sbml2julia documentation

Release 0.1.1

Paul F. Lang

Dec 19, 2020

CONTENTS

SBML2Julia is a tool to for optimizing parameters of ordinary differential equation (ODE) models. *SBML2Julia* translates a model from SBML/PEtab format into Julia for Mathematical Programming (JuMP), performs the optimization task and returns the results.

Source code: https://github.com/paulflang/SBML2Julia

CHAPTER

ONE

OPTIMIZATION METHOD

SBML2Julia uses the optimization method presented in Scalable nonlinear programming framework for parameter estimation in dynamic biological system models. In brief, contrary to typical parameter optimization methods for ODE systems, *SBML2Julia* does not rely on simulation of the ODE system. Instead *SBML2Julia* uses an implicit Euler scheme to time-discretize an ODE system of n equations into m time steps. This transforms the ODE system into a system of n * (m - 1) algebraic equations with n * m variables. These n * m variables (or a subset thereof) can then be cast into an objective function. Per default, *SBML2Julia* uses a least square objective. *SBML2Julia* then uses interior-point optimization implemented in the Julia language to minimize the objective function constraint to the n * (m - 1) algebraic equations.

CHAPTER

TWO

INTERFACES

Optimization tasks can be performed from a Python API or a command line interface.

CHAPTER

THREE

CONTENTS

3.1 Installation

SBML2Julia depends on several Python and Julia packages. If you have Docker installed on your machine, the easiest way of installing these dependencies is to pull the latest SBML2Julia docker image from dockerhub and build a container.:

```
user@bash:/$ docker pull paulflang/sbml2julia:latest
user@bash:/$ docker run -it --mount type=bind,source=<my_host_dir>,target=/media␣
↪paulflang/sbml2julia:latest
```

To install the latest *SBML2Julia* release in the docker container, run:

```
user@bash:/$ python3 -m pip install sbml2julia
```

Alternatively, to install the latest *SBML2Julia* version from GitHub, run:

```
user@bash:/$ git clone https://github.com/paulflang/sbml2julia.git
user@bash:/$ python3 -m pip install sbml2julia
```

To check if the installation was succesful, run:

```
user@bash:/$ sbml2julia -h
```

If you do not want to use Docker, the *SBML2Julia* dependencies can be installed on Ubuntu machines as indicated in the Dockerfile. Once these dependencie are installed, *SBML2Julia* can be installed as above.

3.1.1 Optional installation of efficient HSL linear solvers

SBML2Julia uses the nonlinear optimization solver *Ipopt* as core optimization engine. Its performance critically relies on the efficiency of the linear solver used within *Ipopt*. If the estimation problem faces intractability, we recommend custom installation of efficient HSL linear solvers. Since HSL linear solvers run under a different license that SBML2Julia, we cannot distribute them with *SBML2Julia*. However, academics can request a free license for HSL linear solvers. Using these HSL linear solvers within *SBML2Julia* requires custom installation of Ipopt and its Julia interface.

3.2 Tutorial

This section contains tutorials for the Python API and command line interface.

3.2.1 Python API

The following tutorial illustrates how to use the *SBML2Julia* Python API.

Importing *SBML2Julia*

Run this command to import *SBML2Julia*:

```
>>> import sbml2julia
```

Specifying an optimization problem

SBML2Julia uses the PEtab format for specifying biological parameter estimation problems. PEtab is built around SBML and based on tab-separated values (TSV) files. Please visit the PEtab documentation and have a look at the PEtab examples for detailed instructions on how to specify an optimization problem in PEtab. If you also want to customise upper and lower boundaries for the model species, you can provide an additional species table (see species_Vinod_FEBS2015.tsv as an example).

SBML2Julia also contains the following optimization hyperparameters:

- **t_steps**: number of time-discretization steps. Default `None`.
- **n_starts**: number of multistarts. Default `1`.
- **infer_ic_from_sbml**: infer initial conditions which are not specified in the PEtab condition table from SBML. Default `False`.
- **optimizer_options**: optimization solver options. Default `{}`.
- **custom_code_dict**: dict with replaced code as keys and replacement code as values. Default `{}`.

The problem is then specified as:

```
>>> problem = sbml2julia.SBML2JuliaProblem('my_petab_promlem.yaml', t_steps=100, n_
↪starts=1, infer_ic_from_sbml=False, optimizer_options={}, custom_code_dict={})
```

Once the problem is specified, *sbml2julia* has transformed the problem to a julia JuMP model. The code for this model can be accessed via:

```
>>> code = problem.julia_code
```

or written to a file via:

```
>>> problem.write_jl_file(path='path_to_jl_file.jl')
```

If you want to change the optimization problem in a way that is not yet supported by *SBML2Julia*, you can manually modify the julia code and run the optimization in julia yourself. Alternatively, you can change problem.julia_code via:

```
>>> problem.insert_custom_code({'<replaced lines>': '<replacement lines>'})
```

Choosing an HSL linear solver

Optionally, the `optimizer_options` attribute can be used to specify the linear solver used within *Ipopt*. For example:

```
>>> problem.optimizer_options={'linear_solver': 'ma57'}
```

Running the optimization

The optimization can be run with:

```
>>> problem.optimize()
```

Please note that this may take a while.

Accessing the results

The results can be accessed via:

```
>>> results = problem.results
```

and written to TSV and Excel files with:

```
>>> problem.write_results(path='./tsv_results/')
>>> problem.write_results(path='results.xlsx')
```

Time courses for the optimal solution of condition `cond` and corresponding experimental datapoints can be plotted by:

```
>>> problem.plot_results(cond, path='path_to_plot.pdf', observables=[], size=(6, 5))
```

where the optional `observable` argument accepts a list of observables that shall be plotted (if emppty, all observables specified in PEtab are plotted). The optional `size` argument specifies the size of the figure.

If you want to update the nominal parameter values in your PEtab problem parameter table with the fitted values, run:

```
>>> problem.write_optimized_Parameter_table()
```

This will create a *post_fit_parameters.tsv* file in your PEtab problem directory. This can be useful to perform sensitivity analysis in other PEtab compatible optimization toolboxes such as pyPesto.

3.2.2 Command line interface

To excecute a parameter fitting problem from the command line interface (CLI) you need to specify your optimization problem in the PEtab format, which is built around SBML and TSV files. If you also want to customise upper and lower boundaries for model species, you can provide an additional species table (see species_Vinod_FEBS2015.tsv as an example).

The *sbml2julia* CLI allows you to specify the following optimization options:

- **-t**, **–t_steps**: number of time-discretization steps. Default `None`.
- **-n**, **–n_starts**: number of multistarts. Default `1`.
- **-i**, **–infer_ic_from_sbml**: infer initial conditions which are not specified in the PEtab condition table from SBML. Default `False`.

- **-o**, **–optimizer_options**: optimization solver options. Default `{}`.
- **-c**, **–custom_code_dict**: dict with replaced code as keys and replacement code as values. Default `{}`.
- **-d**, **–out_dir**: output directory for julia_code, results and plot. Default `'./results/'`.
- **-p**, **–plot_obs**: list of observables to be plotted. Default all, i.e. `[]`.

The problem is then specified and solved via:

```
user@bash:/$ sbml2julia optimize 'my_petab_promlem.yaml' -t 100 -n 1 -i 'False' -o {}␣
↪-c {} -d './results' -p '[]'
```

The results can be found in the output directory given to the `-d` argument.

Choosing an HSL linear solver

Optionally, the `optimizer_options` attribute can be used to specify the linear solver used within *Ipopt*. For example:

```
user@bash:/$ sbml2julia optimize 'my_petab_promlem.yaml' -t 100 -n 1 -i 'False' -o '
↪{linear_solver: ma57}' -c {} -d './results' -p '[]'
```

3.3 Examples

3.3.1 G2/M cell cycle transition

The SBML2Julia GitHub repository contains a version of the Vinod et Novak model of the G2/M cell cycle transition. The model contains 13 species, 13 simulated observables, 24 parameters and 4 experimental conditions.

Using the Python API

The *SBML2Julia* problem can be created using the Python API (and assuming that the current working directory is the *SBML2Julia* root directory) via:

```
>>> import sbml2julia

>>> problem = sbml2julia.SBML2JuliaProblem('examples/Vinod_FEBS2015/Vinod_FEBS2015.
↪yaml')
```

and solved by:

```
>>> problem.optimize()
```

The results are then available under `problem.results`, which returns a dictionary containing `'par_best'` (the best found parameter set), `'species'`, `'observables'`, `'fval'` (the negative log-likelihood) and `'chi2'` (chi2 values of residuals). For example, to access the best parameter set, type:

```
>>> problem.results['x_best']
                Name     par_0   par_best  par_best_to_par_0
0            kDpEnsa    0.0500   0.048840           0.976807
1              kPhGw    1.0000   0.955727           0.955727
2             kDpGw1    0.2500   0.236274           0.945095
```

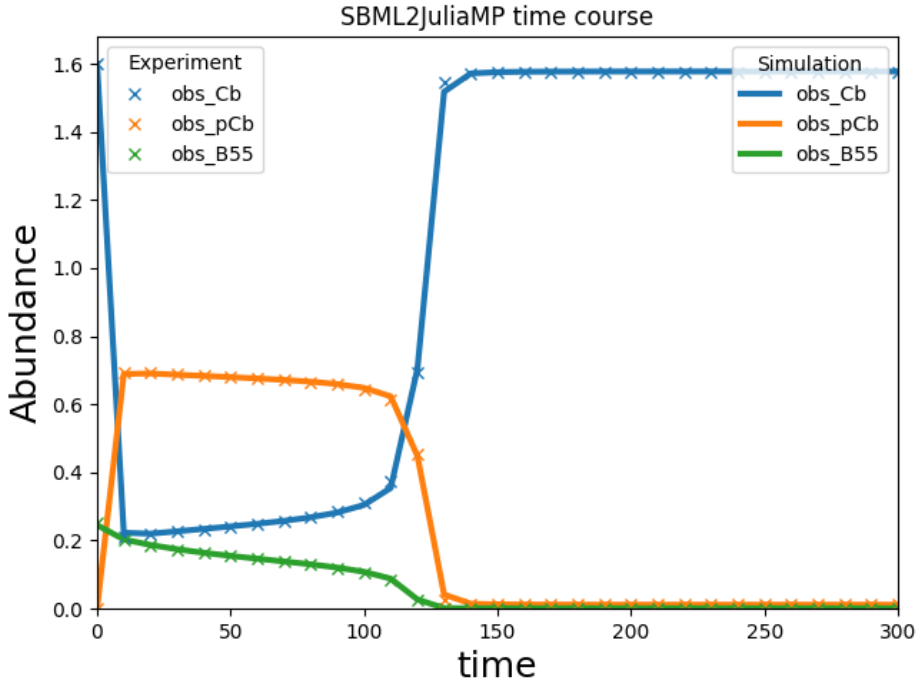
(continues on next page)

(continued from previous page)

```
3              kDpGw2  10.0000    9.727009           0.972701
4               kWee1   0.0100    0.009697           0.969738
5               kWee2   0.9900    1.308203           1.321418
6              kPhWee   1.0000    1.064760           1.064760
7              kDpWee  10.0000   10.181804           1.018180
8           kCdc25_1    0.1000    0.135812           1.358117
9           kCdc25_2    0.9000    1.099638           1.221820
10         kPhCdc25    1.0000    0.961436           0.961436
11         kDpCdc25   10.0000    8.990541           0.899054
12         kDipEB55    0.0068    0.003400           0.500002
13         kAspEB55   57.0000   46.386552           0.813799
14               fCb    2.0000    1.999269           0.999634
15             jiWee    0.1000    0.199999           1.999988
16          fB55_wt    1.0000    0.967920           0.967920
17        fB55_iWee    0.9000    0.886548           0.985053
18      fB55_Cb_low    1.1000    1.066191           0.969265
19     fB55_pGw_weak   1.0000    0.971742           0.971742
20       kPhEnsa_wt    0.1000    0.100393           1.003933
21      kPhEnsa_iWee   0.1000    0.097819           0.978187
22    kPhEnsa_Cb_low   0.1000    0.101325           1.013248
23  kPhEnsa_pGw_weak   0.0900    0.090801           1.008902
```

Selected observables of the optimized simulation of a given simulation condition can be plotted via:

```
>>> problem.plot_results(condition='wt', observables=['obs_Cb', 'obs_pCb', 'obs_B55'])
```



Using the command line interface

Similarly, the same example problem can be solved from the command line interface:

```
user@bash:/sbml2julia$ sbml2julia optimize 'examples/Vinod_FEBS2015/Vinod_FEBS2015.
↪yaml' -d 'examples/Vinod_FEBS2015/results/'
```

The results can be found in the output directory given to the `-d` argument.

3.3.2 Gut microbial community

The SBML2Julia GitHub repository contains a generalised Lotka-Volterra model of 12 gut bacteria. The model conceived by Shin et al. contains 2 species, 2 observables, 156 parameters and 210 experimental conditions in its PEtab formulation.

Using the Python API

The *SBML2Julia* problem can be created using the Python API (and assuming that the current working directory is the *sbml2julia* root directory) via:

```
>>> import sbml2julia

>>> problem = sbml2julia.SBML2JuliaProblem('examples/Shin_PLOS2019/Shin_PLOS2019.yaml
↪'})
```

and solved by:

```
>>> problem.optimize()
```

Again, the results are available under `problem.results`. Plots of the observables can be generated with the `problem.plot_results()` method and results can be written to TSV files with `problem.write_results()`.

Using the command line interface

Similarly, the same example problem can be solved from the command line interface:

```
user@bash:/sbml2julia$ sbml2julia optimize 'examples/Shin_PLOS2019/Shin_PLOS2019.yaml
↪' -d './examples/Shin_PLOS2019/results/'
```

The results can be found in the output directory given to the `-d` argument.

3.4 Known limitations

- **Local minima**: To avoid local minima, you can try to increase the number of starting point `n_starts`.
- **Stiff equations**: For some parameter sets, the model ODEs may be very stiff. The implicit Euler scheme used by *SBML2Julia* may encounter numerical errors. You can try increasing the number of discretization time steps by increasing `t_ratio` or reducing the parameter search window in the *Petab* parameter table.

3.5 API Reference

class sbml2julia.core.**SBML2JuliaProblem**(*petab_yaml*, *t_steps=None*, *n_starts=1*, *infer_ic_from_sbml=False*, *optimizer_options={}*, *custom_code_dict={}*)

Class to create and solve an optimization and retreive the results

property custom_code_dict
Get custom_code_dict

Returns custom code dict

Return type dict

import_julia_code(*file*)
Summary

Parameters **file** (*TYPE*) – Description

property infer_ic_from_sbml
Get infer_ic_from_sbml

Returns if missing initial conditions shall be infered from SBML model

Return type bool

insert_custom_code(*custom_code_dict*)
Inserts custom code into Julia code

Parameters **custom_code_dict** (dict) – dict with replaced code as keys and replacement code as values

property julia_code
Get julia_code

Returns julia code for optimization

Return type str

property n_starts
Get n_starts

Returns number of multistarts

Return type int

optimize()
Optimize SBML2JuliaProblem

Returns

Results in a dict with keys 'species', 'observables', 'parameters' and 'par_est'

Return type dict

property optimizer_options
Get optimizer_options

Returns optimization solver options

Return type dict

property petab_problem
Get petab_problem

Returns petab problem

Return type petab.problem.Problem

property petab_yaml_dict
Get petab_yaml_dict

Returns petab_yaml_dict

Return type dict

plot_results(*condition*, *path='./plot.pdf'*, *observables=[]*, *size=(6, 5)*)
Plot results

Parameters

- **condition** (str) – experimental condition to plot
- **path** (str, optional) – path to output plot
- **observables** (list, optional) – list of observables to be plotted
- **size** (tuple, optional) – size of image

Raises **ValueError** – if *observables* is not a list

property results
Get results

Returns optimization results

Return type dict

property t_steps
Get t_steps

Returns number of time-discretiation steps

Return type t_steps (int, optional)

write_jl_file(*path='./julia_code.jl'*)
Write code to julia file

Parameters **path** (str, optional) – path to output Julia file

write_optimized_parameter_table()
Writes a new parameter table were nominal values are replaced with optimized values

write_results(*path='./results'*, *df_format='long'*)
Write results to excel file

Parameters

- **path** (str, optional) – path of excel file to write results to
- **df_format** (str, optional) – long or wide table format

Raises **ValueError** – if *path* is not a directory or Excel file

3.6 Contributing to *SBML2Julia*

We welcome contributions to *SBML2Julia*, including to the software, tests and documentation. Please use GitHub pull requests to contribute to *SBML2Julia* or contact us by email.

1. Create a fork of the *SBML2Julia* Git repository. Please see the GitHub documentation for more information.
2. Edit the code, unit tests or documentation.
3. Commit your changes to your fork of the *SBML2Julia* repository.
4. Push your changes to GitHub.
5. Use the GitHub website to create a pull request for your changes. Please see the GitHub documentation for more information.

3.7 About

3.7.1 License

The software is released under the MIT license

3.7.2 Development team

This package was developed by the Paul F. Lang at the University of Oxford, UK and Sungho Shin at the University of Wisconsin-Madison, USA.

3.7.3 Acknowledgements

We would like to thank Frank T. Bergmann for sharing his code to parse SBML files. This work was supported by a the Engineering and Physical Sciences Research Council [grant number 2105279].

3.7.4 Questions and comments

Please contact Paul F. Lang with any questions or comments.

INDEX