
yaml2sbml

Release 0.2.4

Jakob Vanhoefer, Marta R. A. Matos, Dilan Pathirana, Yannik Schä

Dec 09, 2022

MAIN

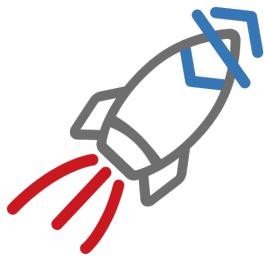
1	Installation	3
2	Input Format for yaml2sbml	5
3	Examples	9
4	API documentation	27
5	Release Notes	33
6	License	35
7	Logo	37
8	Contact	39
9	Contribute	41
10	Deploy	43
	Python Module Index	45
	Index	47

Release

0.2.4

Source code

<https://github.com/yaml2sbml-dev/yaml2sbml>



yaml2sbml

yaml2sbml allows the user to convert a system of ODEs specified in a [YAML](#) file to [SBML](#). In addition, if experimental data is provided in the YAML file, it can also be converted to [PETab](#).

INSTALLATION

This package requires Python 3.6 or later. [Miniconda](#) provides a small installation.

1.1 Install from PyPI

Install yam12sbml from [PyPI](#) via:

```
pip install yam12sbml
```

1.2 Install from GitHub

To work with the latest development version, install yam12sbml from [GitHub](#) via:

```
pip install https://github.com/yam12sbml-dev/yam12sbml/archive/develop.zip
```

or clone the repository and install from local via:

```
git clone https://github.com/yam12sbml-dev/yam12sbml
cd yam12sbml
git checkout develop
pip install -e .
```

where `-e` is short for `--editable` and links the installed package to the current location, such that changes there take immediate effect.

1.3 Additional dependencies for running the examples

The notebooks come with additional dependencies. Information on the installation of the ODE simulator [AMICI](#) is given in its [installation guide](#). Further dependencies can be installed via:

```
pip install yam12sbml[examples]
```


INPUT FORMAT FOR YAML2SBML

2.1 General scope

- *yaml2sbml*: translate ODEs (initial value problems) of the form $x' = f(t, x, p)$ with time t , states x and (potentially) unknown parameters p into an SBML file for simulation purpose.
- *yaml2PEtab*: define a fitting problem of the form $y(t_i) = h(x(t_i), p) + \text{eps}_i$ with independent normal- or Laplace-distributed error terms eps . h denotes the mapping from system states to observables. PEtab allows one to formulate MLE and MAP based fitting problems.

2.2 General remarks

- All identifiers of states, parameters etc. need to be valid SBML identifiers. Therefore, identifiers must consist of only upper and lower case letters, digits and underscores, and must not start with a digit.
- Mathematical equations are parsed by *libsbml*'s `parseL3Formula`. Hence for correct syntax see its [documentation](#) and the corresponding section of the format specification.
- Equations starting with a minus must be surrounded by brackets or quotation marks, since a leading minus also has a syntactic meaning in YAML and the YAML file will not be valid otherwise.

2.3 time [optional]

```
time:  
  variable: t
```

Define the **time variable**, in case the right hand side of the ODE is time-dependent.

2.4 parameters [optional]

```
parameters:  
  - parameterId: p_1  
    nominalValue: 1  
  
  - parameterId: p_2  
    ...
```

Define **parameters**. *nominalValue* is optional for SBML/PETab generation, but will be needed for model simulation. Further optional entries are *parameterName*, *parameterScale*, *lowerBound*, *upperBound*, *estimate* and entries regarding priors. These entries will be written in the corresponding column of the `_parameter` **table** by *yaml2PETab*.

For a detailed description see the documentation of the [PETab parameter table](#) (“PETab parameter table documentation”).

Further entries are possible and will be written to the `_parameter` **table** as well but are currently not part of the PETab standard.

2.5 odes

```
odes:
- stateId: x_1
  rightHandSide: p_1 * x_1
  initialValue: 1

- stateId: x_2
  ...
```

Define **ODEs** (and states). An ODE consists of a *stateId* (string), a *rightHandSide* (string, encoding a mathematical expression), and an *initial value*. Initial values can be either numerical values or parameter Ids.

For a more detailed description of the parsing of mathematical expressions (for *rightHandSide*) we refer to the corresponding section of this documentation.

2.6 assignments [optional]

```
assignments:
- assignmentId: sum_of_states
  formula: x_1 + x_2

- assignmentId: ...
  ...
```

Assign the mathematical expression *formula* to the term *assignmentId*. The value is dynamically updated and can depend on parameters, states and time. In SBML, assignments are represented via parameter assignment rules.

For a more detailed description of the parsing of mathematical expressions (e.g. for *formula*) we refer to the [corresponding section](#parsing-of-mathematical-equations) of this documentation.

2.7 functions [optional]

```
functions:
- functionId: g_1
  arguments: x_1, s
  formula: s * x_1 + 1

- functionId: g_2
  ...
```

Define **functions**, which can be called in other parts of the ODE definitions, e.g. in the example above via $g_I(x_I, s)$.

Please note that all unknowns appearing in the formula (e.g. also parameters or the time variable) also have to be arguments of the function.

For a more detailed description of the parsing of mathematical expressions (e.g. for *formula*) we refer to the [corresponding section](#parsing-of-mathematical-equations) of this documentation.

2.8 observables [optional]

observables:

- **observableId:** Obs_1
observableFormula: $x_1 + x_2$

noiseFormula: noiseParameter1
noiseDistribution: normal
- **observableId:** Obs_2
...

Define **observables**. Observables are not part of the SBML standard. If the SBML is generated via the `yaml2sbml.yaml2sbml` command and the `observables_as_assignments` flag is set to `True`, observables are represented as assignments to parameters of the form `observable_<observable_id>`. If the SBML is created via `yaml2sbml.yaml2petab`, observables are represented in the PETab observables table. The entries are written to the corresponding columns of the PETab observable table. According to the PETab standard, an observable table can take the following entries: *observableId*, *observableName*, *observableFormula*, *observableTransformation*, *noiseFormula*, *noiseDistribution*.

For a detailed discussion see the corresponding part of the PETab documentation <https://github.com/PETab-dev/PETab/blob/master/doc/documentation_data_format.rst#observables-table>`_.

2.9 conditions [optional]

conditions:

- **conditionId:** condition1
p_1: 1
x_1: 2
...

Conditions allow one to set parameters or initial conditions of states to a numeric value/unknown parameter. This allows for the specification of different experimental setups in the data generation (e.g. different initial conditions for different runs of an experiment).

The “trivial condition table” (if only one setup exists) is generated by:

conditions:

- **conditionId:** condition1

For a detailed discussion see the corresponding part of the [PETab documentation](#).

2.10 Parsing of mathematical equations

Throughout *yaml2sbml* formulas are parsed by *libsbml*'s *parseL3Formula* function. Further information on the syntax are given by:

- the [working with math](#) - section of the *libsbml* documentation.
- the [documentation](#) of *libsbml.parseL3Formula*.

This gives access to e.g.:

- +, -, *, /, and ^{power};
- trigonometric/hyperbolic functions;
- exponential/logarithmic functions;
- piecewise defined functions (via *piecewise*); and
- boolean expressions like “<”.

EXAMPLES

We provide the following Jupyter notebooks with examples:

- Three notebooks use the Lotka Volterra equations as a running example for the Python tool, CLI and the Model Editor.
- A notebook demonstrating format features with minimal examples.
- A notebook showing the Sorensen model as a real world application.
- A notebook implementing the Finite State Projection.

These examples can also be found on Github [here](#).

3.1 Example: “Lotka-Volterra” Equations

3.1.1 Scope

This notebook introduces the **input format** of [yaml2sbml](#) and showcases the **Python interface** of the tool. Furthermore, this notebook demonstrates the **simulation** of an SBML model using [AMICI](#) and **parameter estimation** from PETab using [pyPESTO](#).

3.1.2 Equations

The “Lotka-Volterra” Equations are given by

$$\begin{aligned}\frac{d}{dt}x_1 &= \alpha x_1 - \beta x_1 x_2, \\ \frac{d}{dt}x_2 &= \delta x_1 x_2 - \gamma x_2.\end{aligned}$$

Here x_1 denotes the number of a prey and x_2 the number of a predator species and $\alpha, \beta, \gamma, \delta$ are positive parameters describing the birth/death and interactions of the two species.

3.1.3 YAML file for model simulation in SBML:

The **states** x_1 and x_2 , as well as their initial values and dynamics are defined in the `odes:` section:

```
odes:
  - stateId: x_1
    rightHandSide: alpha * x_1 - beta * x_1 * x_2
    initialValue: 2

  - stateId: x_2
    rightHandSide: delta * x_1 * x_2 - gamma * x_2
    initialValue: 2
```

Furthermore, the **parameters** together with their values are defined in the `parameters:` section:

```
parameters:
  - parameterId: alpha
    nominalValue: 2

  - parameterId: beta
    nominalValue: 4

  ...
```

The full file is given in `Lotka_Volterra_basic.yml`.

3.1.4 Format validation of the YAML model:

The YAML model can be validated by the `yaml2sbml` package via:

```
[1]: import yaml2sbml

yaml_file_basic = 'Lotka_Volterra_basic.yml'
yaml2sbml.validate_yaml(yaml_dir=yaml_file_basic)

YAML file is valid
```

3.1.5 Conversion to SBML for model simulation:

We now want to convert the YAML file into a SBML file. Within Python this is possible via:

```
[2]: import yaml2sbml

sbml_output_file = 'Lotka_Volterra_basic.xml'
yaml2sbml.yaml2sbml(yaml_file_basic, sbml_output_file)
```

We will now use the SBML simulator **AMICI** to simulate the ODEs. First AMICI generates the models' C++ code.

```
[3]: %%capture

import amici
```

(continues on next page)

(continued from previous page)

```
import amici.plotting

model_name = 'Lotka_Volterra'
model_output_dir = 'Lotka_Volterra_AMICI'

sbml_importer = amici.SbmlImporter(sbml_output_file)
sbml_importer.sbml2amici(model_name,
                        model_output_dir)
```

Then we can reload and simulate the compiled model.

```
[4]: %%capture

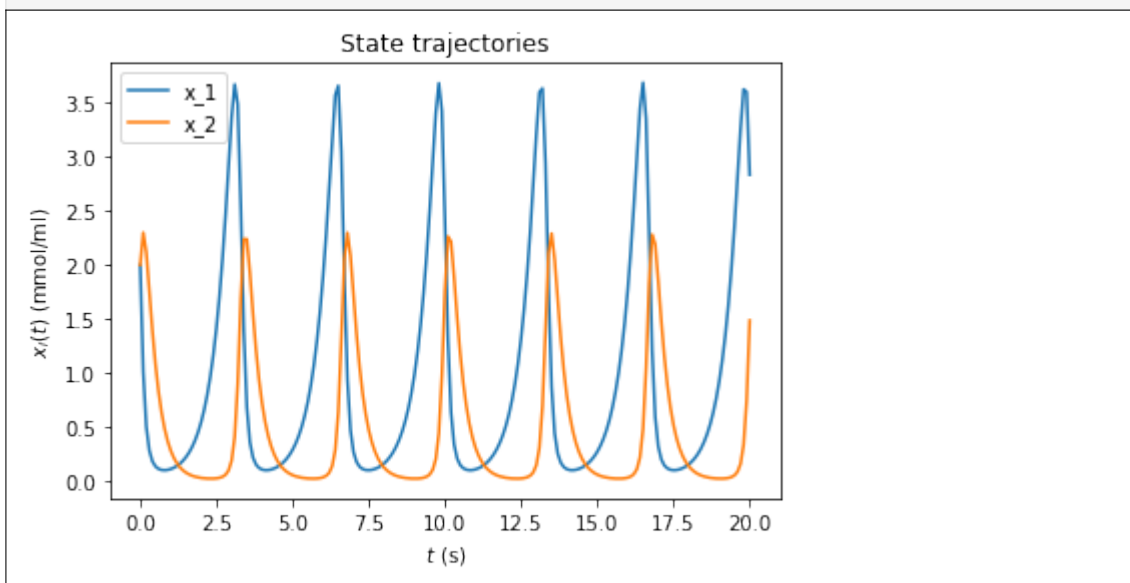
import os
import sys
import importlib
import numpy as np

# import model
sys.path.insert(0, os.path.abspath(model_output_dir))
model_module = importlib.import_module(model_name)

# create model + solver instance
model = model_module.getModel()
solver = model.getSolver()

# Define time points and run simulation using default model parameters/solver_
↳ options
model.setTimepoints(np.linspace(0, 20, 201))
rdata = amici.runAmiciSimulation(model, solver)
```

```
[5]: # plot trajectories
amici.plotting.plotStateTrajectories(rdata, model=model)
```



3.1.6 Conversion to P_Etab for parameter fitting:

In order to obtain valid P_Etab tables, we need to extend the .yaml file. The extended .yaml file is given as Lotka_Volterra_P_Etab.yaml.

Parameters:

The parameters: section can be extended to store parameter bounds, transformations, ... This information is written to the P_Etab *parameter table*.

```
parameters:
  - parameterId: alpha
    nominalValue: 2
    parameterScale: log10
    lowerBound: 0.01
    upperBound: 100
    estimate: 1
  ...
```

(**Note:** P_Etab measurement tables are outside of the scope of yaml2sbml. Hence a predefined measurement table for this example is given under ./Lotka_Volterra_P_Etab/measurement_table.tsv.)

Observables:

The observables: section allows to specify the mapping of system state to measurement and the measurement noise. This information is written in the P_Etab *observable table*.

```
observables:
  - observableId: prey_measured
    observableFormula: log10(x_1)
    observableTransformation: lin
    noiseFormula: 1
    noiseDistribution: normal
```

Conditions:

The conditions: section allows to specify different experimental setups. This information is written in the P_Etab *condition table*. This example only defines the trivial condition table, consisting of only one experimental setup.

```
conditions:
  - conditionId: condition1
```


yaml2petab:

We want to create the corresponding PETab tables for the model in `Lotka_Volterra_PETab.yml`. Since the definition of PETab measurement tables is outside of the scope of `yaml2sbml`, a predefined measurement table is given under `./Lotka_Volterra_PETab/measurement_table.tsv`.

```
[6]: yaml_file_petab = 'Lotka_Volterra_PETab.yml'
      PETab_dir = './Lotka_Volterra_PETab/'
      PETab_yaml_name = 'problem.yml'
      measurement_table_name = 'measurement_table.tsv'
      model_name = 'Lotka_Volterra_with_observables'

      yaml2sbml.yaml2petab(yaml_file_petab,
                           PETab_dir,
                           model_name,
                           PETab_yaml_name,
                           measurement_table_name)
```

`yaml2petab` created the PETab files. In the following we show how to import and fit a PETab problem in `pyPESTO`.

```
[7]: %%capture
      import pypesto
      import pypesto.petab
      import pypesto.optimize as optimize
      import pypesto.visualize as visualize

      # import PETab problem
      importer = pypesto.petab.PetabImporter.from_yaml(os.path.join(PETab_dir, PETab_
      ↪yaml_name),
                                                         model_name=model_name)

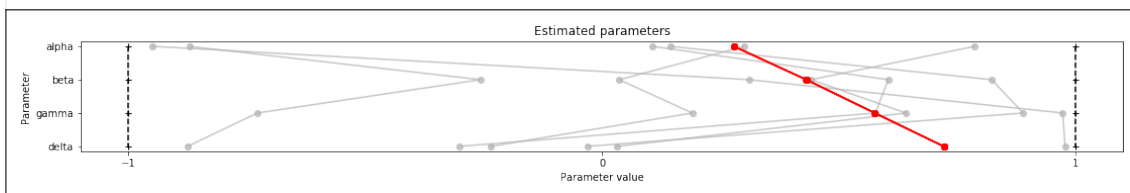
      problem = importer.create_problem()
```

Perform the fitting:

```
[8]: %%capture
      # perform optimization
      optimizer = optimize.ScipyOptimizer()
      result = optimize.minimize(problem,
                                optimizer=optimizer,
                                n_starts=10)
```

Next we want to visualize the results using parallel coordinate plots.

```
[9]: visualize.parameters(result)
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff95ff93d30>
```



3.2 Command Line Interface

This notebook shows the **command line interface** of yaml2sbml, using the “Lotka Volterra” example. For details on * the definition of the .yaml file * simulation of the SBML model using AMICI (Python) * fitting the Petab problem using pyPESTO (Python),

see the *corresponding notebook*

3.2.1 YAML validation:

yaml2sbml allows to validate the input YAML via

```
yaml2sbml_validate <yaml_inout_file>
```

```
[1]: !yaml2sbml_validate ../Lotka_Volterra_python/Lotka_Volterra_basic.yaml
Path to yaml file: ../Lotka_Volterra_python/Lotka_Volterra_basic.yaml
Validating...
YAML file is valid
```

3.2.2 SBML generation

yaml2sbml generates an **SBML** via

```
yaml2sbml <yaml_input_file> <sbml_output_file>
```

```
[2]: !yaml2sbml ../Lotka_Volterra_python/Lotka_Volterra_basic.yaml Lotka_Volterra_
↳ basic.xml
Path to yaml file: ../Lotka_Volterra_python/Lotka_Volterra_basic.yaml
Path to sbml file: Lotka_Volterra_basic.xml
Converting...
```

3.2.3 Petab generation:

The basic way of generating **PETab** tables is given by

```
yaml2petab <yaml_input_file> <petab_output_directory> <model_name>
```

```
[3]: # create target directory, if it doesn't exist yet
!mkdir -p ../Lotka_Volterra_PETab/
# needed .yaml /measurement files in current directory.
!cp ../Lotka_Volterra_python/Lotka_Volterra_PETab/measurement_table.tsv ./
↳ Lotka_Volterra_PETab/
!cp ../Lotka_Volterra_python/Lotka_Volterra_PETab.yaml .
```

```
[4]: # This is the actual command
!yaml2petab Lotka_Volterra_PETab.yaml ../Lotka_Volterra_PETab Lotka_Volterra
```

```
Path to yaml file: Lotka_Volterra_PEtab.yaml
Output directory: ./Lotka_Volterra_PEtab
Path to sbml/petab files: Lotka_Volterra
Converting...
```

Further Options:

Two further optional arguments to the `yaml2petab` command allow to create a *PEtab* *yaml* file and a measurement table. A *PEtab* *yaml* allows for a easier input of a *PEtab* problem to some toolboxes: * In the directory , given by `--petab_yaml`, a *yaml* file, that groups the *PEtab* problem, will be created. * `--measurement_table` allows to specify the measurement table. This option is only possible in combination with `--petab_yaml`.

```
[5]: # now generate the PEtab table:
!yaml2petab Lotka_Volterra_PEtab.yaml ./Lotka_Volterra_PEtab Lotka_Volterra --
  ↪petab_yaml petab_problem.yaml --measurement_table measurement_table.tsv
```

```
Path to yaml file: Lotka_Volterra_PEtab.yaml
Output directory: ./Lotka_Volterra_PEtab
Path to sbml/petab files: Lotka_Volterra
Converting...
```

3.2.4 Checking the *PEtab* files

Finally, *PEtab* offers a *PEtab*-linter (no output = no error found). (This is a feature of the *PEtab* python library, not of `yaml2sbml`.)

```
[6]: !petablint -vy Lotka_Volterra_PEtab/petab_problem.yaml
```

```
Checking SBML model...
Checking measurement table...
Checking condition table...
Checking observable table...
Checking parameter table...
OK
```

3.3 Model Editor

This notebook demonstrates functionality of the `yaml2sbml` model editor using the Lotka Volterra equations as an example. The “Lotka-Volterra” equations are given by

$$\begin{aligned}\frac{d}{dt}x_1 &= \alpha x_1 - \beta x_1 x_2, \\ \frac{d}{dt}x_2 &= \delta x_1 x_2 - \gamma x_2.\end{aligned}$$

3.3.1 ODE model

We first generate a basic model, that only contains all necessary information for generating an SBML file for model simulation.

```
[1]: from yaml2sbml import YamlModel

# generate model
model = YamlModel()

# add ODEs
model.add_ode(state_id='x_1',
              right_hand_side='alpha*x_1 - beta*x_1*x_2',
              initial_value=2)
model.add_ode(state_id='x_2',
              right_hand_side='delta*x_1*x_2 - gamma*x_2',
              initial_value=2)

# add parameters
model.add_parameter(parameter_id='alpha', nominal_value=2)
model.add_parameter(parameter_id='beta', nominal_value=4)
model.add_parameter(parameter_id='gamma', nominal_value=3)
model.add_parameter(parameter_id='delta', nominal_value=3)
```

yaml2sbml can export the model object either to YAML or to SBML directly, via

```
[2]: # write to YAML
model.write_to_yaml('Lotka_Volterra_basic.yaml', overwrite=True)

# write to SBML
model.write_to_sbml('Lotka_Volterra_basic.xml', overwrite=True)
```

There are further functions to: * get all parameter_ids via `model.get_parameter_ids()` * get a parameter by its id (`model.get_parameter_by_id('alpha')`) * delete a parameter by its id (`model.delete_parameter('alpha')`)

Similar functions also exist for the other model components.

3.3.2 Parameter Estimation Problem

Now we want to extend the current model to include all the necessary information for parameter estimation in PETab. Therefore we load the model from the .yaml file and modify the parameters, such that it also contains all information that is going to be written into the PETab parameter table.

```
[3]: model = YamlModel.load_from_yaml('Lotka_Volterra_basic.yaml')

# extend parameters
model.add_parameter(parameter_id='alpha',
                    nominal_value=2,
                    parameter_scale='log10',
                    lower_bound=0.1,
                    upper_bound=10,
                    estimate=1,
```

(continues on next page)

(continued from previous page)

```

        overwrite=True)

model.add_parameter(parameter_id='beta',
                    nominal_value=4,
                    parameter_scale='log10',
                    lower_bound=0.1,
                    upper_bound=10,
                    estimate=1,
                    overwrite=True)

model.add_parameter(parameter_id='gamma',
                    nominal_value=3,
                    parameter_scale='log10',
                    lower_bound=0.1,
                    upper_bound=10,
                    estimate=1,
                    overwrite=True)

model.add_parameter(parameter_id='delta',
                    nominal_value=3,
                    parameter_scale='log10',
                    lower_bound=0.1,
                    upper_bound=10,
                    estimate=1,
                    overwrite=True)

```

A parameter fitting problem in PETab allows for the specification of observables and experimental conditions:

```

[4]: # specify an observable:
model.add_observable(observable_id='prey_measured',
                    observable_formula='log10(x_1)',
                    noise_formula='noiseParameter1_prey_measured',
                    noise_distribution='normal',
                    observable_transformation='lin')

# specify trivial condition
model.add_condition(condition_id='condition1',
                    condition_dict={})

```

The modified model can either be exported to YAML or PETab via

```

[5]: # write to YAML
model.write_to_yaml('Lotka_Volterra_PETab.yaml', overwrite=True)

# write to PETab
model.write_to_petab(output_dir='./Lotka_Volterra_PETab',
                    model_name='Lotka_Volterra',
                    petab_yaml_name='Lotka_Volterra_problem',
                    measurement_table_name='measurement.tsv')

```

3.4 Toy Examples of Format Features

This notebook demonstrates minimal examples of the following format features:

- Time-dependent right hand sides
- Step functions in the right hand side
- Function definitions

3.4.1 Time-dependent right hand side:

`time_dependent_rhs.yml` implements the ODE

$$\dot{x} = -\frac{x}{t+1} \quad (3.1)$$

$$x(0) = 1 \quad (3.2)$$

via

```
time:
  variable: t

odes:
  - stateId: x
    rightHandSide: -x/(t+1)
    initialValue: 1
```

The time variable `t` needs to be defined.

```
[1]: import yml2sbml

yaml_file_basic = 'time_dependent_rhs.yml'
sbml_output_file = 'time_dependent_rhs.xml'

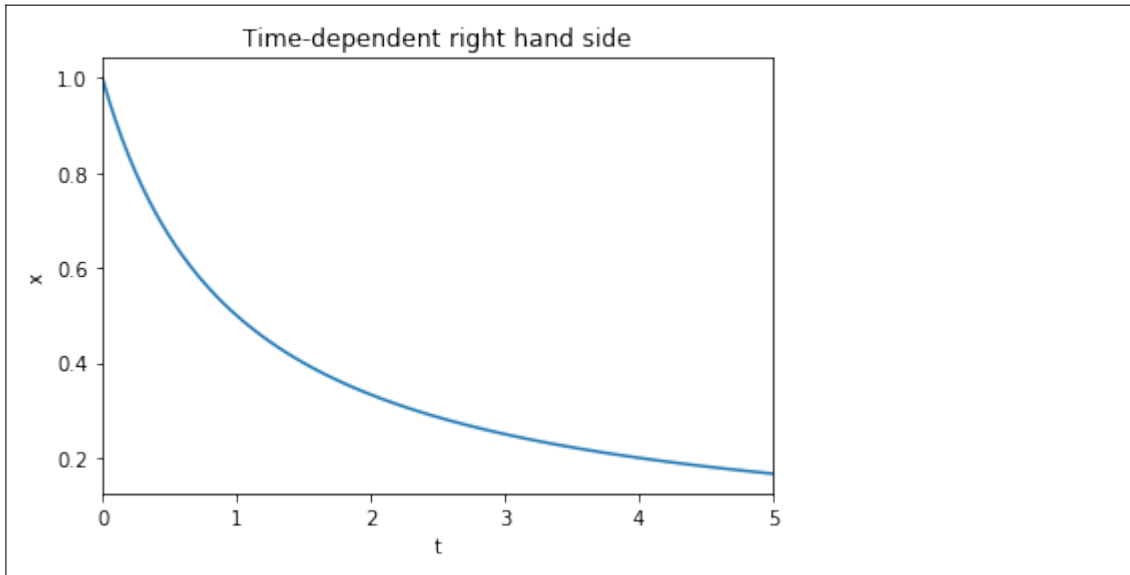
yml2sbml.yml2sbml(yaml_file_basic, sbml_output_file)
```

For the sake of brevity, simulation and plotting is done in a separate function. Details on these steps can be found in [other notebooks](#).

```
[2]: %matplotlib inline

from simulation_and_plotting import simulate_AMICI, plot_AMICI
import amici.plotting

#simulate
amici_model, rdata = simulate_AMICI(sbml_output_file)
#plot
plot_AMICI(amici_model, rdata, 'Time-dependent right hand side')
```



3.4.2 Step functions in the right hand side

step_function.yml implements the ODE

$$\dot{x} = \begin{cases} x, & t < 1 \\ -x, & \text{otherwise} \end{cases}$$

via

```
time:
  variable: t

odes:
  - stateId: x
    rightHandSide: piecewise(x, t < 1, -x)
    initialValue: 1
```

More details on piecewise functions can be found in the [libsbml documentation](#).

```
[3]: import yaml2sbml

yaml_file_basic = 'step_function.yml'
sbml_output_file = 'step_function.xml'

# translate yaml file to sbml
yaml2sbml.yaml2sbml(yaml_file_basic, sbml_output_file)
```

As discussed above, we plot via:

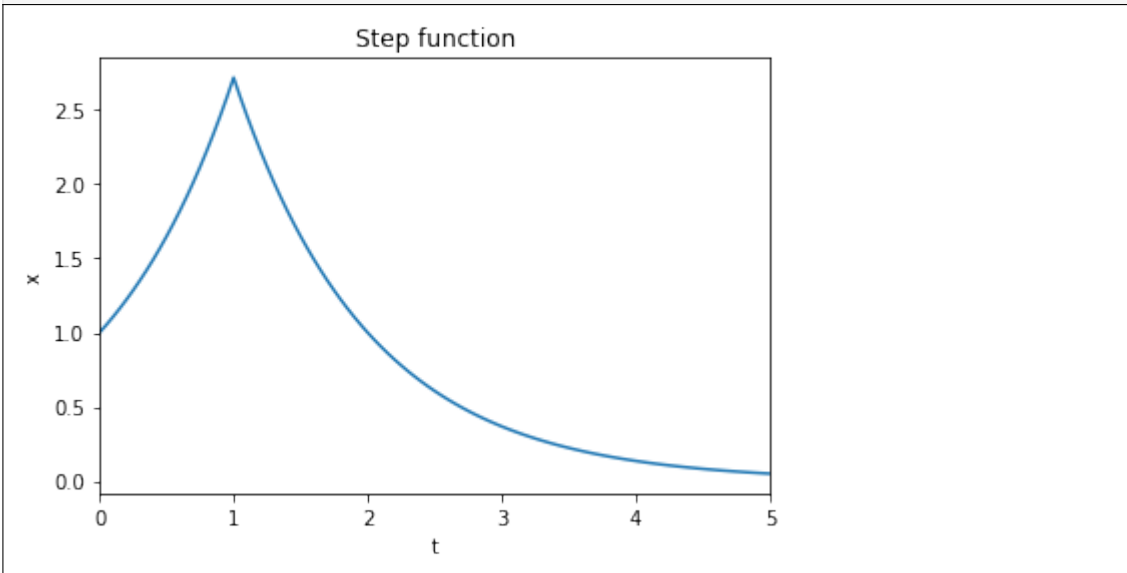
```
[4]: %matplotlib inline

#simulate
amici_model, rdata = simulate_AMICI(sbml_output_file)
```

(continues on next page)

(continued from previous page)

```
#plot
plot_AMICI(amici_model, rdata, 'Step function')
```



3.4.3 Function definition

yaml2sbml allows to define functions, that can then be called in other parts of the model. `functions_in_rhs.yaml` implements the ODE

$$\begin{aligned}\dot{x}_1 &= f(x_2, 1, 2) \\ \dot{x}_2 &= f(x_1, -1, 0),\end{aligned}$$

where $f(x, a, b) = a \cdot x + b$ via

```
odes:
- stateId: x_1
  rightHandSide: f(x_2, 1, 2)
  initialValue: 1

- stateId: x_2
  rightHandSide: f(x_1, -1, 0)
  initialValue: 1

functions:
- functionId: f
  arguments: x, a, b
  formula: a * x + b
```

```
[6]: import yaml2sbml

yaml_file_basic = 'functions_in_rhs.yaml'
sbml_output_file = 'functions_in_rhs.xml'

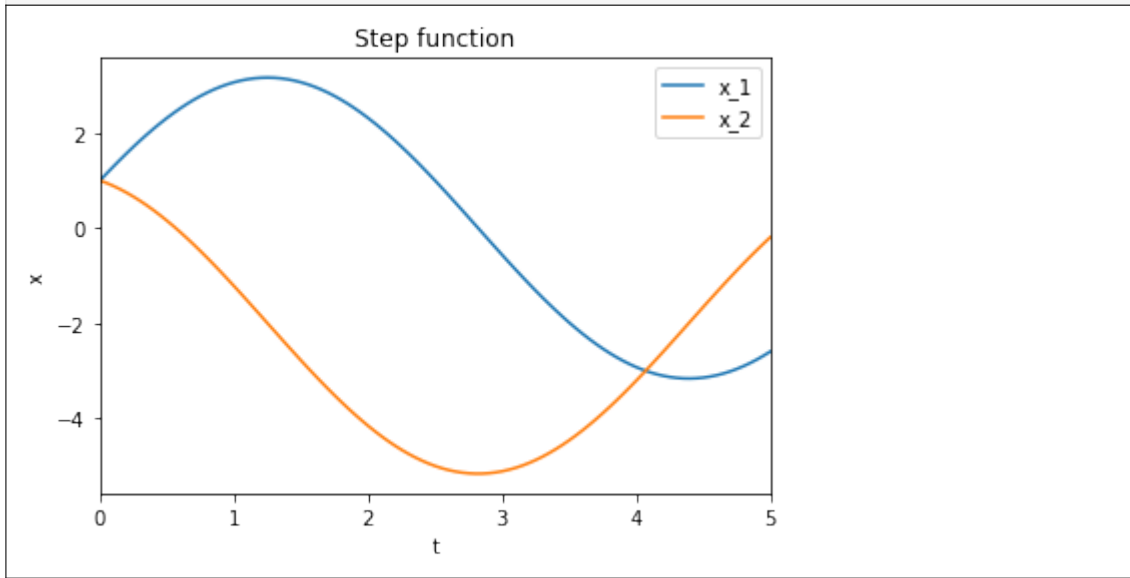
# translate yaml file to sbml
yaml2sbml.yaml2sbml(yaml_file_basic, sbml_output_file)
```


We now follow the usual work flow for simulation and plotting.

```
[7]: %matplotlib inline

#simulate
amici_model, rdata = simulate_AMICI(sbml_output_file)

#plot
plot_AMICI(amici_model, rdata, 'Step function')
```



3.5 Glucose Insulin Metabolism Model by Sorensen (1985).

In this notebook, the model of glucose and insulin metabolism, from the thesis of Thomas J. Sorensen, 1985 [1], is used as an example of how the `yaml2sbml.YamlModel` class can be used to easily extend a pre-existing `yaml2sbml` model.

Specifically, the model is edited to reproduce a figure from the original thesis (Fig 71) [1]. The implementation that is loaded here is based on the implementation provided by Panunzi et al., 2020 [2].

In the end, the model is exported to [SBML](#) and simulated via [AMICI](#).

3.5.1 Extend the current YAML model

The Sorensen model has already been encoded in the YAML format. Within this notebook, the model is loaded, extended in order to model an intravenous infusion administration of glucose, using the `yaml2sbml` Model editor. The extended model is then plotted, to reproduce a figure from Sorensens PhD thesis.

Load the model

```
[1]: import yaml2sbml
      from yaml2sbml import YamlModel

      yaml_model = YamlModel.load_from_yaml('Sorensen1985.yaml')
```

Define the glucose infusion term

An intravenous infusion administration of glucose is added to the model. After 17 minutes of initialization, glucose is infused at a rate of 64.81 mM/min for a three-minute period.

In the YAML model, this is represented by a novel term IVG (“intravenous glucose”), that is set by a step function in an assignment rule.

```
[2]: # describe the intravenous glucose infusion
      yaml_model.add_assignment('IVG', formula='piecewise(64.81, 17 <= t && t < 20, 0)')
```

Add the glucose infusion term to the model

IVG is now added to the ODE of the glucose concentration of heart and lung space `GlucH`. Therefore, the current ODE is overwritten.

```
[3]: # Get the current ODE.
      gluch_ode_old = yaml_model.get_ode_by_id('GlucH')
      gluch_rhs_old = gluch_ode_old['rightHandSide']

      # Modify the ODE. `IVG` is divided by the volume term `VolGH` to
      # model concentration.
      gluch_rhs_new = gluch_rhs_old + ' + (IVG / VolGH)'

      # Set the new ODE.
      yaml_model.add_ode(state_id='GlucH',
                        right_hand_side=gluch_rhs_new,
                        initial_value=gluch_ode_old['initialValue'],
                        overwrite=True)
```

Export to SBML

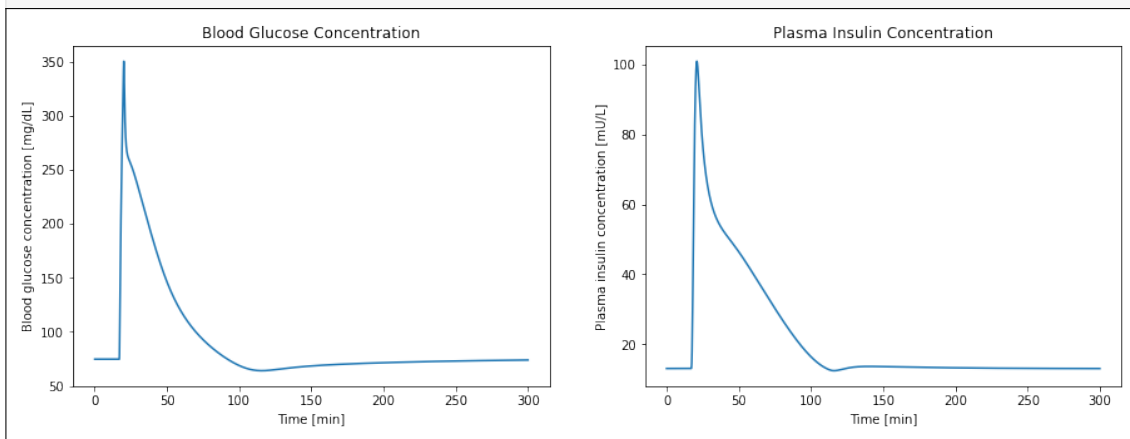
```
[4]: # Write to SBML
yaml_model.write_to_sbml('Fig71_Sorensen1985.xml', overwrite=True)
```

3.5.2 Simulation in AMICI

The model is now setup to reproduce to the figure. The utility function `simulate_and_plot_sorensen` simulates and plots the Sorensen model in `AMICI`.

```
[5]: %matplotlib inline
from utils import simulate_and_plot_sorensen

simulate_and_plot_sorensen('Fig71_Sorensen1985.xml')
```



Trajectories are plotted, which closely match the original thesis figures.

Note that this figure has a shift in the x-axis, as the simulation here started at `time == 0`, but in the thesis started at approximately `time == -20`. As there are no explicit time dependencies in the model, the figure is otherwise similar.

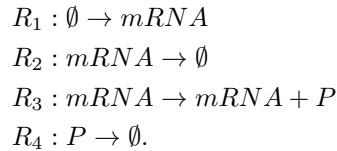
3.6 References

- [1] Sorensen, J. T. (1985). “A physiologic model of glucose metabolism in man and its use to design and assess improved insulin therapies for diabetes.” <https://dspace.mit.edu/handle/1721.1/15234>
- [2] Panunzi, S., Pompa, M., Borri, A., Piemonte, V., & De Gaetano, A. (2020). A revised Sorensen model: Simulating glycemic and insulinemic response to oral and intra-venous glucose load. Plos one, 15(8), e0237215. <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0237215>

3.7 Finite State Projection of a Gene Transmission Model

3.7.1 Model

In this application example, we consider the two-stage model of gene transmission, discussed e.g. in. [1]. The two-stage model describes the stochastic transcription and translation of a gene by the reactions



The solution to the Chemical Master Equation (CME) [2] gives the probabilities of the states at a given time point as the solution of an infinite dimensional ODE

$$\begin{aligned} \frac{d}{dt}x_{r,p} = & - (k_1 + (k_2 + k_3) \cdot r + k_4 \cdot p) \cdot x_{r,p} \\ & + k_1 \cdot x_{r-1,p} \\ & + k_2 \cdot (r + 1) \cdot x_{r+1,p} \\ & + k_3 \cdot r \cdot x_{r,p-1} \\ & + k_4 \cdot (p + 1) \cdot x_{r,p+1} \end{aligned}$$

We assume the initial probabilities to be independent Poisson distributions for mRNA and Protein abundances.

3.7.2 Finite State Projection

The Finite State Projection [3] approximates this CME by restricting the states to a finite domain and approximating the probability of the left out states by zero. In our case, we will restrict the states to $x_{r,p}$ for $(r, p) \in [0, r_{max} - 1] \times [[0, p_{max} - 1]]$.

3.7.3 Model Construction:

The model is constructed using the yaml2sbml Model Editor and exported to SBML. This takes only a few lines of code:

```
[1]: from yaml2sbml import YamlModel
      from itertools import product
      from scipy.stats import poisson

      model = YamlModel()

      r_max = 20
      p_max = 50

      lambda_r = 5
      lambda_p = 5

      # add parameters
```

(continues on next page)

(continued from previous page)

```

model.add_parameter(parameter_id='k_1', nominal_value=2)
model.add_parameter(parameter_id='k_2', nominal_value=1)
model.add_parameter(parameter_id='k_3', nominal_value=10)
model.add_parameter(parameter_id='k_4', nominal_value=3)

# add ODEs & construct the rhs
for r, p in product(range(r_max), range(p_max)):

    rhs = f'-(k_1 + (k_2 + k_3)*{r} + k_4*{p}) * x_{r}_{p} '

    if r>0:
        rhs += f'+ k_1 * x_{r-1}_{p}'
    if r+1 < r_max:
        rhs += f'+ k_2 * {r+1} * x_{r+1}_{p}'
    if p>0:
        rhs += f'+ k_3 * {r} * x_{r}_{p-1}'
    if p+1 < p_max:
        rhs += f'+ k_4 * {p+1} * x_{r}_{p+1}'

    model.add_ode(state_id = f"x_{r}_{p}",
                  right_hand_side=rhs,
                  #convert from np.float to float
                  initial_value=float(poisson.pmf(r, lambda_r)*poisson.pmf(p,
↪lambda_p))
    )

model.write_to_sbml('gene_expression.xml', overwrite=True)

```

3.7.4 Model Simulation

We now use AMICI to simulate the generated model and plot the marginal distributions of the mRNA and protein abundance.

```

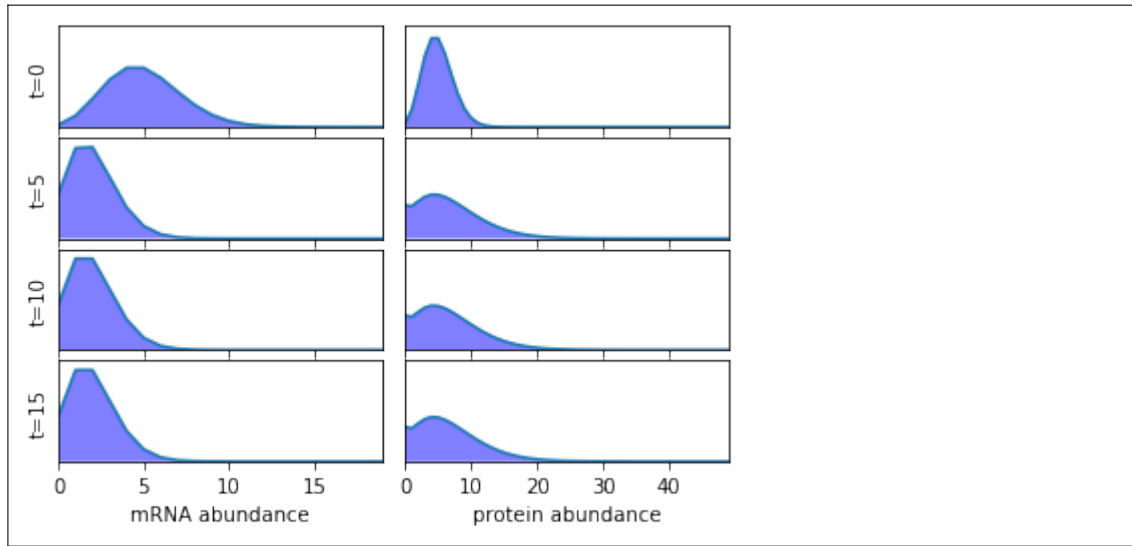
[2]: %matplotlib inline

import numpy as np
from utils import plot_AMICI

# sim, model = compile_and_simulate('gene_expression.xml', 0.1 * np.arange(10),
↪r_max, p_max)

plot_AMICI('gene_expression.xml', [0, 5, 10, 15], r_max, p_max)

```



3.8 References

- [1] Shahrezaei, V., and Swain, P.S. (2008). “Analytical distributions for stochastic gene expression” PNAS 17256–17261 <https://www.pnas.org/content/105/45/17256>
- [2] Gillespie, D. T. (1992). “A rigorous derivation of the chemical master equation” Physica A: Statistical Mechanics and its Applications <https://www.sciencedirect.com/science/article/abs/pii/037843719290283V>
- [3] Munsky, B. and Khammash, M. (2006) “The finite state projection algorithm for the solution of the chemical master equation” The Journal of chemical physics <https://aip.scitation.org/doi/full/10.1063/1.2145882>

API DOCUMENTATION

4.1 Convert YAML to SBML

Translate ODEs in the YAML format into SBML.

`yaml2sbml.yaml2sbml.main()`

Command-Line Interface.

`yaml2sbml.yaml2sbml.yaml2sbml(yaml_dir, sbml_dir, observables_as_assignments=False)`

Parse a YAML file with the specification of ODEs and write it to SBML.

SBML is written within this function. If *observables_as_assignments=True*, observables will be translated into parameter assignments of the form *observable_<observable_id>*.

Parameters

- **yaml_dir** (`str`) – directory to the YAML file with the ODEs specification
- **sbml_dir** (`str`) – directory to the SBML file to be written out
- **observables_as_assignments** (`bool`) – indicates whether there should be parameter assignments of the form *observable_<observable_id>*.

4.2 Validate YAML file

`yaml2sbml.validate_yaml(yaml_dir)`

Validate the syntax of the YAML file.

Parameters

yaml_dir (`str`) – path to YAML file to be validated

Returns

`jsonschema.validate`

4.3 Convert YAML to PTab

`yaml2sbml.yaml2petab(yaml_dir, output_dir, sbml_name, petab_yaml_name=None, measurement_table_name=None)`

Translate a YAML model into a PTab model.

Takes a YAML file with the ODE specification, parses it, converts it into SBML format, and writes the SBML file. Further it translates the given information into PTab tables.

If a *petab_yaml_name* is given, a YAML file is created, that organizes the PTab problem. If additionally a *measurement_table_file_name* is specified, this file name is written into the created YAML file.

Parameters

- **yaml_dir** (`str`) – path to the YAML file with the ODEs specification
- **output_dir** (`str`) – path the output file(s) are be written out
- **sbml_name** (`str`) – name of SBML model
- **petab_yaml_name** (`Optional[str]`) – name of YAML organizing the PTab problem.
- **measurement_table_name** (`Optional[str]`) – Name of measurement table

4.4 Validate PTab tables

`yaml2sbml.validate_petab_tables(sbml_dir, output_dir)`

Validate the PTab tables via *petab.lint*.

Throws an error if the PTab tables do not follow the PTab format standard.

Parameters

- **sbml_dir** (`str`) – directory of the sbml
- **output_dir** (`str`) – output directory for petab files

Raises

Errors are raised by lint, if PTab files are invalid... –

4.5 Model editor

`class yaml2sbml.YamlModel`

Functionality to set up, edit, load and write yaml models.

add_assignment(*assignment_id*, *formula*, *overwrite=False*)

Add assignment.

Overwrite an existing assignment with the same id, if *overwrite==True*.

Parameters

- **assignment_id** (`str`) – str, function id
- **formula** (`str`) – str, right hand side of assignment definition.
- **overwrite** (`bool`) – bool, indicates if an existing assignment should be overwritten

add_condition(*condition_id*, *condition_dict*, *overwrite=False*, *condition_name=None*)

Add condition *condition_id*.

Conditions are not represented inside an SBML and only play a role when generating a PETab problem (see PETabs condition table).

Overwrite an existing condition with the same id, if *overwrite==True*.

Parameters

- **condition_id** (*str*) – str, condition id
- **condition_dict** (*dict*) – dict, of the form {<parameter or state id>: <value>}. Corresponds to entries in the PETab condition table. See details there.
- **overwrite** (*bool*) – bool, indicates if an existing condition should be overwritten
- **condition_name** (*Optional[str]*) – Condition name. Optional.

add_function(*function_id*, *arguments*, *formula*, *overwrite=False*)

Add function.

Overwrite an existing function with the same id, if *overwrite==True*.

Parameters

- **function_id** (*str*) – str, function id
- **arguments** (*str*) – str, arguments, separated by a comma
- **formula** (*str*) – str, right hand side of the function definition
- **overwrite** (*bool*) – bool, indicates if an existing function should be overwritten

add_observable(*observable_id*, *observable_formula*, *noise_formula*, *overwrite=False*, *observable_name=None*, *observable_transformation=None*, *noise_distribution=None*)

Add observable.

Observables are not represented inside an SBML and only play a role when generating a PETab problem see PETabs observable table).

Overwrite an existing observable with the same id, if *overwrite==True*.

Parameters

- **observable_id** (*str*) – str, observable id
- **observable_formula** (*str*) – str, formula of the observable function
- **noise_formula** (*str*) – str, formula of the noise
- **overwrite** (*bool*) – bool, indicates if an existing observable should be overwritten
- **observable_name** (*Optional[str]*) – Observable name. Optional.
- **observable_transformation** (*Optional[str]*) – Observable transformation ('lin'/'log'/'log10'). Optional

add_ode(*state_id*, *right_hand_side*, *initial_value*, *overwrite=False*)

Add state/ODE.

Overwrite an existing state/ODE with the same id, if *overwrite==True*.

Parameters

- **state_id** (*str*) – str, state id

- **right_hand_side** (`Union[float, str]`) – str or float, right hand side of the ODE.
- **initial_value** (`Union[float, str]`) – str or float, initial value of the ODE at t=0
- **overwrite** (`bool`) – bool, indicates if an existing state/ODE should be overwritten

add_parameter(*parameter_id*, *overwrite=False*, *nominal_value=None*, *parameter_name=None*, *parameter_scale=None*, *lower_bound=None*, *upper_bound=None*, *estimate=None*)

Add a parameter.

Overwrite an existing parameter with the same id, if *overwrite==True*.

Parameters

- **parameter_id** (`str`) – str, parameter id
- **overwrite** (`bool`) – bool, indicates if an existing state/ODE should be overwritten
- **nominal_value** (`Optional[float]`) – float, nominal value of the parameter.
- **parameter_name** (`Optional[str]`) – str, name of parameter in PETA parameter table, optional.
- **parameter_scale** (`Optional[str]`) – str. scale of parameter in PETA parameter table, optional.
- **lower_bound** (`Optional[float]`) – float, lower bound of parameter in PETA parameter table, optional.
- **upper_bound** (`Optional[float]`) – float, upper bound of parameter in PETA parameter table, optional.
- **estimate** (`Optional[int]`) – int, estimate flag of parameter in PETA parameter table, optional.

delete_assignment(*assignment_id*)

Delete an assignment.

Raise a `ValueError`, if assignment does not exist.

delete_condition(*condition_id*)

Delete a condition.

Raise a `ValueError`, if condition does not exist.

delete_function(*function_id*)

Delete a function.

Raise a `ValueError`, if function does not exist.

delete_observable(*observable_id*)

Delete an observable.

Raise a `ValueError`, if observable does not exist.

delete_ode(*state_id*)

Delete a state + ODE.

Raise a `ValueError`, if state does not exist.

delete_parameter(*parameter_id*)

Delete a parameter.

Raise a `ValueError`, if parameter does not exist.

delete_time()
Delete time variable.

get_assignment_by_id(*assignment_id*)
Return dict for corresponding assignment.
Raise a *ValueError*, if the assignment does not exist.

get_assignment_ids()
Return a list with all assignment ids.

get_condition_by_id(*condition_id*)
Return dict for corresponding condition.
Raise a *ValueError*, if the condition does not exist.

get_condition_ids()
Return a list with all conditions ids.

get_function_by_id(*function_id*)
Return dict for corresponding function.
Raise a *ValueError*, if the function does not exist.

get_function_ids()
Return a list with all function ids.

get_observable_by_id(*observable_id*)
Return dict for corresponding observable.
Raise a *ValueError*, if the observable does not exist.

get_observable_ids()
Return a list with all observable ids.

get_ode_by_id(*state_id*)
Return dict for corresponding ODE/state.
Raise a *ValueError*, if the ODE/state does not exist.

get_ode_ids()
Return a list with all state ids.

get_parameter_by_id(*parameter_id*)
Return dict for corresponding parameter.
Raise a *ValueError*, if the parameter does not exist.

get_parameter_ids()
Return a list with all parameter ids.

get_time()
Get time variable.

is_set_time()
Check whether there is a time variable.

static load_from_yaml(*yaml_dir*)
Create a model instance from a YAML file.

Parameters

yaml_dir (*str*) – directory to the YAML file, that should be imported

Returns

new model

Return type

cls

set_time(*time_variable*)

Set time variable.

validate_model()

Validate the YAML model.

Raises

ValidationError –

write_to_petab(*output_dir, model_name, petab_yaml_name=None, measurement_table_name=None*)

Write the YamlModel as a PETab problem.

Equivalent to calling `yaml2petab` on the file produced by the YAML output.

If a `petab_yaml_name` is given, a YAML file is created, that organizes the petab problem. If additionally a `measurement_table_file_name` is specified, this file name is written into the created YAML file.

Parameters

- **output_dir** (*str*) – path the output file(s) are be written out
- **model_name** (*str*) – name of SBML model
- **petab_yaml_name** (*Optional[str]*) – name of the YAML organizing the PETab problem.
- **measurement_table_name** (*Optional[str]*) – Name of measurement table

write_to_sbml(*sbml_dir, overwrite=False*)

Write the model as an SBML file to the directory given in *sbml_dir*.

Parameters

- **sbml_dir** (*str*) – path/file, where the sbml should be written
- **overwrite** (*bool*) – Indicates, whether an existing yaml should be overwritten

Raises

- **ValueError** –
- **FileExistsError** –

write_to_yaml(*yaml_dir, overwrite=False*)

Write the model to a YAML file given as *yaml_dir*.

Parameters

- **yaml_dir** (*str*) – path/file, where the YAML should be written
- **overwrite** (*bool*) – Indicates, whether an existing YAML should be overwritten

Raises

- **ValueError** –
- **FileExistsError** –

RELEASE NOTES

5.1 0.2 series

5.1.1 0.2.4 (2021-05-28)

- JOSS badge in the README (#133)
- Citation section in the README (#133)

5.1.2 0.2.3 (2021-05-25)

- Add additional names to the license (#128).
- Add information about the installation of dependencies for the example notebooks (#128).

5.1.3 0.2.2 (2021-03-19)

- Add checks in SBML conversion, e.g. to catch invalid identifiers and equations (#118, #123).
- Add tests for MacOS and Windows (#115).
- Fix pydocstyle and swig (#120).
- Add a logo license. (#116)
- Smaller fixes in notebooks (#113, #117).

5.1.4 0.2.1 (2021-02-22)

- New option to write observables as assignments in *yaml2sbml* (#105).
- Set SBML file name as model id in the SBML (#105).
- Clarify docs and warnings, e.g. if formula starts with a minus (#109).
- Add issue template (#96).
- Restructure README (#55, #103, #104).
- Testing via tox (#88, #94, #95).
- Replace *requirements.txt* and *setup.py* by *setup.cfg* (#87).

5.1.5 0.2.0 (2021-02-03)

- Initial release on *pypi*.

LICENSE

This package is licensed under an MIT license.

MIT License

Copyright (c) 2020 Jakob Vanhoefer, Marta R. A. Matos, Dilan Pathirana, Yannik Schaelte and Jan Hasenauer.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



yaml2sbml's logo can be found in multiple variants in the [doc/logo](#) folder, in svg and png format. It is made available under a Creative Commons CC0 1.0 Universal (CC0 1.0) license, with the terms given in [doc/logo/LICENSE.md](#). We encourage to use it e.g. in presentations and posters.

We thank Elba Raimúndez for her contributions to the logo design.

CONTACT

Feel free to submit an [issue](#) or send us an [e-mail](#).

CONTRIBUTE

9.1 Documentation

To make `yaml2sbml` easily usable, we try to provide good documentation, including code annotation and usage examples. The documentation is hosted on yaml2sbml.readthedocs.io and updated automatically every time the main branch is updated. To create the documentation locally, first install the requirements via:

```
pip install .[doc]
```

and then compile the documentation via:

```
cd doc
make html
```

9.2 Test environment

We use the virtual testing tool `tox` for all unit tests, format and quality checks and notebooks. Its configuration is specified in `tox.ini`. To run it locally, first install:

```
pip install tox
```

and then simply execute:

```
tox
```

To run only selected tests (see `tox.ini` for what is tested), use e.g.:

```
tox -e pyroma,flake8
```

For continuous integration testing we use GitHub Actions. All tests are run there on pull requests and required to pass. The configuration is specified in `.github/workflows/ci.yml`.

9.3 Unit tests

Unit tests are located in the `tests` folder. All files starting with `test_` contain tests and are automatically run on GitHub Actions. Run them locally via:

```
tox -e unittests
```

which boils mostly down to:

```
python3 -m pytest tests
```

You can also run only specific tests.

Unit tests can be written with `pytest` or `unittest`.

9.4 PEP8

We try to respect the `PEP8` coding standards. We run `flake8` as part of the tests. The `flake8` plugins used are specified in `tox.ini` and the `flake8` configuration given in `.flake8`. You can run it locally via:

```
tox -e flake8
```

9.5 Workflow

If you start working on a new feature or a fix, please create an issue on GitHub shortly describing the issue and assign yourself.

To get your code merged, please:

1. create a pull request to develop
2. if not already done in a commit message, use the pull request description to reference and automatically close the respective issue (see <https://help.github.com/articles/closing-issues-using-keywords/>)
3. check that all tests pass
4. check that the documentation is up-to-date
5. request a code review

New production releases should be created every time the `main` branch is updated.

10.1 Versions

On every merge to `main`, the version number in `yaml2sbml/version.py` should be incremented. We use version numbers `A.B.C`, where roughly

- `C` is increased for bug fixes,
- `B` is increased for new features and minor API breaking changes,
- `A` is increased for major API breaking changes.

as suggested by the [Python packaging guide](#).

10.2 Create a new release

After new commits have been added via pull requests to the `develop` branch, changes can be merged to `main` and a new version of `yaml2sbml` can be released.

10.2.1 Merge into main

1. create a pull request from `develop` to `main`,
2. check that all tests pass,
3. check that the documentation is up-to-date,
4. update the version number in `yaml2sbml/version.py` (see above),
5. update the release notes in `doc/release_notes.rst`,
6. request a code review.

To be able to actually perform the merge, sufficient rights may be required. Also, at least one review is required.

10.2.2 Create a release on GitHub

After merging into `main`, create a new release on GitHub. This can be done either directly on the project GitHub website, or via the CLI as described in [Git Basics - Tagging](#). In the release form,

- specify a tag with the new version as specified in `yaml2sbml/version.py`,
- include the latest additions to `doc/release_notes.rst` in the release description.

10.3 Upload to PyPI

The upload to the Python package index PyPI has been automated via GitHub Actions, specified in `.github/workflows/deploy.yml`, and is triggered whenever a new release tag is created.

To manually upload a new version to PyPI, proceed as follows: First, create a so-called “wheel” via:

```
python setup.py sdist bdist_wheel
```

A wheel is essentially a ZIP archive which contains the source code and the binaries (if any).

Then upload the archive:

```
twine upload dist/yaml2sbml-x.y.z-py3-non-any.wheel
```

replacing `x.y.z` by the respective version number.

See also the [section on distributing packages](#) of the Python packaging guide.

PYTHON MODULE INDEX

y

`yaml2sbml.yaml2sbml`, [27](#)

INDEX

A

`add_assignment()` (*yaml2sbml.YamlModel* method), 28
`add_condition()` (*yaml2sbml.YamlModel* method), 28
`add_function()` (*yaml2sbml.YamlModel* method), 29
`add_observable()` (*yaml2sbml.YamlModel* method), 29
`add_ode()` (*yaml2sbml.YamlModel* method), 29
`add_parameter()` (*yaml2sbml.YamlModel* method), 30

D

`delete_assignment()` (*yaml2sbml.YamlModel* method), 30
`delete_condition()` (*yaml2sbml.YamlModel* method), 30
`delete_function()` (*yaml2sbml.YamlModel* method), 30
`delete_observable()` (*yaml2sbml.YamlModel* method), 30
`delete_ode()` (*yaml2sbml.YamlModel* method), 30
`delete_parameter()` (*yaml2sbml.YamlModel* method), 30
`delete_time()` (*yaml2sbml.YamlModel* method), 30

G

`get_assignment_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_assignment_ids()` (*yaml2sbml.YamlModel* method), 31
`get_condition_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_condition_ids()` (*yaml2sbml.YamlModel* method), 31
`get_function_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_function_ids()` (*yaml2sbml.YamlModel* method), 31
`get_observable_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_observable_ids()` (*yaml2sbml.YamlModel* method), 31
`get_ode_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_ode_ids()` (*yaml2sbml.YamlModel* method), 31

`get_parameter_by_id()` (*yaml2sbml.YamlModel* method), 31
`get_parameter_ids()` (*yaml2sbml.YamlModel* method), 31
`get_time()` (*yaml2sbml.YamlModel* method), 31

I

`is_set_time()` (*yaml2sbml.YamlModel* method), 31

L

`load_from_yaml()` (*yaml2sbml.YamlModel* static method), 31

M

`main()` (in module *yaml2sbml.yaml2sbml*), 27
module
 yaml2sbml.yaml2sbml, 27

S

`set_time()` (*yaml2sbml.YamlModel* method), 32

V

`validate_model()` (*yaml2sbml.YamlModel* method), 32
`validate_petab_tables()` (in module *yaml2sbml*), 28
`validate_yaml()` (in module *yaml2sbml*), 27

W

`write_to_petab()` (*yaml2sbml.YamlModel* method), 32
`write_to_sbml()` (*yaml2sbml.YamlModel* method), 32
`write_to_yaml()` (*yaml2sbml.YamlModel* method), 32

Y

`yaml2petab()` (in module *yaml2sbml*), 28
`yaml2sbml()` (in module *yaml2sbml.yaml2sbml*), 27
yaml2sbml.yaml2sbml
 module, 27
YamlModel (class in *yaml2sbml*), 28