sys.addCompositor('B', 0)
dEdt = sys.addCompositor('E', 1)
reaction  = Part(
'A + E -k> B + E',
[dAdt, dBdt, dEdt],
[Rate('-k * A * E'), Rate('k * A * E'), Rate('0')])
sys.addPart(reaction)
T = None
Y = None
(T, Y) = sys.run([0, 25])

# Plot the simulation data (optional if You want to plot data)
plt.figure()
plt.plot(T, Y[:, sys.compositorIndex('A')], label="A")
plt.plot(T, Y[:, sys.compositorIndex('B')], label="B")
plt.plot(T, Y[:, sys.compositorIndex('E')], label="E")
plt.legend()
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.show()
```

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 Class Documentation

3.1 Biosystem.BioSystem Class Reference

Biological system to simulate.

Public Member Functions

- def __init__ (self)

  The constructor.
- def addCompositor (self, compositor_or_name, init_value=None)

  Create or add a compositor to the system.
- def compositorIndex (self, name)

  Get compositor index in the `compositors` *with name* `name`*.*
- def addPart (self, new_part)

  Add a part to the system.
- def addConstant (self, constant_or_name, init_value)

  Create or add a Constant to the system.
- def determine_rates (self)

  Determine rates of all compositors unless already determined.
- def reset_rates (self)

  Reset all Compositor rates to '0'.
- def changeConstantValue (self, name, value)

  Set Constant value by Constant name.
- def changeInitialValue (self, name, value)

  Set Compositor value and initial value by a Compositor name.

- def reset_state_variables (self)

  Reset all Compositor values to its initial value.
- def run (self, tspan)

  Run a simulation of the Biosystem.
- def sys_ode (self, y, t)

  Ordinary diferential equatation of the system.
- def run_pulses (self, pulse_series)

  Run simulation given pulse list.
- def time_to_index (ignore, T, t)

  Find the index in T (time point) list that gives a value just before t or exact t.
- def interpolate_traces (ignore, iX1, iY1, iX2, iY2)

  Given two (x, y) traces, interpolate the less dense one to have values for each x-value in the denser trace.

Public Attributes

- parts

  Parts in a Biosystem.
- compositors

  Compositors involving chemical reactions.
- constants

  Constants in a Biosystem.
- symbols

  List of all the Compositor symbols in ths BioSystem by initializing symbols with t we allow t to be a variable of time that's not a Compositor or Constant.
- map_constants

  A mapping between constant name and its index in `constants` list.
- map_compositors

  A mapping between compositor name and its index in `constants` list.
- rates_determined

  Flag if function determine_rates was called.

3.1.1 Detailed Description

Biological system to simulate.

In order to analyze a biological system you create a BioSystem object.

Biological system might need some constants (Const) and compositors (Compositor). Compositors are the total rates of change of substance state variables.

Then parts (Part) are declared and added to a system. Part is a process in a system (reaction) that affect some state variables by changing their value according to a rate law (Rate).

Then a simulation can be started.

Example:

```
from Biosystem import *
from Part import *
from Rate import *
from Pulse import *
import matplotlib.pyplot as plt

sys = BioSystem()
sys.addConstant('k', 0.05)
dAdt = sys.addCompositor('A', 10)
dBdt = sys.addCompositor('B', 0)
dEdt = sys.addCompositor('E', 1)
reaction  = Part(
'A + E -k> B + E',
[dAdt, dBdt, dEdt],
[Rate('-k * A * E'), Rate('k * A * E'), Rate('0')])
sys.addPart(reaction)
T = None
Y = None
(T, Y) = sys.run([0, 25])

# Plot the simulation data
plt.figure()
plt.plot(T, Y[:, sys.compositorIndex('A')], label="A")
plt.plot(T, Y[:, sys.compositorIndex('B')], label="B")
plt.plot(T, Y[:, sys.compositorIndex('E')], label="E")
plt.legend()
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.show()
```

Author

Eglė Plėštytė

Date

2017-06-15

3.1.2 Constructor & Destructor Documentation

3.1.2.1 def Biosystem.BioSystem.__init__ (*self*)

The constructor.

Parameters

self	The object pointer.
compositors	Compositors involving chemical reactions.
constants	Constants of a Biosystem.
symbols	A Symbol with a `name` representing a substance.

3.1.3 Member Function Documentation

3.1.3.1 def Biosystem.BioSystem.addCompositor (*self, compositor_or_name, init_value* = `None`)

Create or add a compositor to the system.

Parameters

self	The object pointer.

Parameters

compositor_or_name	Compositor object to add if `init_value` == None, else Compositor name to add with an `init_value`.
init_value	Initial value of a Compositor or None.

Returns

The Compositor object that was added.

3.1.3.2 def Biosystem.BioSystem.addConstant (*self, constant_or_name, init_value*)

Create or add a Constant to the system.

Parameters

self	The object pointer.
constant_or_name	Constant object to add if `init_value` == None, else Constant name to add with an `init_value`.
init_value	Initial value of a Constant or None.

Returns

The Constant object that was added.

3.1.3.3 def Biosystem.BioSystem.addPart (*self, new_part*)

Add a part to the system.

Parameters

self	The object pointer.
new_part	Part object to add.

Returns

Current system object pointer.

3.1.3.4 def Biosystem.BioSystem.changeConstantValue (*self, name, value*)

Set Constant value by Constant name.

Parameters

self	The object pointer.
name	The name of existing Constant.
value	New value of the Constant.

Returns

None.

3.1.3.5 def Biosystem.BioSystem.changeInitialValue (*self*, *name*, *value*)

Set Compositor value and initial value by a Compositor name.

Parameters

self	The object pointer.
name	The name of existing Compositor.
value	New value of the Compositor `initial_value` and Compositor `value`.

Returns

None.

3.1.3.6 def Biosystem.BioSystem.compositorIndex (*self*, *name*)

Get compositor index in the `compositors` with name `name`.

Parameters

self	The object pointer.
name	Existing compositor name.

Returns

the index of a compositor in the system.

3.1.3.7 def Biosystem.BioSystem.determine_rates (*self*)

Determine rates of all compositors unless already determined.

Parameters

self	The object pointer.

Returns

None.

3.1.3.8 def Biosystem.BioSystem.interpolate_traces (*ignore*, *iX1*, *iY1*, *iX2*, *iY2*)

Given two (x, y) traces, interpolate the less dense one to have values for each x-value in the denser trace.

iX1, iY1 form one trace; iX2, iY2 another. The "denser" trace (more datapoints) is used as the basis. Suppose the first is the denser trace. Then for each value of iX1, we find a linear fit of the second trace at that value using the two closest values of iX2. iX1, iX2 are assumed to be ordered and to both start at the same value. Assume iY1, iY2 are columns, iX1, iX2 are rows.

Parameters

ignore	Ignored argument.
iX1	List of X values in first trace.
iY1	List of Y values in first trace.
iX2	List of X values in second trace.
iY2	List of Y values in second trace.

Returns

Tuple matched and interpolated traces.

3.1.3.9 def Biosystem.BioSystem.reset_rates (*self*)

Reset all Compositor rates to '0'.

Parameters

self	The object pointer.

Returns

None.

3.1.3.10 def Biosystem.BioSystem.reset_state_variables (*self*)

Reset all Compositor values to its initial value.

Parameters

self	The object pointer.

Returns

None.

3.1.3.11 def Biosystem.BioSystem.run (*self, tspan*)

Run a simulation of the Biosystem.

Parameters

self	The object pointer.
tspan	Time interval to simulate, for example [t0, t1].

Returns

Tuple (T, Y), where T - time point list, Y - matrix consisting of Compositor values at a time points.

3.1.3.12 def Biosystem.BioSystem.run_pulses (*self*, *pulse_series*)

Run simulation given pulse list.

Last pulse is not simulated. Each pulse defines time, Compositor, Compositor value to set at provided time.

When the Pulse time comes Pulse defined Compositor is set to value provided. Normal simulation is carried on till next Pulse. While there is next Pulse - action repeats.

First Pulse should start at t = 0 time. Each next Pulse time should be greater than previous. Last Pulse is not simulated - it is a stop time.

Example:

```
from Biosystem import *
from Part import *
from Rate import *
from Pulse import *
import matplotlib.pyplot as plt

sys = BioSystem()
sys.addConstant('k', 0.05)
dAdt = sys.addCompositor('A', 10)
dBdt = sys.addCompositor('B', 0)
dEdt = sys.addCompositor('E', 1)
reaction  = Part(
'A + E -k> B + E',
[dAdt, dBdt, dEdt],
[Rate('-k * A * E'), Rate('k * A * E'), Rate('0')])
sys.addPart(reaction)
T = None
Y = None

pulses = []

# initial condition
pulses.append(Pulse(0, 'A', 10))

# spike in some A
pulses.append(Pulse(100, 'A', 20))

# spike in a bit less A
pulses.append(Pulse(150, 'A', 5))

# spike in more A again
pulses.append(Pulse(250, 'A', 10))

# stop the simulation at time 500 with this empty string as the state
# variable parameter
pulses.append(Pulse(500, '', 0))

# Run pulsed simulation
(T, Y) = sys.run_pulses(pulses)

# Plot the simulation data
plt.figure()
plt.plot(T, Y[:, sys.compositorIndex('A')], label="A")
plt.plot(T, Y[:, sys.compositorIndex('B')], label="B")
plt.plot(T, Y[:, sys.compositorIndex('E')], label="E")
plt.legend()
plt.xlabel('Time')
plt.ylabel('Concentration')
plt.show()
```

Parameters

self	The object pointer.
pulse_series	List of pulse objects.

Returns

Tuple (T, Y), where T - time point list, Y - matrix consisting of Compositor values at a time points.

3.1.3.13 def Biosystem.BioSystem.sys_ode (*self*, *y*, *t*)

Ordinary diferential equatation of the system.

Parameters

self	The object pointer.
y	System compositors values.
t	Time point.

Returns

change of Compositor values.

3.1.3.14 def Biosystem.BioSystem.time_to_index (*ignore*, *T*, *t*)

Find the index in T (time point) list that gives a value just before t or exact t.

Parameters

ignore	Ignored argument.
T	A list of time points.
t	Time.

Returns

Index of T just before t (or exact t).

3.1.4 Member Data Documentation

3.1.4.1 Biosystem.BioSystem.compositors

Compositors involving chemical reactions.

3.1.4.2 Biosystem.BioSystem.constants

Constants in a Biosystem.

3.1.4.3 Biosystem.BioSystem.map_compositors

A mapping between compositor name and its index in `constants` list.

3.1.4.4 Biosystem.BioSystem.map_constants

A mapping between constant name and its index in `constants` list.

3.1.4.5 Biosystem.BioSystem.parts

Parts in a Biosystem.

3.1.4.6 Biosystem.BioSystem.rates_determined

Flag if function determine_rates was called.

All Constant symbols.

If False we need to call determine_rates again.

All Constant values. Pairs of symbols and its value. Convert Compositor rate string/formula to sympy expression and substitute all constants with its values. Convert sympy expression to python function with arguments: time, compositor1_name, compositor2_name, ...

3.1.4.7 Biosystem.BioSystem.symbols

List of all the Compositor symbols in ths BioSystem by initializing symbols with t we allow t to be a variable of time that's not a Compositor or Constant.

The documentation for this class was generated from the following file:

- Biosystem.py

3.2 Compositor.Compositor Class Reference

Substance concentration.

Public Member Functions

- def __init__ (self, name, init_value=0)

  The constructor.
- def addRate (self, new_rate)

  Add new rate represented as a string.
- def setInitialValue (self, init_value)

  Set initial concentration.

Public Attributes

- rate

  Rate formula of the compositor.
- name

  Substance name in a system.
- sym

  A Symbol with a `name` *representing a substance.*
- init_value

  Initial concentration of a substance.
- value

  Current concentration of a substance.

3.2.1 Detailed Description

Substance concentration.

A Compositor is the total rate of change of a state variable, e.g. the concentration of some chemical substances, say dEnzyme/dt.

Author

Eglė Plėštytė

Date

2017-05-10

3.2.2 Constructor & Destructor Documentation

3.2.2.1 def Compositor.Compositor.__init__ (*self,* *name,* *init_value* = 0)

The constructor.

Parameters

self	The object pointer.
name	Name assigned to a Compositor (Substance name).
init_value	Initial concentration of a substance.

3.2.3 Member Function Documentation

3.2.3.1 def Compositor.Compositor.addRate (*self,* *new_rate*)

Add new rate represented as a string.

Parameters

self	The object pointer.
new_rate	Rate to add.

Returns

The object pointer.

3.2.3.2 def Compositor.Compositor.setInitialValue (*self,* *init_value*)

Set initial concentration.

Parameters

self	The object pointer.
init_value	Initial concentration of a substance.

Returns

The object pointer.

3.2.4 Member Data Documentation

3.2.4.1 Compositor.Compositor.init_value

Initial concentration of a substance.

3.2.4.2 Compositor.Compositor.name

Substance name in a system.

3.2.4.3 Compositor.Compositor.rate

Rate formula of the compositor.

3.2.4.4 Compositor.Compositor.sym

A Symbol with a `name` representing a substance.

3.2.4.5 Compositor.Compositor.value

Current concentration of a substance.

The documentation for this class was generated from the following file:

- Compositor.py

3.3 Const.Const Class Reference

A Const is some constant in a system.

Public Member Functions

- def __init__ (self, name, value=0)
  The constructor.

Public Attributes

- name

  A name of Const.
- sym

  A Symbol with a `name` representing a constant.
- value

  A value of Const.

3.3.1 Detailed Description

A Const is some constant in a system.

A Const class defines numeric constant in a system.

Author

Eglė Plėštytė

Date

2017-05-10

3.3.2 Constructor & Destructor Documentation

3.3.2.1 def Const.Const.__init__ (*self*, *name*, *value* = `0`)

The constructor.

Parameters

self	The object pointer.
name	Name assigned to a Const.
value	Value of a constant.

3.3.3 Member Data Documentation

3.3.3.1 Const.Const.name

A name of Const.

3.3.3.2 Const.Const.sym

A Symbol with a `name` representing a constant.

3.3.3.3 Const.Const.value

A value of Const.

The documentation for this class was generated from the following file:

- Const.py

3.4 Part.Part Class Reference

Representation of a chemical reaction.

Public Member Functions

- def __init__ (self, name, compositors, rates)

    The constructor.

Public Attributes

- name

    Name assigned to a Part.
- compositors

    Compositors involving chemical reactions.
- rates

    Chemical reactions substances change.

3.4.1 Detailed Description

Representation of a chemical reaction.

A Part is a process, changing the values of compositors according to some rate laws.

Author

Eglė Plėštytė

Date

2017-05-10

3.4.2 Constructor & Destructor Documentation

3.4.2.1 def Part.Part.__init__ (*self*, *name*, *compositors*, *rates*)

The constructor.

Parameters

self	The object pointer.
name	Name assigned to a Part.
compositors	Compositors involving chemical reactions.
rates	Chemical reactions substances change.

3.4.3 Member Data Documentation

3.4.3.1 Part.Part.compositors

Compositors involving chemical reactions.

3.4.3.2 Part.Part.name

Name assigned to a Part.

3.4.3.3 Part.Part.rates

Chemical reactions substances change.

The documentation for this class was generated from the following file:

- Part.py

3.5 Pulse.Pulse Class Reference

Chemical reaction substance change of concentration at particular time.

Public Member Functions

- def __init__ (self, time, compositor_name, value)

  The constructor.

Public Attributes

- time

  The time when to change a concentration of the `compositor_name` *compositor.*
- compositor_name

  Compositor name in Biosystem.
- value

  Compositor `compositor_name` *concentration value at* `time`*.*

3.5.1 Detailed Description

Chemical reaction substance change of concentration at particular time.

A Pulse tells that at time `time` we should set value of the compositor named `compositor_name` to `value` in our simulation.

Author

Eglė Plėštytė

Date

2017-05-10

3.5.2 Constructor & Destructor Documentation

3.5.2.1 def Pulse.Pulse.__init__ (*self, time, compositor_name, value*)

The constructor.

Parameters

self	The object pointer.
time	The time when to change a concentration of the `compositor_name` compositor.
compositor_name	Represents Biosystem compositor.
value	Compositor `compositor_name` concentration value at `time`.

3.5.3 Member Data Documentation

3.5.3.1 Pulse.Pulse.compositor_name

Compositor name in Biosystem.

3.5.3.2 Pulse.Pulse.time

The time when to change a concentration of the `compositor_name` compositor.

3.5.3.3 Pulse.Pulse.value

Compositor `compositor_name` concentration value at `time`.

The documentation for this class was generated from the following file:

- Pulse.py

3.6 Rate.Rate Class Reference

The representation of the rate law.

Public Member Functions

- def __init__ (self, rate_string)

  The constructor.
- def __str__ (self)

  String representation of a rate object.

Public Attributes

- rate_string

  Chemical reaction rate string formula.

3.6.1 Detailed Description

The representation of the rate law.

A Rate class defines chemical reaction rate formula using a string representation of a rate law involving compositors, constants, and potentially other functions (including of time).

Author

Eglė Plėštytė

Date

2017-05-10

3.6.2 Constructor & Destructor Documentation

3.6.2.1 def Rate.Rate.__init__ (*self*, *rate_string*)

The constructor.

Parameters

self	The object pointer.
rate_string	Chemical reaction rate formula represented as a string

3.6.3 Member Function Documentation

3.6.3.1 def Rate.Rate.__str__ (*self*)

String representation of a rate object.

Parameters

self	The object pointer.

Returns

string representation of a rate law

3.6.4 Member Data Documentation

3.6.4.1 Rate.Rate.rate_string

Chemical reaction rate string formula.

The documentation for this class was generated from the following file:

- Rate.py

Index