AMICI Documentation

Release 0.16.0

The AMICI developers

Jan 25, 2023

ABOUT

Version: 0.16.0
Source code: https://github.com/AMICI-dev/amici

CHAPTER

ONE

ABOUT AMICI

AMICI provides a multi-language (Python, C++, Matlab) interface to the *SUNDIALS* solvers *CVODES* (for *ODE*s) and *IDAS* (for *DAE*s). AMICI allows the user to read differential equation models specified as *SBML* or *PySB* and automatically compiles such models into Python modules, C++ libraries or *.mex* simulation files (Matlab).

In contrast to the (no longer maintained) sundialsTB Matlab interface, all necessary functions are transformed into native C++ code, which allows for a significantly faster simulation.

Beyond forward integration, the compiled simulation file also allows for forward sensitivity analysis, steady state sensitivity analysis and adjoint sensitivity analysis for likelihood-based output functions.

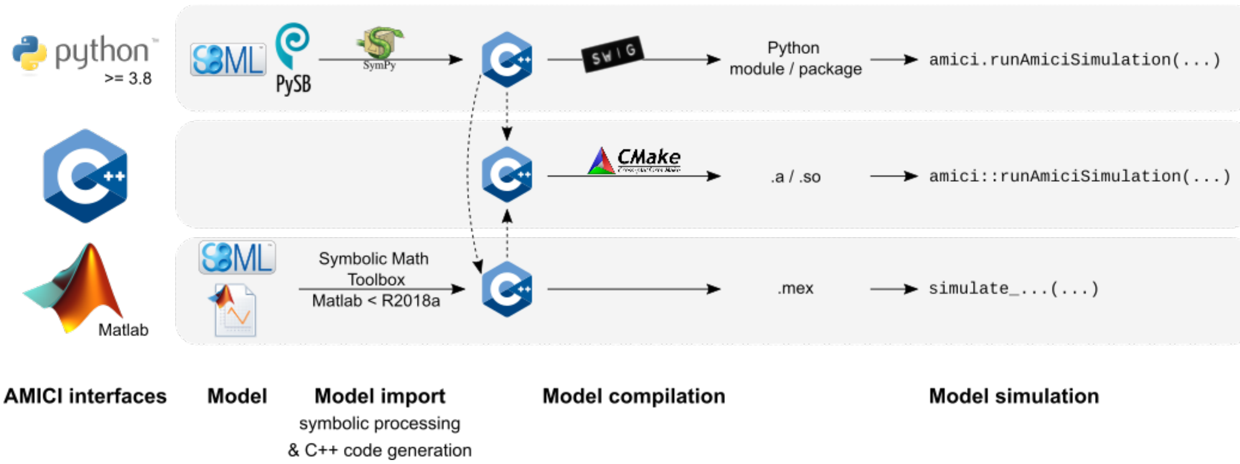
The interface was designed to provide routines for efficient gradient computation in parameter estimation of biochemical reaction models but it is also applicable to a wider range of differential equation constrained optimization problems.

1.1 Features

- *SBML* import
- *PySB* import
- Generation of C++ code for model simulation and sensitivity computation
- Access to and high customizability of *CVODES* and *IDAS* solver
- Python, C++, Matlab interface
- Sensitivity analysis
  - forward
  - steady state
  - adjoint
  - first- and second-order (second-order Matlab-only)
- Pre-equilibration and pre-simulation conditions
- Support for discrete events and logical operations

1.2 Interfaces & workflow

The AMICI workflow starts with importing a model from either *SBML* (Matlab, Python), *PySB* (Python), or a Matlab definition of the model (Matlab-only). From this input, all equations for model simulation are derived symbolically and C++ code is generated. This code is then compiled into a C++ library, a Python module, or a Matlab *.mex* file and is then used for model simulation.



The functionality of the Python, Matlab and C++ interfaces slightly differ, as shown in the following table:

Feature \ Interface	Python	C++	Matlab
SBML import	yes (*details*)	no	yes (<=R2017b)
PySB import	yes	no	no
DAE import	no	no	yes
Forward sensitivities	yes	yes	yes
Adjoint sensitivities	yes	yes	yes
Steadystate sensitivities	yes	yes	yes
Second-order sensitivities	no	no	yes
Events	yes	yes	yes
preequilibration	yes	yes	yes
presimulation	yes	yes	no

CHAPTER

TWO

AVAILABILITY

2.1 Source code

The AMICI source code is available as

- tar archive
- zip archive
- Git repository on GitHub

If AMICI was downloaded as an archive, it needs to be unpacked. If AMICI was obtained via cloning the Git repository, no further unpacking is necessary.

2.1.1 Obtaining AMICI via the Git version control system

In order to always stay up-to-date with the latest AMICI versions, simply pull it from our Git repository and recompile it when a new version is available. For more information about Git, check out their website.

The Git repository can currently be found at https://github.com/AMICI-dev/AMICI and clone is done via:

```
git clone https://github.com/AMICI-dev/AMICI.git AMICI
```

2.2 Python package

A Python package is available on PyPI.

2.3 Installation instructions

Installation instructions are available for

- *Python*
- *C++*
- *Matlab*

CHAPTER THREE

LICENSE CONDITIONS

3.1 AMICI

AMICI is released under the 3-Clause BSD License (BSD-3-Clause) with the following terms:

3.2 AMICI Logo

The AMICI logo is released under the Creative Commons CC0 1.0 Universal (CC0 1.0) license with the terms given in `documentation/gfx/LICENSE.md`.

3.3 Dependencies

- Parts of the *SUNDIALS* solver suite are redistributed under the BSD 3-Clause License (BSD-3-Clause) with terms given in `ThirdParty/SuiteSparse/LICENSE.txt`
- Parts of *SuiteSparse* are redistributed under the various licenses with the terms given in `ThirdParty/SuiteSparse/LICENSE.txt`
- *gsl-lite* is redistributed under the MIT License (MIT) with the terms given in `ThirdParty/gsl/gsl/gsl-lite.hpp`
- *xml2struct* and *struct2xml* are redistributed under the BSD 2-Clause License (BSD-2-Clause) with terms given in `matlab/auxiliary/xml2struct/license.txt` and `matlab/auxiliary/struct2xml/license.txt`
- *CalcMD5* is redistributed under the BSD 2-Clause License (BSD-2-Clause) with terms given in `matlab/auxiliary/CalcMD5/license.txt`

CHAPTER FOUR

HOW TO CITE AMICI

Citable DOI for the latest AMICI release:

There is a list of publications using AMICI. If you used AMICI in your work, we are happy to include your project, please let us know via a Github issue.

When using AMICI in your project, please cite

- Fröhlich, F., Weindl, D., Schälte, Y., Pathirana, D., Paszkowski, Ł., Lines, G.T., Stapor, P. and Hasenauer, J., 2021. AMICI: High-Performance Sensitivity Analysis for Large Ordinary Differential Equation Models. Bioinformatics, btab227, DOI:10.1093/bioinformatics/btab227.

```
@article{frohlich2020amici,
  title={AMICI: High-Performance Sensitivity Analysis for Large Ordinary Differential Equation Models},
  author={Fr{\"o}hlich, Fabian and Weindl, Daniel and Sch{\"a}lte, Yannik and Pathirana, Dilan and Paszkowski, {\L}ukasz and Lines, Glenn Terje and Stapor, Paul and Hasenauer, Jan},
  journal = {Bioinformatics},
  year = {2021},
  month = {04},
  issn = {1367-4803},
  doi = {10.1093/bioinformatics/btab227},
  note = {btab227},
  eprint = {https://academic.oup.com/bioinformatics/advance-article-pdf/doi/10.1093/bioinformatics/btab227/36866220/btab227.pdf},
}
```

When presenting work that employs AMICI, feel free to use one of the icons in documentation/gfx/, which are available under a CC0 license:

CHAPTER FIVE

REFERENCES

List of publications using AMICI. Total number is 74.

If you applied AMICI in your work and your publication is missing, please let us know via a new Github issue.

CHAPTER

SIX

BACKGROUND

This section is to be extended.

6.1 Publications on various features of AMICI

Some mathematical background for AMICI is provided in the following publications:

- Fröhlich, F., Kaltenbacher, B., Theis, F. J., & Hasenauer, J. (2017). Scalable Parameter Estimation for Genome-Scale Biochemical Reaction Networks. PLOS Computational Biology, 13(1), e1005331. doi:10.1371/journal.pcbi.1005331.
- Fröhlich, F., Theis, F. J., Rädler, J. O., & Hasenauer, J. (2017). Parameter estimation for dynamical systems with discrete events and logical operations. Bioinformatics, 33(7), 1049-1056. doi:10.1093/bioinformatics/btw764.
- Terje Lines, Glenn, Łukasz Paszkowski, Leonard Schmiester, Daniel Weindl, Paul Stapor, and Jan Hasenauer. 2019. "Efficient Computation of Steady States in Large-Scale Ode Models of Biochemical Reaction Networks. *IFAC-PapersOnLine* 52 (26): 32–37. DOI: 10.1016/j.ifacol.2019.12.232.
- Stapor, Paul, Fabian Fröhlich, and Jan Hasenauer. 2018. "Optimization and Profile Calculation of ODE Models Using Second Order Adjoint Sensitivity Analysis." *Bioinformatics* 34 (13): i151–i159. DOI: 10.1093/bioinformatics/bty230.
- Lakrisenko, Polina, Paul Stapor, Stephan Grein, Łukasz Paszkowski, Dilan Pathirana, Fabian Fröhlich, Glenn Terje Lines, Daniel Weindl, and Jan Hasenauer. 2022. "Efficient Computation of Adjoint Sensitivities at Steady-State in ODE Models of Biochemical Reaction Networks." *bioRxiv* 2022.08.08.503176. DOI: 10.1101/2022.08.08.503176.

Note: Implementation details of the latest AMICI versions may differ from the ones given in the references manuscripts.

6.2 Third-Party numerical algorithms used by AMICI

AMICI uses the following packages from SUNDIALS:

- CVODES:

  The sensitivity-enabled ODE solver in SUNDIALS. Radu Serban and Alan C. Hindmarsh. *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2005. PDF
- IDAS

AMICI uses the following packages from SuiteSparse:

- Algorithm 907: **KLU** A Direct Sparse Solver for Circuit Simulation Problems. Timothy A. Davis, Ekanathan Palamadai Natarajan, *ACM Transactions on Mathematical Software*, Vol 37, Issue 6, 2010, pp 36:1-36:17. PDF
- Algorithm 837: **AMD**, an approximate minimum degree ordering algorithm, Patrick R. Amestoy, Timothy A. Davis, Iain S. Duff, *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 381-388. PDF
- Algorithm 836: **COLAMD**, a column approximate minimum degree ordering algorithm, Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, Esmond G. Ng *ACM Transactions on Mathematical Software*, Vol 30, Issue 3, 2004, pp 377-380. PDF

Others:

- SuperLU_MT

  "A general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations" (https://crd-legacy.lbl.gov/~xiaoye/SuperLU/#superlu_mt). SuperLU_MT is optional and is so far only available from the C++ interface.

CHAPTER

SEVEN

CHANGELOG

7.1 v0.X Series

7.1.1 v0.16.0 (2023-01-25)

Features

- Python 3.11 compatibility by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1876
- AMICI now runs on binder (https://mybinder.org/v2/gh/AMICI-dev/AMICI/develop?labpath=binder%2Foverview.ipynb) by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1935, https://github.com/AMICI-dev/AMICI/pull/1937, https://github.com/AMICI-dev/AMICI/pull/1939
- More informative `Solver.__repr__` and `ExpData.__repr__` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1928 and @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1948
- `simulate_petab` returns the generated/used `ExpDatas` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1933
- Model module is now accessible from model instance by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1932
- Added `plot_jacobian` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1930
- Now logs all nested execution times as debug by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1947
- Always check for finite initial states, not only with `Model.setAlwaysCheckFinite(True)` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1955

Fixes

- `ReturnDataView.status` now returns `int` instead of `float` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1929
- Updated simulation status codes by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1931
- Skip irrelevant frames in stacktraces by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1934
- Fixed compiler warning (matlab) by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1954

Documentation:

- Added a notebook demonstrating common simulation failures and show how to analyze / fix them by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1946
- various minor fixes / updates

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.15.0…v0.16.0

7.1.2 v0.15.0 (2023-01-11)

Features

- Improved logging by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1907

  For Python: Don't print messages to stdout, but collect them in ReturnData and forward them to python logging, making it easier to filter specific messages or to disable output completely. Messages are also available via `ReturnData.messages`.

  breaking change for C++ interface: Messages aren't printed to stdout by default, but are collected in `ReturnData`. The user has to decide what to do with them.

- MultiArch docker build by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1903
- Added cmake target for cmake-format by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1909
- Updated clang-format style, fixed clang-format target by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1908
- Subsetting `ReturnData` fields by ID via `ReturnDataView.by_id` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1911 https://github.com/AMICI-dev/AMICI/pull/1916

Fixes

- PEtab import: fixed handling of fixed parameters for rule targets by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1915
- Fixed compiler warnings for matlab interface by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1919
- Fixed pandas DeprecationWarning for Series.iteritems() by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1921
- Fixed circular import in amici.petab_import_pysb by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1922
- Fixed 'operator ==' swig warning by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1923
- Prevent swig4.0.1 segfault by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1924

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.14.0…v0.15.0

7.1.3 v0.14.0 (2022-11-23)

Features:

- Added optional functionality to apply C99 math optimization to generated C++ code by @dweindl and @lcontento in https://github.com/AMICI-dev/AMICI/pull/1377, https://github.com/AMICI-dev/AMICI/pull/1878
- Added option to treat fixed parameters as constants in PEtab import

  by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1877

- Added equality operator for ExpData

  by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1881

- Updated base image for Dockerfile to Ubuntu 22.04/Python 3.10

  by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1896

Fixes:

- Fixed deprecation warnings by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1873, https://github.com/AMICI-dev/AMICI/pull/1893
- Fixes/updates to GitHub actions by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1885, https://github.com/AMICI-dev/AMICI/pull/1893, https://github.com/AMICI-dev/AMICI/pull/1889, https://github.com/AMICI-dev/AMICI/pull/1891
- Added hdf5 search directories for arm64 architecture (M1/M2 macs)

  by @Doresic in https://github.com/AMICI-dev/AMICI/pull/1894
- Fixed missing return in generated non-void functions

  by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1892
- Fixed import failure for pre-compiled models

  by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1897

Documentation:

- Update reference list by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1874, https://github.com/AMICI-dev/AMICI/pull/1884

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.13.0…v0.14.0

7.1.4 v0.13.0 (2022-10-04)

- Fixed extraction of common subexpressions by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1865
- Added function to convert `ReturnData::status` flags to string by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1864

And further contributions by @dweindl, @FFroehlich

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.12.0…v0.13.0

7.1.5 v0.12.0 (2022-08-26)

Features:

- Support for event observables via the Python interface by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1845
- Treat non-estimated parameters as constants during SBML-PEtab import by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1810
- Updated SUNDIALS to v5.8.0 by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1836
- Option to extract common subexpressions by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1852, https://github.com/AMICI-dev/AMICI/pull/1856 **not available in this release, use v0.13.0**
- Parallelize matrix simplification by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1778
- Validate PEtab problems before attempting import by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1842

- Improved type annotations for the swig interface by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1860

Fixes:

- Fixed an issue with potentially infinite loops during conservation law processing by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1833
- Fixed potential deadlocks during parallel simplification by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1844
- Fix resetting `ReturnData::numstepsB` when re-using Solver by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1841

And further contributions by @dilpath, @dweindl, @FFroehlich

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.32…v0.12.0

7.1.6 v0.11.32 (2022-07-15)

Fixes:

- Fixed `ImportErrors` during package installation with recent setuptools by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1830

7.1.7 v0.11.31 (2022-07-12)

Fixes:

- Fixed `ParameterMapping.__getitem__` to either return a `ParameterMappingForCondition` or a new `ParameterMapping`, but not a list by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1826

7.1.8 v0.11.30 (2022-07-07)

Features:

- Allow overriding model name during PySB import by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1801
- Added **repr** for parameter mapping classes by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1799
- More informative warning messages for NaNs/Infs by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1798
- Moved `sim_steps` increment by @plakrisenko in https://github.com/AMICI-dev/AMICI/pull/1806
- Re-arranged application of parameters from `ExpData` to avoid initial sensitivities error by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1805
- Checking for unused parameters in `simulate_petab` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1816
- Add `create_parameter_mapping` kwarg forwarding by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1820

Other

- Remove `constant_species_to_parameters` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1809

Fixes

- Fixed handling of SBML models given as `pathlib.Path` in `petab_import.import_model_sbml by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1808
- Fixed missing CPU time reset by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1814
- Raise in `simulate_petab` with `scaled_parameters=True` `problem_parameters=None` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1819

…

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.29…v0.11.30

7.1.9 v0.11.29 (2022-05-06)

7.2 What's Changed

Features:

- Performance: Limit newton step convergence check by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1780
- More informative NaN/Inf warnings by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1640
- SBML import can now handle initial events by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1789

Fixes:

- Avoid error if no measurements in PEtab problem; fixed type handling in PEtab parameter mapping by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1783
- Fixed substitution of expressions in root and stau by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1784
- Workaround for PEtab problems with state-dependent noise models by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1791

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.28…v0.11.29

7.2.1 v0.11.28 (2022-04-08)

New features:

- Added `Solver.setSteadyStateToleranceFactor` and `Solver.setSteadyStateSensiToleranceFactor` to specify a steady state tolerance factor by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1758

  NOTE: This also relaxed the default steady state (sensitivity) tolerances by a factor of 100.
- Added support for `pathlib.Path` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1769
- Allow specifying initial timepoint with `ExpData` by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1776

Performance:

- Speedup for models with conservation laws by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1765
- Improved efficiency of newton step convergence check by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1775

Fixes:

- Fixed deprecation warning for pandas.DataFrame.append in `rdatas_to_measurement_df` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1770
- Fixed Rule-target handling in PEtab import by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1753

Removed functionality:

- Removed long deprecated `sbml2amici` arguments `modelName` and `constantParameters` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1774

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.27…v0.11.28

7.2.2 v0.11.27 (2022-04-04)

New features:

- Checking condition number when computing sensitivities via Newton by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1730
- Removed SPBCG solver by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1729
- Added Newton step convergence checks to steadystate solver by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1737
- Removed legacy options/members `amioption.newton_preeq` and `Solver::r… by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1744
- Added `ReturnData::cpu_time_total` to track total time spent in `runAmiciSimulation` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1743
- SBML import: Alternative algorithm for identifying conservation laws by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1748
- Use `amici.AmiciVersionError` to indicate version mismatch by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1764

Performance:

- Optional parallel computation of derivatives during model import by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1740
- Sparsify jacobian by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1766
- Speedup conservation law computation by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1754
- Exploit stoichiometric matrix in pysb import by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1761
- Speedup edata construction from petab problems by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1746

Fixes:

- Fixed `get_model_settings` that would to setting incorrect initial states and initial state sensitivities for models with parameter-dependent initial states by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1751
- Use correct tolerances for convergence check in Newton solver by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1728
- Harmonized convergence checks by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1731
- Made sundials' KLU_INDEXTYPE match actual klu index type by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1733

- Fixed `Model::setStateIsNonNegative` logic that would raise exceptions in cases where it shouldn't by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1736
- Fixed undefined reference to dladdr by @kristianmeyerr in https://github.com/AMICI-dev/AMICI/pull/1738
- Fixed HDF5 OSX intermediate group creation errors by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1741
- Fixed recent cmake-based build issues due to changed sundials library directory by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1756
- Updated Windows installation instructions by @paulflang in https://github.com/AMICI-dev/AMICI/pull/1763

… and other contributions by @FFroehlich, @dweindl

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.26…v0.11.27

7.2.3 v0.11.26 (2022-03-14)

New features:

- Import of BioNetGenLanguage (BNGL) models by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1709
- Added support for observable-dependent sigmas by @dweindl, @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1692
- Added support for pysb local functions by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1666
- Added experimental support for conservation laws for non-constant species to SBML import: conservation laws for non-constant species by @stephanmg, @dweindl in https://github.com/AMICI-dev/AMICI/pull/1669 Enable this feature by setting environment variable `AMICI_EXPERIMENTAL_SBML_NONCONST_CLS` to any value
  - Allow using states eliminated by conservation laws to be used in root functions by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1677
  - Added support for parameter-dependent conservation laws by @dweindl, @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1678
- Added optional caching for symbolic simplifications in ODE export by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1672
- Added CLI option `--no-sensitivities` to `amici_import_petab` by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1688

Fixes:

- SBML import: Raise in case of nested observables by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1690
- Sympy 1.10 compatibility by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1694

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.25…v0.11.26

7.2.4 v0.11.25 (2022-02-09)

- Fixed a bug where `Model::setStateIsNonNegative(Model::getStateIsNonNegative())` would raise an exception in case conservation laws were enabled - by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1648
- Fixed a bug where `Model::setStateIsNonNegative` would be ignored in certain model expressions - by @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1650
- Fixed a bug where special function parsing inside `min()` and `max()` would not be parsed correctly - by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1655
- Fixed a numpy dependency issues for Mac+ARM systems - by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1657
- Fixed convergence check in Newton method - by @plakrisenko in https://github.com/AMICI-dev/AMICI/pull/1663
- Add `AMICI_CXX_OPTIONS` to pass libamici-specific compiler options during CMake-based builds - by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1664
- Fixed various warnings and updated documentation - by @dweindl

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.24…v0.11.25

7.2.5 v0.11.24 (2022-02-01)

Features:

- Introduced environment variable `AMICI_DLL_DIRS` to control DLL directories on Windows (useful for setting BLAS library directory, as required by Python>=3.8) by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1637
- Dropped Python3.7 support by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1635
- Include header files in CMake targets for better IDE integration by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1639

Fixes:

- Fixed an issue in PEtab import where all-integer parameters would previously result in a TypeError by @stephanmg in https://github.com/AMICI-dev/AMICI/pull/1634
- Fixed tempdir deletion issues for test suite on Windows by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1636
- Added functions to provide state IDs/names for x_solver by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1638
- Fixed docs on RTD by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1643

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.23…v0.11.24

7.2.6 v0.11.23 (2022-01-11)

Features:

- Added overload for Model::setParameterScale with vector by @dilpath in https://github.com/AMICI-dev/AMICI/pull/1614
- Removed assert_fun argument from gradient checking, improve output by @dweindl, @FFroehlich in https://github.com/AMICI-dev/AMICI/pull/1609
- Added get_expressions_as_dataframe by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1621
- Added `id` field to ExpData and ReturnData by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1622
- Included condition id in dataframes by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1623

Fixes:

- C++: Fixed SUNMatrixWrapper ctor for size 0 matrices by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1608
- Python: Handle TemporaryDirectory cleanup failures on Windows by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1617
- Python: pysb.Model.initial_conditions throws a DeprecationWarning by @PaulJonasJost in https://github.com/AMICI-dev/AMICI/pull/1620
- Fixed wrong array size in warnings by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1624

NOTE: AMICI 0.11.23 requires numpy<1.22.0

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.22…v0.11.23

7.2.7 v0.11.22 (2021-12-02)

- **Require sympy>=1.9,pysb>=1.13.1** by @FFroehlich, @dweindl in https://github.com/AMICI-dev/AMICI/pull/1599
- Fixed sympy deprecation warning by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1600
- Updated Windows installation instructions for Python>=3.8 by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1597
- Fixed plot labels by @dweindl in https://github.com/AMICI-dev/AMICI/pull/1598

Full Changelog: https://github.com/AMICI-dev/AMICI/compare/v0.11.21…v0.11.22

7.2.8 v0.11.21 (2021-11-21)

Fixes:

- Fixed a bug in recursion depth computation for model expressions. This may have resulted in incorrect sensitivities for models with expressions nested more than 2 levels. (#1595)
- Fixed improper handling of Piecewise functions in PySB import which may have produced incorrect simulation results. (#1594)
- Fixed changed googletest reference which broke the CMake-based build if tests were enabled (#1592)

New:

- It’s now possible to build AMICI using Ninja (#1593)

7.2.9 v0.11.20 (2021-11-12)

New:

- Changed parameter mappings such that unassigned values have non-nan default values. This fixes erroneous evaluation of `llh` as `NaN` in some situations (#1574)
- Added support for Python 3.10 (#1555)

Fixes:

- Fixed a bug when simulation start time was not transferred when copying a solver instance (#1573)
- Fixed a bug which led to incorrect sensitivities for models with multiple assignment rules or rate rules (#1584)

Other:

- Update CI and documentation settings (#1569, #1527, #1572, #1575, #1579, #1580, #1589, #1581)
- Extend set of validated benchmark models that is checked during CI (#1571, #1577)
- Fixed string formatting in derivative checks (#1585)
- Added helper methods to save and restore model instance-only settings (#1576)

7.2.10 v0.11.19 (2021-10-13)

New:

- Added support for observable transformations (lin/log/log10) (#1567). Thereby supporting additional noise distributions in combination with least squares solvers.

Fixes:

- Fixed a bug when Newton sensitivity computation was activated despite specifying newton_steps == 0. The error occurs when simulation converges to a steadystate but simulation sensitivities are not converged according to convergence criteria. In that case simulation returned failure, but the newton rootfinding "finds" a steadystate even before the iteration check, leading to the erroneous computation of sensitivities via Newton/IFT. For singular jacobians this means the overall simulation still fails, but a different, more informative error message is displayed. (#1541)
- Fixed a bug where argument "outdir" in ODEExporter.**init** would not be used (#1543)

Other:

- Improve checking support for SBML extensions (#1546)
- SBML import: Use more descriptive IDs for flux expressions (#1551)
- Optimized SUNMatrixWrapper functions (#1538)
- C++: Changed test suite from CppUTest to gtest (#1532)
- Add CITATION.cff (#1559)
- Updated documentation (#1563, #1554, #1536)
- Removed distutils dependency (#1557)
- Require sympy<1.9

7.2.11 v0.11.18 (2021-07-12)

New:

- Allow specifying maximum integration time via `amici::Solver::setMaxTime()` (#1530)
- Py: Add `failfast` and `num_threads` argument to `simulate_petab` (#1528, #1524)
- Enable typehints / static type checking for AMICI-generated model modules (#1514) (`amici.ModelModule`, available with Python>=3.8)

Fixes:

- Python: Fix unused argument `generate_sensitivity_code` in `pysb2amici` (#1526)
- Python: Fix C(++) stdout redirection which could have let to deadlocks in exotic situations (#1529)
- Py: Fixed deprecation warning (#1525)
- Py: Proper typehints for SWIG interface (#1523), enabling better static type checking and IDE integration (available with Python>=3.9)
- C++: Fixed clang compiler warning (#1521)
- C++: Fix inherited variadic ctor in exception class (#1520)
- PEtab: More informative output for unhandled species overrides (#1519)
- Return SbmlImporter from PEtab model import (#1517)

7.2.12 v0.11.17 (2021-05-30)

Fixes:

- Fix "maybe-uninitialized" compiler warning (#1495)
- Fix substitution of expressions in `drootdt_total (#1512)
- C++: Fix serialization and == operator (#1493)
- C++: Avoid `w` in `root` and `stau` headers (refactor) (#1503)

Documentation:

- Updated OpenBLAS Windows installation instructions (#1496)
- Updated how-to-cite to Bioinformatics paper (#1499)
- Updated list of papers using AMICI (#1509)

Other:

- Remove sllh computation from `petab_objective.simulate_petab` (#1498)
- Add **main**.py to Python package to provide info on AMICI installation via `python -m amici` (#1500)

7.2.13 v0.11.16 (2021-04-13)

Fixes:

- Fixed serialization bug (#1490)

New:

- Construction of condition specific plist for parameter mappings (#1487, #1488)
- **Add support for error residuals** (#1489)

7.2.14 v0.11.15 (2021-03-31)

Fixes:

- Fixed initial state sensitivities in adjoint preequilibration (#1468)
- Fixed various model import / parameter mapping issues (#1469, #1473, #1475)

New:

- New AMICI releases will automatically trigger releases at https://biosimulators.org/simulators/amici/latest
- Transparent logo

7.2.15 v0.11.14 (2021-03-16)

New features:

- **Python import now supports SBML Events** (#1443)
- Implement support for compilation without sensitivities (#1454)

Fixes:

- Issue #1446: Check whether constant parameters are valid targets (#1450)
- Issue #1422: Fix Steadystate solver failing if preequilibration starts in steadystate (#1461)
- Issue #1401: Ensure diagnostics variables in ReturnData are always of expected length (#1438, #1447)
- Fix FIM approximation for parameter dependent sigma (#1441)
- Fix invalid SBML in PEtab/PySB import (#1433)
- Fix: No context for inspect.getouterframes (#1432)

Documentation:

- Added this CHANGELOG
- Added feature request issue template (#1437)
- Updated reference list (#1430)
- Overhauled experimental conditions notebook (#1460)

CI:

- Test Matlab interface on GHA (#1451)
- Include componentTags in SBML test suite output (#1462)
- Split SBML semantic test suite into multiple jobs (#1452)

- Fix Crauste ref val, fixes #1458 (#1459)

Misc:

- Various cleanup (#1465, #1448, #1455)
- Micro-optimize SUNMatrixWrapper::transpose (#1439)
- Remove constant triggers from roots in Heaviside (#1440)

7.2.16 v0.11.13 (2021-02-20)

Breaking changes:

- AMICI requires Python>=3.7
- Updated package installation (PEP517/518): Creating source distributions requires https://github.com/pypa/build (#1384) (but now handles all package building dependencies properly)

Features:

- More flexible state reinitialization (#1417)

Updated dependencies:

- Upgrade to sundials 5.7.0 (#1392)

Fixes:

- Python: account for heaviside functions in expressions (#1382)
- Python: allow loading of existing models in import_petab_problem (#1383)
- Python: Don't override user-provided compiler/linker flags (#1389)
- Python: PEtab import reinitialization fixes (#1417)
- Python: Fix PEtab observables for pysb models (#1390)
- Python: Substitute expressions in event condition expressions (#1404)
- Python: Unspecified initial states in PEtab conditions table default to SBML initial value (#1397)
- C++: Fix timepoint out of bounds access (#1402)
- C++: Fix exported CMake config (#1388)
- Fixed Dockerfile: add python3-venv (#1398, #1408)

Other:

- Slim exported swig interface (#1425)
- Updated documentation
  - Getting started tutorial (#1423)
  - List supported SBML test tags (#1428)
  - Add AMICI C++/Python/Matlab feature comparison (#1409)
  - ...
- Various minor CI improvements
- ...

7.2.17 v0.11.12 (2021-01-26)

Features:

- Add expression IDs and names to generated models (#1374)

Fixes:

- Raise minimum sympy version to 1.7.1 (Closes #1367)
- Fix species assignment rules in reactions (#1372)
- Fix id vector for DAEs (#1376)

Docs:

- Update how-to-cite (#1378)

7.2.18 v0.11.11 (2020-12-15)

Python

- Restore support for species references (#1358)
- Add support for noise models in pysb (#1360)
- Proper handling of discontinuities in the ODE rhs (#1352)
- Fix directly calling AMICI from snakemake (#1348)
- Extend mathml function support, particularly for numerical arguments (#1357)
- Restore support for sympy 1.6 (#1356)

C++

- Fix some compiler related warnings (#1349, #1362)
- Fix a rare segfault for adjoint sensitivities (#1351)

CI

- Move windows tests to GHA (#1354)
- Pin breathe to 4.24.1

Docker

- Update ubuntu to 20.04

7.2.19 v0.11.10 (2020-11-30)

Bugfix release that restores compatibility with sympy 1.7

7.2.20 v0.11.9 (2020-11-29)

Python

- General improvements to SBML import (#1312, #1316, #1315, #1322 , #1324 #1333, #1329)
- Small bugfixes and improvements (#1321)
- Improve derivative computation for instances of `power` (#1320)

C++

- Fix FIM and residual computation for models with parameter dependent sigma. (#1338)
- Disable chi2/residual/FIM computation for non-gaussian objective functions. (#1338)
- Bugfix for integration failure during adjoint solve (#1327)

Doc

- Update references (#1331, #1336)

CI

- Update OpenBLAS for windows (#1334)

7.2.21 v0.11.8 (2020-10-21)

Python

- Fix pysb-petab support (#1288)
- Fix ExpData constructor overloading (#1299)
- Fix support for positivity enforcement (#1306)
- **Refactor SBML import, adds support for parameter rate rules and initial assignments** (#1284, #1296, #1304)
- Improve model generation for models with many parameters (#1300)
- Add support for PEtab based synthetic data generation (#1283)

C++

- Make HDF5 an optional dependency (#1285)

Doc

- General Improvements to Documentation (#1289, #1291, #1292, #1293, #1294, #1286, #1277, #1281)

CI

- Add python 3.9 support test (#1282)
- Allow manual triggering of GitHub actions (#1287)
- Remove appveyor config (#1295)
- Update GHA env and path management (#1302)

7.2.22 v0.11.7 (2020-09-22)

Python

- Improve and extend available objective functions (#1235)
- Fix processing of compartment definitions (#1223)
- Fix replacement of reserved symbols (#1265)
- Use Hierarchical Derivatives for Expressions (#1224, #1246)
- Fix duplicate running of swig (#1216)
- Overload python interface functions for amici.{Model,Solver,ExpData} and amici.{Model,Solver,ExpData}Ptr (#1271)

C++

- Fix and extend use of sparse matrix operations (#1230, #1240, #1244, #1247, #1271)
- **Fix application of maximal number of steps**, MaxNumStep parameter now limit total number of steps, not number of steps between output times. (#1267)

Doc

- Move all Documentation to RTD (#1229, #1241)
- General Improvements to Documentation (#1225, #1227, #1219, #1228, #1232, #1233, #1234, #1237, #1238, #1239, #1243, #1253, #1255, #1262)

CI

- Better check for doc building (#1226)
- Add more gradient checks (#1213)
- Update GHA to Ubuntu 20.04 (#1268)

7.2.23 v0.11.6 (2020-08-20)

Python

- Bugfix for piecewise functions (#1199)
- Refactor swigging - generate one single wrapper (#1213)

C++

- Fix warnings: account for zero indexing in nan/inf error (#1112)

Doc

- Update Windows build instructions (#1200, #1202)
- Update README: Projects using AMICI (#1209)
- Add CODE_OF_CONDUCT.md (#1210)
- Update documentation for Python interface (#1208)

CI

- Create sdist on GHA using swig4.0.1 (#1204) (Fixing broken pypi package)
- Fix links after repository move
- Speed-up swig build: disable all languages except python (#1211)
- Fix doc generation on readthedocs (#1196)

7.2.24 v0.11.5 (2020-08-07)

General

- Move repo to new organization (#1193)
- Update Bibliography

Python

- Fix bug for energyPySB models (#1191)

CI

- Fix release deployment (#1189)

7.2.25 v0.11.4 (2020-08-06)

Python

- Skip unnecessary expressions in pysb models (#1185)
- MSVC compiler support (this time for real… #1152)

CI

- Implement MSVC tests (#1152)
- Rename and group GitHub actions (#1186)
- Fix release deployment (#1186)

7.2.26 v0.11.3 (2020-08-06)

Python

- Fix simplification for pysb models (#1168)
- Pass verbosity flags to pysb network generation (#1173)
- Enable experimental pysb-petab support (#1175)
- Add installation instructions for Fedora (#1177)
- Implement support for SBML rate-references (#1180)

C++

- Refactoring (#1162, #1163)

CI

- Move majority of tests to Github Actions (#1166, #1160)
- Improve reporting of skipped tests in SBML testsuite (#1183)

7.2.27 v0.11.2 (2020-07-17)

Python

- Speed up model import, compilation (#1123, #1112)
- Improve/Add steady-state solver documentation (#1102)
- Improve extension import (#1141)
- Bugfixes SBML import (#1135, #1134, #1145, #1154)
- Fixed issue that prevented simplification (#1158)

C++

- Bugfixes (#1121, #1125, #1131, #1132, #1136)
- Enable openMP by default (#1118)
- Improve memoy footprint for simulations with replicates (#1153)
- Improve steady-state solver and add option to to adjoint-steadystate hybrid (#1143, #1099, #1129, #1146)

CI

- Store build artifacts from github actions (#1138)

7.2.28 v0.11.1 (2020-06-05)

Python

- Upgrade to sympy 1.6.0, which is now required minimum version (#1098, #1103)
- Speed up model import
  - Speed-up computation of sx0, reduce file size (#1109)
  - Replace terribly slow sympy.MutableDenseMatrix.is_zero_matrix by custom implementation (#1104)
- speedup dataframe creation in `get*AsDataFrame` (#1088)
- Allow caching edatas for simulate_petab (#1106)
- Fix wrong deprecation warning (Fixes #1093)
- Fix segmentation faults in NewtonSolver under certain conditions (#1089, #1090, #1097)
- fix wrong power function call in `unscale_parameter` (#1094)
- Fix MathML conversion (#1086)
- Fix deepcopy of SymPy objects (#1091)

Matlab

- handle empty rdata->{pre|post}eq_numlinsteps (Closes #1113), which previously made the matlab interface unusable
- Fix generation of compileMexFile.m for matlab compilation of python code (#1115)

C++

- Reduce memory requirements and speedup compilation of large models (#1105)
- Place generated code into own namespace (#937) (#1112)
- Fix several msvc compiler warnings (#1116) (Note that MSVC support is still experimental) **breaking change for users of C++ interface**
- Fix swig warning: ensure base class ContextManager is known before use (Fixes #1092) (#1101)

CI

- Don't install/run valgrind on travis CI (done with github actions… (#1111)

7.2.29 v0.11.0 (2020-05-10)

Python:

- **Implement support for variable compartments (#1036)**
- Better handling of constant species (#1047)
- **Better handling of C++ enums, this makes `amici.SensitivityMethod_forward` available as `amici.SensitivityMethod.forward` (#1042)**
- Improve installation routines (#1055, #1056, #1058, #1076)
- Add option to reduce memory usage (#1044)
- **Fix handling of symbolic expressions in nested rules (#1081, 1069)**

Library:

- Update Sundials to 5.2.0 (#1039)
- Update SuiteSparse to 5.4.0 (#1040)
- Refactor use of ReturnData, now completely created post-hoc (#1002)
- **Fix propagation of reinitialization in ExpData constructor (#1041)**
- **Fix issue where InternalSensitivityParameter was sometimes not set (#1075)**
- **Fix or disable certain combinations of equilibraition, presimulation and adjoint sensitivity analysis**

CI:

- Move from Codacy to Sonarcloud (#1065)
- Run SBML Testsuite when appropriate (#1058)

7.2.30 v0.10.21 (2020-04-04)

Library:

- Fix: Handle paths with blanks in build scripts
- Feature: Add function to write amici::Solver settings to HDF5 (#1023)
- Fix: typehints (#1018, #1022)
- Refactor: Move creation of parameter mapping for objective<->simulation to classes (#1020)

CI:

- Refactor: Cleanup and reorganize tests (#1026)
- Fix: benchmark problem test should fail on missing files (Closes #1015)

7.2.31 v0.10.20 (2020-03-18)

- Fixed (re)initialization of sensitivities if ExpData::fixedParametersPreequilibration is set (#994)
- Fixed sensitivities for parameters in sigma expressions for Python/SBML in case provided expression was not just a single parameter ID
- Enable parallel compilation of model files from Python (#997) based on AMICI_PARALLEL_COMPILE enviroment variable
- Fixed computation of log-likelihood for log10-normal distributed noise
- Added `reinitializeFixedParameterInitialStates` to ExpData (#1000) (**breaking change**: overrides settings in `amici::Model`)
- Python model import now verifies that chosen model name is a valid identifier (Closes #928)
- Made w available in ReturnData (Closes #990) (#992)
- Fixed setting of log level when passing boolean values to verbose (#991)
- Documentation now on ReadTheDocs https://amici.readthedocs.io/en/
- Use proper state/observable names in plotting functions (#979)
- PEtab support:
  - Adapt to most recent PEtab (0.1.5)
  - Extended support for import of PEtab models
  - Added support for computing cost function based on PEtab problem
  - Implemented handling of species in condition table
  - petab_import.import_model now provides reproducible parameter list (Closes #976)
  - Fix python import error in import_petab_problem: Add absolute paths to python path, invalidate caches and reload (#970)
  - Added example notebook
- CI: PEtab test suite integrated in CI workflow
- Added AMICI dockerfile and image deployment to dockerhub (#948)
- Removed mention of 'mex' in warning/error ids (#968)
- More informative errors on SWIG interface import failures (#959)

7.2.32 v0.10.19 (2020-02-13)

Python:

- Fix logo display on pypi
- Fix deadlocks in multithreaded python environments when using openMP parallelization

Matlab:

- Fix compilation errors due to switch to C++14

7.2.33 v0.10.18 (2020-02-11)

General:

- AMICI now comes with a logo
- implement getName function for models
- Updated documentation / examples

Python:

- Enable MSVC compilation of Python extensions (#847)
- Always recompile clibs and extensions (Closes #700)
- Extended PEtab support (Running
- enable multithreading in swig (#938)
- Fixes pysb (#902) (#907)

C++

- Build optimized AMICI and sundials by default (Closes #934)

Matlab:

- Fix(matlab) Compile CalcMD5 on demand (Fixes #914)
- Don't pass empty include strings to mex
- Fix Matlab compilation error if AMICI or model path contains blanks

CI:

- Running additional test models

… and various minor fixes/updates

7.2.34 v0.10.17 (2020-01-15)

- **added python 3.8 support, dropped python 3.6 support** (#898)
- Added logging functionality (#900)
- Fixes PySB import (#879, #902)
- Fixes symbolic processing (#899)
- Improved build scripts (#894,
- Improved petab support (#886, #888, #891)

- CI related fixes (#865, #896)

7.2.35 v0.10.16 (2019-12-11)

- **Sparsify dwdp to reduce computation time for adjoints (#858)**
- Fix(matlab) update example name example_dae_events->example_calvetti (Closes #866)
- Fix nullptr deferencing for simulations with events when no measurements are provided (Fixes #866)
- Fix accessing empty vector during adjoint state event update (Closes #866)
- Fix pysb_import (fixes #878)

7.2.36 v0.10.15 (2019-12-03)

Bugfix release due to incorrect sensitivities w.r.t. sigmas introduced in 0.10.14.

No other changes.

7.2.37 v0.10.14 (2019-12-02)

NOTE: For Python-imported SBML-models this release may compute incorrect sensitivities w.r.t. sigma. Bug introduced in 0.10.14, fixed in 0.10.15.

Python:

- Don't require use of ModelPtr.get to call ExpData(Model)
- Fix import in generated model Python package
- Setup AMICI standalone scripts as setuptools entrypoints
- Simplify symbolic sensitivity expressions during Python SBML import Fixes Infs in the Jacobian when using Hill-functions with states of 0.0.
- Extended Newton solver #848The changes that allow performing Newton tests from the paper:G. T. Lines, Ł. Paszkowski, L. Schmiester, D. Weindl, P. Stapor, and J. Hasenauer. Efficient computation of steady states in large-scale ODE models of biochemical reaction networks. accepted for Proceedings of the 8th IFAC Conference on Foundations of Systems Biology in Engineering (FOSBE), Valencia, Spain, October 2019.
- Use SWIG>=4.0 on travis to include PyDoc in sdist / pypi package (#841)
- **Fix choice of likelihood formula; failed if observable names were not equal to observable IDs**
- Fix(sbml-import) Compartment IDs in right-hand side of Rules are not replaced and lead to undefined identifiers in c++ files
- Fix invalid logging level
- Speed up sympy simplification (#871)

C++:

- Performance: Avoid unnecessary repeated function calls for SUNMatrixWrapper dimensions
- Add AmiciApplication class as context for handling so far global settings. This allows for example setting custom logging functions for concurrent AMICI runs, e.g. in multi-thread applications (Closes #576).

Misc:

- Setup performance test on github actions (#853)

- Update documentation and FAQ for CBLAS requirement and others
- Update reference list

7.2.38 v0.10.13 (2019-10-09)

- BREAKING CHANGE: Renaming {get|set}tNewtonPreequilibration to {get|set}Preequilibration (Closes #720)
- Make wurlitzer non-optional requirement for AMICI python package (Fixes missing AMICI errors when running from jupyter notebooks)
- Compute initial state for Model::getInitialStates if not already set (Fixes #818)
- Make swig generate pydoc comments from doxygen comments #830 (Closes #745) to provide Python docstrings for C++ wrapper functions
- feature(cmake) Add option to disable compiler optimizations for wrapfunctions.cpp (Fixes #828) (#829)
- Change SBML test suite to pytest to allow for parallel test execution… (#824)
- Fix(cmake): -E option is not available in all sed versions. Neither is the equivalent -r. Use –regexp-extended instead (Closes #826)
- Refactor(python) Move PEtab import code from command line script… (#825)
- Fix(core) Fix regular expressions for intel compiler (Closes #754) (#822)
- Update workflow figure to include PySB (Closes #799)
- Fix compiler warnings

7.2.39 v0.10.12 (2019-09-28)

- Fix handling of species specified in PEtab condition table (#813)
- Fix some Visual C++ issues, update cppcheck handling, cleanup (VisualC++ still not fully supported)
- Minor fixups (#801)
- Create SBML test suite result files for upload to http://sbml.org/Facilities/Database/ (#798)

7.2.40 v0.10.11 (2019-08-31)

- Fixed setting initial conditions for preequilibration (#784)
- Fixed species->parameter conversion during PEtab import (#782)
- Set correct Matlab include directories in CMake (#793)
- Extended and updated documentation (#785, #787)
- Fix various SBML import issues
- Run SBML test suite using github actions instead of travisCI (#789)

7.2.41 v0.10.10 (2019-08-07)

- Simplify/fix AMICI installation
  - If available use environment modules to detect dependencies
  - Add SWIG installation script
- Update list of publication
- Update documentation
  - Update doc for SWIG build and custom SWIG location.
  - Add AMICI interface overview / workflow figure and show in README
  - Document environment variables for model/core compilation (Closes #737)
- Added handling of abs function, since there seem to be problems with case sensitivity (#713) Closes #770

Detaills: * cmake: Use package_ROOT environment variables * fix(cmake) Fix finding version.txt * cmake: Auto-detect loaded MKL environment module * cmake: Use new FindPython3 modules where possible * fix(python) Restore python3.6 compatibility * Inside venv, use pip instead of pip3 which should point to the correct version * fix(python) Workaround for missing ensurepip during venv creation [ci skip] * feature(python) Use MKL from environment modules to provide cblas * fix(python) Fix define_macros not being passed to setuptools for Extension * fix(python) Fix define_macros not being passed to setuptools for clibs * Do not always add 'cblas' library since users may want to override that by a cblas-compatible library with a different name (closes #736)* Update HDF5 path hints; use shared library if static is not available. * Check for HDF5_BASE from environment module * Fix system-dependent sundials library directory (Fixes #749) (#750) * Handle OSTYPE==linux in scripts/buildBNGL.sh (Fixes #751) * Add SWIG download and build script * Improve finding swig executable and allow user override via SWIG environment variable * Provide installation hints if no SWIG found (Closes #724) * Allow overriding cmake executable with environment variables in build scripts (Closes #738)

7.2.42 v0.10.9 (2019-07-24)

Fixup for missing version bump in v0.10.8 release. No code changes compared to v0.10.8.

7.2.43 v0.10.8 (2019-07-24)

Changes in this release:

All:

- Updated / extended documentation
- Fix reuse of `Solver` instances (#541)

C++:

- Check for correct AMICI version for model in CMake
- Add reporting of computation times (#699)

Python:

- Fix manifest file (#698)
- Fix initial amounts/concentrations in SBML import

… and various other minor fixes/improvements

7.2.44 v0.10.7 (2019-05-01)

Python

- fix unset noise distribution when automatically generating observables in case None are passed (#691)

7.2.45 v0.10.6 (2019-04-19)

C++

- Add SuperLUMT support (#681)
- Sparsified dJydy (#686)
- Enabled support of impulse-free events for DAE models (#687) - thanks to Sebastien Sten for providing a testcase for this

Python

- Enabled support for piecewise functions in SBML import (#662)
- Fix numeric type when constructing ExpData from Dataframes (#690)
- Fix dynamic override in PETab

7.2.46 v0.10.5 (2019-04-08)

Bugfix release

Doc

- Update documentation of Windows installation

C++

- Fix missing source files in CMakeLists.txt (#658)
- Set CMake policies to prevent warnings (Closes #676) (#677)
- Start using gsl::span instead of raw pointers (#393) (#678)

Python

- PySB parsing fix (#669)
- Fix failure to propagate BLAS_LIBS contents (#665)
- Require setuptools at setup (#673)
- Updated PEtab import to allow for different noise models

7.2.47 v0.10.4 (2019-03-21)

Features / improvements:

- Implement ReturnData and ExpData wrappers as more efficient views (#657)
- Add list of references using AMICI (#659)
- Custom llh (normal/laplace, lin/log/log10) (#656)

Bugfixes:

- Speedup and fix travis build

7.2.48 v0.10.3 (2019-03-13)

Features / improvements:

- adds the option for early termination on integration failures for runAmiciSimulations
- improve runtime of `SUNMatrixWrapper::mutliply`
- expose finite difference routines in public API
- enable parallel compilation of clib source files

Bugfixes:

- fixed symbolic processing for unreleased pysb features

7.2.49 v0.10.2 (2019-03-07)

Features / improvements:

- extended `ExpData` interface to allow for condition specific parameters, parameter scales, parameter lists, initial conditions and initial condition sensitivities.

Bugfixes:

- fixed output values of `ReturnData::x_ss` and `ReturnData::sx_ss`

7.2.50 v0.10.1 (2019-03-04)

- travis-ci.com migration
- fix problem with has{variable} functions
- allow to import sbml model from string, not only file

7.2.51 v0.10.0 (2019-03-01)

Features / improvements:

- updated sundials to 4.1.0
- updated SuiteSparse to 5.4.0
- added generic implementations for symbolic expressions that were sparse matrix vector products

Bugfixes:

- fixed return value of `rz` when no data is provided.

7.2.52 v0.9.5 (2019-02-26)

Features / improvements:

- allow python installations without compilation of c++ extension
- improvements to ExpData <-> pandas.DataFrame interface
- allow generation of matlab models from python
- implement CLI interface for PEtab
- improve computation time for conservation laws from pysb import

Bugfixes:

- Fix sign in undamped Newton step.

Maintenance:

- use newer CI images

7.2.53 v0.9.4 (2019-02-11)

Minor fixes only:

- fix(core) Get solver diagnostics for first(last) timepoint (#588) (Closes #586)
- fix(ci) Fix autodeploy (Closes #589)

7.2.54 v0.9.3 (2019-02-07)

CRITICAL FIXES

- **fix(python) fix symbolic computations for adjoint (#583)**

Features

- feature(python) Check for matching AMICI versions when importing model (Closes #556). Set exact AMICI version as model package requirement.
- feature(core) Add option to rethrow AMICI exception (Closes #552)
- feature(python) Redirect C/C++ output in stdout is redirected (e.g. in ipython notebooks) (Closes #456)

Minor fixes

- fix(python) Fix doc and rename sys_pipes to something more meaningful
- fix(ci) Fix premature exit of scripts/runNotebook.sh
- fix(deploy) Update pyenv shims to find twine

7.2.55 v0.9.2 (2019-01-30)

Bugfixes:

- fixes a critical bug in the newton solver
- fixes multiple bugs in sbml import for degenerate models, empty stoichiometry assignments and conversion factors
- improved error messages for sbml import
- #560
- #557
- #559

7.2.56 v0.9.1 (2019-01-21)

Features / improvements:

- make pure steadystate results available as `rdata['x_ss']` and `rdata['sx_ss']`
- add option to specify integration tolerances for the adjoint problem via `atolB` and `rtolB`

Bugfixes:

- improved conservation law identification to also account for constant species.
- fixed a bug where simulation results were written into results of the second newton solver attempt
- fixed an openMP related warning

Maintenance:

- attempt to fix automatic deploy to pypi via travis

7.2.57 v0.9.0 (2019-01-18)

Features / improvements:

- Allow computation and application of conservation laws for pysb importet models. This enables use of Newton-Solver for preequilibration for models where it was previously not possible.
- Use `klu_refactor` in the sparse Newton solver instead of always using `klu_factor` and only perform symbolic factorization once (#421)
- Allow more detailed finiteness checks (#514)

Bugfixes:

- #491

Maintenance:

- Several improvements to travis log sizes and folding
- Use default copy constructor for Model by implementing class wrappers for sundials matrix types (#516)
- Reenable run of SBML testsuite

7.2.58 v0.8.2 (2019-01-07)

Features / improvements:

- Speedup symbolic processing for ODE generation in python

Bugfixes:

- Fix(python) Add missing deepcopy (introduced in 6847ba675f2088854db583199b8754aaa6e01576)
- Fix(core) Prevent parameter scaling length mismatch (#488)
- Fix(python) Set distutils dependency to current version to fix </usr/lib/python3.6/distutils/dist.py:261: UserWarning: Unknown distribution option: 'long_description_content_type'>
- fix(python) add symlink to version.txt to be included in package

Backwards-compatibility:

- Replace 'newline' by literal to restore <R2016b compatibility (Fixes #493)

Maintenance:

- Remove obsolete swig library build via cmake and related file copying
- Provide issue template for bug reports
- Providing valid SBML document to import is not optional anymore
- Update documentation and tests
- Add python version check and raise required version to 3.6 to prevent cryptic error messages when encountering f-strings

7.2.59 v0.8.1 (2018-11-25)

- [all] **critical** Fix long standing bugs in solving steadystate problems (including preequilibration) (#471)
- [all] Fix AmiVector constructor from std::vector (#471)
- [python] Reenable Solver and Model copy constructors
- Update documentation

7.2.60 v0.8.0 (2018-11-25)

- replaced symengine by sympy for symbolic processing in python *which fixes several critical bugs* that were due to bugs in symengine (#467)

7.2.61 v0.7.13 (2018-11-18)

- fixes a critical bug in objective function computation for models compiled using `sbml2amici` and `pysb2amici` that was introduced in v0.7.12
- fixes a critical bug in sensitivity computation when`model.reinitializeFixedParameterInitialStates` was set to true
- readds the python interface to the ExpData copy constructor that was inadvertently removed in 0.7.12 and streamlines the respective convenience wrapper to provide access to the full range of constructors.

7.2.62 v0.7.12 (2018-11-17)

- fixed a critical bug in `amici.constructEdataFromDataFrame`
- enabled multithreaded simulation of multiple experiments (requires compilation with openMP)
- modularized sbml import and added pysb import

7.2.63 v0.7.11 (2018-10-15)

- [python] Added numpy and python wrappers that provide a more user friendly python API
- [python] Enable import of SBML models with non-float assignment rules
- [python] Enable handling of exceptions in python
- [python] Enable nativ python access to std::vector data-structures
- [core] Provide an API for more fine-grained control over sensitivity tolerances and steady-state tolerances
- [core] Provide an API to specify non-negative state variables (this feature is still preliminary)

7.2.64 v0.7.10 (2018-08-29)

- Fixed python SBML import `log()` issues (#412)

7.2.65 v0.7.9 (2018-08-24)

- fixes MATLAB compilation of models
- adds option to perform steady state sensitivity analysis via FSA
- condition dependent intitial conditions are now newly set after preequilibration is done

7.2.66 v0.7.8 (2018-08-19)

- bugfixes for the ExpData interface
- created build configuration that enables debugging of c++ extensions on os x
- fixed python sbml import when stoichiometry is empty

7.2.67 v0.7.7 (2018-08-17)

Fixes a couple of bugs just introduced in v0.7.6

7.2.68 v0.7.6 (2018-08-13)

Important: **Use AMICI v0.7.7 due to https://github.com/ICB-DCM/AMICI/pull/403/commits/3a495d3db2fdbba70c2b0d52a3d465**

Bug fixes:

- Python import: Fix log10 issues in observables (#382)
- Matlab: Fix broken model compilation (#392)
- Fixed simulation for models without observables (#390)
- Fixed potential matlab memory leaks (#392)

Breaking C++ API changes:

- Revised ExpData interface (#388)

7.2.69 v0.7.5 (2018-07-30)

Features/enhancements:

- Add computation of residuals, residuals sensitivity, Fisher information matrix (#223)
- More efficient conversion of std::vector to numpy ndarray (#375)
- Allow specifying timepoints in ExpData (#370)

Minor fixes:

- Condition parameters in ExpData now only temporarily override Model parameters (#371)
- Ensure non-negative states for Newton solver (#378)

7.2.70 v0.7.4 (2018-07-27)

Features/enhancements:

- Check SBML model validity (#343)
- Allow per-parameter setting of amioptions::pscale from matlab interface (#350)
- Documentation

Major fixes:

- Don't compile main.cpp into python model module which broke modules if amici was compiled without libhdf5 (#363)

Minor fixes:

- Fix compiler warnings (#353)
- Plotting, SBML example mode, ...

7.2.71 v0.7.3 (2018-07-13)

Features:

- Added symbol names to python-wrapped models and make available via Model.getParameterNames(), model.getStateNames(), ...
- Extended Python interface example

Python package available via pypi: https://pypi.org/project/amici/0.7.3/

7.2.72 v0.7.2 (2018-07-03)

Features:

- Python package: more flexible HDF5 library localization
- Extended CI: python tests, preequilibration tests, run in venv

Major bugfixes:

- Fix python sbml model import / compilation error (undefined function)
- Fix model preequilibration

Minor fixes:

- Various fixes for mingw compilation of python source distribution
- Cmake compatibility with < 3.8 restored

7.2.73 v0.7.1 (2018-06-12)

Features:

- Allow specifying sigma-parameters from Python interface

Major bugfixes:

- Fix dsigma_y/dp and downstream sensitivity errors

7.2.74 v0.7.0 (2018-06-09)

- Major revision of documentation
- Improved Python interface
- More comprehensive Python interface example
- Fixed sensitivity computation in Python-generated models
- Various other bug fixes

WARNING:

- For models with sigma-parameters and dsigma_y/dp != 0, dsigma_y/dp was computed incorrectly. This propagates to all dependent sensitivities. This applies also to some older releases and has been fixed in v0.7.1.

7.2.75 v0.6.0 (2018-05-22)

Implement experimental support for python via swig. Python interface is now usable, but API will still receive some updates in the future.

WARNING:

- There is a bug in sensitivity computation for Python-generated models
- Matlab C++ compilation will fail due to undefined M_PI -> Please use v0.7.0

7.2.76 v0.5.0 (2018-03-15)

Main new features are:

- Reimplemented support for DAE equations
- Added newton solver for steady state calculation and preequilibration
- Better caching for recompilation of models
- Blas support to allow compilation of large models with many observables
- Improved SBML support
- Added c++ interface
- Added support for second order adjoint computation
- Rewrote large parts of the code as proper c++11 code to allow easier code maintanance
- Substantially extended testing in continuous integration to assure code quality

7.2.77 v0.4.0 (2017-05-15)

- First citable version of AMICI (via zenodo integration).
- Better support for standalone compilation
- Improved SBML import scripts
- General Code Cleanup

7.2.78 v0.3.0 (2016-09-05)

This update comes with many improvements, bug fixes and several new features. Most notably:

1. AMICI should now run on older versions of MATLAB (back to R2014a)
2. AMICI now compiles using older versions of Visual Studio
3. AMICI now also supports second order adjoint sensitivities (full (via the o2flag = 1 and as a vector product via o2flag = 2)
4. AMICI now supports more SBML, SBML v2 and rate rules

7.2.79 0.2.1 (2016-05-09)

Bugfix release. This release also includes some changes that should improve the performance on the new R2016a release of MATLAB.

7.2.80 v0.2 (2016-03-17)

This update comes with many improvements to the computation time for both compilation and simulation. Moreover several new features were included:

1. Hessian Vector products for second order forward sensitivities
2. Correct treatment of parameter/state dependent discontinuities for both forward and adjoint sensitivities

7.2.81 v0.1 (2015-11-05)

This is the initial release of the toolbox

CHAPTER

EIGHT

GLOSSARY

BNGL BioNetGenLanguage is a language for modular, structure-based modeling of biochemical reaction networks.

CVODES CVODES is a solver for stiff and non-stiff *ODE* systems with sensitivity analysis capabilities and is used by AMICI. It is part of the *SUNDIALS* solver suite.

DAE Differential-Algebraic Equation

fixed parameters In AMICI, *fixed parameters* are parameters with respect to which no sensitivities are computed. They usually correspond to experimental conditions. For fixed parameters, different values can be set for *preequilibration*, *presimulation* and simulation.

IDAS IDAS is a solver *DAE* systems with sensitivity analysis capabilities and is used by AMICI. It is part of the *SUNDIALS* solver suite.

ODE Ordinary Differential Equation

PEtab PEtab is a format for specifying parameter estimation problems. It is based on an *SBML* model and tab-separated value files specifying the observation model and experimental conditions.

preequilibration Simulating or solving the dynamical system for the steadystate.

presimulation Simulation for a fixed time before the regular simulation. Can be used to specify pretreatments.

PySB PySB is a tool for specifying rule-based systems biology models as Python code.

SBML The Systems Biology Markup Language is a commonly used format for specifying systems biology models.

SUNDIALS SUNDIALS: SUite of Nonlinear and DIfferential/ALgebraic equation Solvers. Provides the *CVODES* and *IDAS* solvers used by AMICI.

SWIG SWIG is a tool that creates interfaces for C(++) code to a variety of languages. Much of the AMICI Python interface is generated by SWIG.

CHAPTER

NINE

CONTRIBUTING

9.1 Contributing to AMICI

We are happy about contributions to AMICI in any form, be it new functionality, documentation, bug reports, or anything else.

If you would to contribute to AMICI, a good start is looking for issues tagged `good first issue` or `help wanted`. For other ideas or questions, just post an issue.

For code contributions, please read our developer's guide first.

9.2 Code of Conduct

9.2.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

9.2.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

9.2.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

9.2.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

9.2.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at froehlichfab@gmail.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

9.2.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq

CHAPTER

TEN

PYTHON INTERFACE

10.1 Installing the AMICI Python package

10.1.1 Short guide

Installation of the AMICI Python package has the following prerequisites:

- Python>=3.8
- *SWIG*>=3.0
- CBLAS compatible BLAS library (e.g., OpenBLAS, CBLAS, Atlas, Accelerate, Intel MKL)
- a C++14 compatible C++ compiler and a C compiler (e.g., g++, clang, Intel C++ compiler, mingw)

If these requirements are fulfilled and all relevant paths are setup properly, AMICI can be installed using:

```
pip3 install amici
```

If this worked, you can now import the Python module via:

```
import amici
```

If this does not work for you, please follow the full instructions below.

10.1.2 Installation on Linux

Ubuntu 20.04

Install the AMICI dependencies via `apt` (this requires superuser privileges):

```
sudo apt install libatlas-base-dev swig

# optionally for HDF5 support:
sudo apt install libhdf5-serial-dev
```

Install AMICI:

```
pip3 install amici
```

Fedora 32

Install the AMICI dependencies via apt (this requires superuser privileges):

```
sudo dnf install blas-devel swig
```

Install AMICI:

```
pip3 install amici
```

10.1.3 Installation on OSX

Install the AMICI dependencies using homebrew:

```
brew install swig

# optionally for HDF5 support:
brew install hdf5

# optionally for parallel simulations:
brew install libomp
```

Install AMICI:

```
pip3 install amici
```

10.1.4 Installation on Windows

Some general remarks:

- Install all libraries in a path not containing white spaces, e.g. directly under C:.
- Replace the following paths according to your installation.
- Slashes can be preferable to backslashes for some environment variables.
- See also [#425](https://github.com/AMICI-dev/amici/issues/425) for further discussion.

Using the MinGW compilers

- Install MinGW-W64 (the 32bit version will succeed to compile, but fail during linking).

  MinGW-W64 GCC-8.1.0 for `x86_64-posix-sjlj` (direct link) has been shown to work on Windows 7 and 10 test systems.
- Add the following directory to your `PATH`: `C:\mingw-w64\x86_64-8.1.0-posix-sjlj-rt_v6-rev0\mingw64\bin`
- Make sure that this is the compiler that is found by the system (e.g. `where gcc` in a `cmd` should point to this installation).
- Download CBLAS headers and libraries, e.g. OpenBLAS, binary distribution 0.2.19.

  Set the following environment variables:

  – `BLAS_CFLAGS=-IC:/OpenBLAS-v0.2.19-Win64-int32/include`

  - BLAS_LIBS=-Wl,-Bstatic -LC:/OpenBLAS-v0.2.19-Win64-int32/lib -lopenblas -Wl, -Bdynamic

- Install SWIG and add the SWIG directory to PATH (e.g. C:\swigwin-3.0.12 for version 3.0.12)
- Install AMICI using:

```
pip install --global-option="build_clib" \
            --global-option="--compiler=mingw32" \
            --global-option="build_ext" \
            --global-option="--compiler=mingw32" \
            amici --no-cache-dir --verbose`
```

Note: Possible sources of errors:

- On recent Windows versions, anaconda3\Lib\distutils\cygwinccompiler.py fails linking msvcr140.dll with [...] x86_64-w64-mingw32/bin/ld.exe: cannot find -lmsvcr140. This is not required for amici, so in cygwinccompiler.py return ['msvcr140'] can be changed to return [].
- If you use a python version where python/cpython#880 has not been fixed yet, you need to disable define hypot _hypot in anaconda3\include/pyconfig.h yourself.
- import amici in Python resulting in the very informative

  ImportError: DLL load failed: The specified module could not be found.

  means that some amici module dependencies were not found (not the AMICI module itself). DependencyWalker can show you which ones.

Using the Microsoft Visual Studio

Note: Support for MSVC is experimental.

We assume that Visual Studio (not to be confused with Visual Studio Code) is already installed. Using Visual Studio Installer, the following components need to be included:

- Microsoft Visual C++ (MSVC). This is part of multiple packages, including Desktop Development with C++.
- Windows Universal C Runtime. This is an individual component and installs some DLLs that we need.

OpenBLAS

There are prebuilt OpenBLAS binaries available, but they did not seem to work well here. Therefore, we recommend building OpenBLAS from scratch.

To build OpenBLAS, download the following scripts from the AMICI repository:

- https://github.com/AMICI-dev/AMICI/blob/master/scripts/installOpenBLAS.ps1
- https://github.com/AMICI-dev/AMICI/blob/master/scripts/compileBLAS.cmd

The first script needs to be called in Powershell, and it needs to call compileBLAS.cmd, so you will need to modify line 11:

  cmd /c "scriptscompileBLAS.cmd $version" so that it matches your directory structure. This will download OpenBLAS and compile it, creating

C:\BLAS\lib\openblas.lib C:\BLAS\bin\openblas.dll

You will also need to define two environment variables:

```
BLAS_LIBS="/LIBPATH:C:\BLAS\lib openblas.lib"
BLAS_CFLAGS="/IC:/BLAS/OpenBLAS-0.3.19/OpenBLAS-0.3.19"
```

One way to do that is to run a PowerShell script with the following commands:

```
[System.Environment]::SetEnvironmentVariable("BLAS_LIBS", "/LIBPATH:C:/BLAS/lib openblas.
↪lib", [System.EnvironmentVariableTarget]::User)
[System.Environment]::SetEnvironmentVariable("BLAS_LIBS", "/LIBPATH:C:/BLAS/lib openblas.
↪lib", [System.EnvironmentVariableTarget]::Process)
[System.Environment]::SetEnvironmentVariable("BLAS_CFLAGS", "-IC:/BLAS/OpenBLAS-0.3.19/
↪OpenBLAS-0.3.19", [System.EnvironmentVariableTarget]::User)
[System.Environment]::SetEnvironmentVariable("BLAS_CFLAGS", "-IC:/BLAS/OpenBLAS-0.3.19/
↪OpenBLAS-0.3.19", [System.EnvironmentVariableTarget]::Process)
```

The call ending in `Process` sets the environment variable in the current process, and it is no longer in effect in the next process. The call ending in `User` is permanent, and takes effect the next time the user logs on.

Now you need to make sure that all required DLLs are within the scope of the `PATH` variable. In particular, the following directories need to be included in `PATH`:

C:\BLAS\bin C:\Program Files (x86)\Windows Kits\10\Redist\ucrt\DLLs\x64

The first one is needed for `openblas.dll` and the second is needed for the Windows Universal C Runtime.

If any DLLs are missing in the `PATH` variable, Python will return the following error upon `import amici`:

ImportError: DLL load failed: The specified module could not be found.

Almost all of the DLLs are standard Windows DLLs and should be included in either Windows or Visual Studio. But, in case it is necessary to test this, here is a list of some DLLs required by AMICI (when compiled with MSVC):

- `openblas.dll`
- `python37.dll`
- `MSVCP140.dll`
- `KERNEL32.dll`
- `VCRUNTIME140_1.dll`
- `VCRUNTIME140.dll`
- `api-ms-win-crt-convert-l1-1-0.dll`
- `api-ms-win-crt-heap-l1-1-0.dll`
- `api-ms-win-crt-stdio-l1-1-0.dll`
- `api-ms-win-crt-string-l1-1-0.dll`
- `api-ms-win-crt-runtime-l1-1-0.dll`
- `api-ms-win-crt-time-l1-1-0.dll`
- `api-ms-win-crt-math-l1-1-0.dll`

MSVCP140.dll, VCRUNTIME140.dll, and VCRUNTIME140_1.dll are needed by MSVC (see Visual Studio above). KERNEL32.dll is part of Windows and in C:\Windows\System32. The api-ms-win-crt-XXX-l1-1-0.dll are needed by openblas.dll and are part of the Windows Universal C Runtime.

Note: Since Python 3.8, the library directory needs to be set either from Python:

```
import os
# directory containing `openblas.dll`
os.add_dll_directory("C:\\BLAS\\bin")
import amici
```

or via the environment variable AMICI_DLL_DIRS.

If Python returns the following error upon import amici, try updating to the latest Python version.

> OSError: [WinError 87] The parameter is incorrect: ‘’

10.1.5 Further topics

Installation of development versions

To install development versions which have not been released to PyPI yet, you can install AMICI with pip directly from GitHub using:

```
pip3 install -e git+https://github.com/AMICI-dev/amici.git@develop#egg=amici\&
↪subdirectory=python/sdist
```

Replace develop by the branch or commit you want to install.

Note that this will only work on Windows if you have enabled developer mode, because symlinks are not supported by default (more information).

Light installation

In case you only want to use the AMICI Python package for generating model code for use with Matlab or C++ and don't want to bothered with any unnecessary dependencies, you can run

```
pip3 install --install-option --no-clibs amici
```

Note: Following this installation, you will not be able to simulate the imported models in Python.

Note: If you run into an error with above installation command, install all AMICI dependencies listed in setup.py manually, and try again. (This is because pip --install-option is applied to *all* installed packages, including dependencies.)

Custom installation

Installation of the AMICI Python package can be customized using a number of environment variables:

Variable	Purpose	Example
SWIG	Path to the *SWIG* executable	SWIG=$HOME/bin/swig4.0
CC	Setting the C(++) compiler	CC=/usr/bin/g++
CFLAGS	Extra compiler flags used in every compiler invocation	
BLAS_CFLAGS	**Compiler flags for, e.g. BLAS** include directories	
BLAS_LIBS	Flags for linking BLAS	
ENABLE_GCOV_COVERAGE	Set to build AMICI to generate code coverage information	ENABLE_GCOV_COVERAGE=TRUE
ENABLE_AMICI_DEBUGGING	Set to build AMICI with debugging symbols	ENABLE_AMICI_DEBUGGING=TRUE
AMICI_PARALLEL_COMPILE	Set to the number of parallel processes to be used for C(++) compilation (defaults to 1)	AMICI_PARALLEL_COMPILE=4

Installation under Anaconda

To use an Anaconda installation of Python https://www.anaconda.com/distribution/, Python>=3.7), proceed as follows:

Since Anaconda provides own versions of some packages which might not work with AMICI (in particular the gcc compiler), create a minimal virtual environment via:

```
conda create --name ENV_NAME pip python
```

Here, replace ENV_NAME by some name for the environment.

To activate the environment, run:

```
source activate ENV_NAME
```

(and conda deactivate later to deactivate it again).

SWIG must be installed and available in your PATH, and a CBLAS-compatible BLAS must be available. You can also use conda to install the latter locally, using:

```
conda install -c conda-forge openblas
```

To make AMICI use openblas, set the following environment variable:

```
export BLAS_LIBS=-lopenblas
```

BLAS_LIBS needs to be set during installation of the AMICI package, as well as during any future model import.

To install AMICI, now run:

```
pip install amici
```

The pip option --no-cache may be helpful here to make sure the installation is done completely anew.

Now, you are ready to use AMICI in the virtual environment.

Note: Anaconda on Mac

If the above installation does not work for you, try installing AMICI via:

```
CFLAGS="-stdlib=libc++" CC=clang CXX=clang pip3 install --verbose amici
```

This will use the `clang` compiler.

You will have to pass the same options when compiling any model later on. This can be done by inserting the following code before model import:

```
import os
os.environ['CC'] = 'clang'
os.environ['CXX'] = 'clang'
os.environ['CFLAGS'] = '-stdlib=libc++'
```

(For further discussion see https://github.com/AMICI-dev/AMICI/issues/357)

Optional Boost support

Boost is an optional C++ dependency only required for special functions (including e.g. gamma derivatives) in the Python interface. Boost can be installed via package managers via

```
apt-get install libboost-math-dev
```

or

```
brew install boost
```

As only headers are required, also a source code download suffices. The compiler must be able to find the module in the search path.

10.2 Using AMICI's Python interface

In the following we will give a detailed overview how to specify models in Python and how to call the generated simulation files.

10.2.1 Model definition

This document provides an overview of different interfaces to import models in AMICI. Further examples are available in the AMICI repository in the python/examples directory.

SBML import

AMICI can import *SBML* models via the *amici.sbml_import.SbmlImporter()* class.

Status of SBML support in Python-AMICI

Python-AMICI currently **passes 1030 out of the 1821 (~57%) test cases** from the semantic SBML Test Suite (current status).

The following SBML test suite tags are currently supported (i.e., at least one test case with the respective test passes; tag descriptions):

Component tags:

- AssignmentRule
- Compartment
- CSymbolAvogadro
- CSymbolTime
- EventNoDelay
- FunctionDefinition
- InitialAssignment
- Parameter
- RateRule
- Reaction
- Species

Test tags:

- 0D-Compartment
- Amount
- AssignedConstantStoichiometry
- AssignedVariableStoichiometry
- BoolNumericSwap
- BoundaryCondition
- Concentration
- ConstantSpecies
- ConversionFactors
- DefaultValue
- EventT0Firing
- HasOnlySubstanceUnits
- InitialValueReassigned
- L3v2MathML
- LocalParameters

- MultiCompartment
- NoMathML
- NonConstantCompartment
- NonConstantParameter
- NonUnityCompartment
- NonUnityStoichiometry
- ReversibleReaction
- SpeciesReferenceInMath
- UncommonMathML
- VolumeConcentrationRates

In addition, we currently plan to add support for the following features (see corresponding issues for details and progress):

- Algebraic rules (#760)

However, the following features are unlikely to be supported:

- any SBML extensions
- *factorial()*, *ceil()*, *floor()*, due to incompatibility with symbolic sensitivity computations
- *delay()* due to missing *SUNDIALS* solver support
- events with delays, events with non-persistent triggers

Tutorials

A basic tutorial on how to import and simulate SBML models is available in the *Getting Started notebook*, while a more detailed example including customized import and sensitivity computation is available in the *Example Steadystate notebook*.

PySB import

AMICI can import *PySB* models via `amici.pysb_import.pysb2amici()`.

BNGL import

AMICI can import *BNGL* models via `amici.bngl_import.bngl2amici()`.

PEtab import

AMICI can import *PEtab*-based model definitions and run simulations for the specified simulations conditions. For usage, see *python/examples/example_petab/petab.ipynb*.

Importing plain ODEs

The AMICI Python interface does not currently support direct import of ODEs. However, it is straightforward to encode them as RateRules in an SBML model. The yaml2sbml package may come in handy, as it facilitates generating SBML models from a YAML-based specification of an ODE model. Besides the SBML model it can also create PEtab files.

SED-ML import

We also plan to implement support for the Simulation Experiment Description Markup Language (SED-ML).

10.2.2 Examples

Getting Started in AMICI

This notebook is a brief tutorial for new users that explains the first steps necessary for model simulation in AMICI, including pointers to documentation and more advanced notebooks.

Model Compilation

Before simulations can be run, the model must be imported and compiled. In this process, AMICI performs all symbolic manipulations that later enable scalable simulations and efficient sensitivity computation. The first towards model compilation is the creation of an SbmlImporter instance, which requires an SBML Document that specifies the model using the Systems Biology Markup Language (SBML).

For the purpose of this tutorial, we will use `model_steadystate_scaled.xml`, which is contained in the same directory as this notebook.

[1]:
```
import amici
sbml_importer = amici.SbmlImporter('model_steadystate_scaled.xml')
```

Next, we will compile the model as python extension using the amici.SBMLImporter.sbml2amici method. The first two arguments of this method are the name of the model, which will also be the name of the generated python module, and the model directory, which defines the directory in which the model module will be placed. Compilation will take a couple of seconds.

[2]:
```
model_name = 'model_steadystate'
model_dir = 'model_dir'
sbml_importer.sbml2amici(model_name, model_dir)
```

Loading the model module

To run simulations, we need to instantiate amici.Model and amici.Solver instances. As simulations requires instances matching the imported model, they have to be imported from the generated model module.

[3]:
```
# load the model module
model_module = amici.import_model_module(model_name, model_dir)
# instantiate model
model = model_module.getModel()
# instantiate solver
solver = model.getSolver()
```

The model allows the user to manipulate model related properties of simulations. This includes the values of model parameters that can be set by using amici.Model.setParameterByName. Here, we set the model parameter `p1` to a value of `1e-3`.

[4]:
```
model.setParameterByName('p1',1e-3)
```

In contrast, the solver instance allows the specification of simulation related properties. This includes setting options for the SUNDIALS solver such as absolute tolerances via amici.Solver.setAbsoluteTolerance. Here we set the absolute integration tolerances to `1e-10`.

[5]:
```
solver.setAbsoluteTolerance(1e-10)
```

Running Model Simulations

Model simulations can be executed using the amici.runAmiciSimulations routine. By default the model does not not contain any timepoints for which the model is to be simulated. Here we define a simulation timecourse with two timepoints at `0` and `1` and then run the simulation.

[6]:
```
# set timepoints
model.setTimepoints([0,1])
rdata = amici.runAmiciSimulation(model, solver)
```

Simulation results are returned as ReturnData instance. The simulated SBML species are stored as `x` attribute, where rows correspond to the different timepoints and columns correspond to different species.

[7]:
```
rdata.x
```

[7]:
```
array([[0.1       , 0.4       , 0.7       ],
       [0.98208413, 0.51167992, 0.10633388]])
```

All results attributes are always ordered according to the model. For species, this means that the columns of `rdata.x` match the ordering of species in the model, which can be accessed as amici.Model.getStateNames

[8]:
```
model.getStateNames()
```

[8]:
```
('x1', 'x2', 'x3')
```

This notebook only explains the basics of AMICI simulations. In general, AMICI simulations are highly customizable and can also be used to simulate sensitivities. The ExampleSteadystate notebook in this folder gives more detail about the model employed here and goes into the basics of sensitivity analysis. The ExampleEquilibrationLogic notebook, builds on this by using a modified version of this model to give detailed insights into the methods and options to compute steady states before and after simulations, as well as respective sensitivities. The ExampleExperimentalConditions example notebook, goes into the details of how even more complex experimental setups, such as addition of drugs at predefined timepoints, can be simulated in AMICI. Finally, the petab notebook explains how standardized definitions of experimental data and conditions in the PEtab format can be imported in AMICI.

AMICI Python example "steadystate"

This is an example using the [model_steadystate_scaled.sbml] model to demonstrate and test SBML import and AMICI Python interface.

[1]:
```
# SBML model we want to import
sbml_file = 'model_steadystate_scaled_without_observables.xml'
# Name of the model that will also be the name of the python module
model_name = 'model_steadystate_scaled'
# Directory to which the generated model code is written
model_output_dir = model_name

import libsbml
import importlib
import amici
import os
import sys
import numpy as np
import matplotlib.pyplot as plt
```

The example model

Here we use `libsbml` to show the reactions and species described by the model (this is independent of AMICI).

[2]:
```
sbml_reader = libsbml.SBMLReader()
sbml_doc = sbml_reader.readSBML(sbml_file)
sbml_model = sbml_doc.getModel()
dir(sbml_doc)

print('Species: ', [s.getId() for s in sbml_model.getListOfSpecies()])

print('\nReactions:')
for reaction in sbml_model.getListOfReactions():
    reactants = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfReactants()])
    products  = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfProducts()])
    reversible = '<' if reaction.getReversible() else ''
    print('%3s: %10s %1s->%10s\t\t[%s]' % (reaction.getId(),
                        reactants,
                        reversible,
                         products,
                        libsbml.formulaToL3String(reaction.getKineticLaw().getMath())))
```

```
Species:  ['x1', 'x2', 'x3']

Reactions:
 r1:       2 x1  ->        x2           [p1 * x1^2]
 r2:    x1 +  x2  ->       x3           [p2 * x1 * x2]
 r3:          x2  ->     2 x1           [p3 * x2]
 r4:          x3  ->  x1 +  x2          [p4 * x3]
 r5:          x3  ->                    [k0 * x3]
 r6:              ->       x1           [p5]
```

Importing an SBML model, compiling and generating an AMICI module

Before we can use AMICI to simulate our model, the SBML model needs to be translated to C++ code. This is done by `amici.SbmlImporter`.

[3]:
```
# Create an SbmlImporter instance for our SBML model
sbml_importer = amici.SbmlImporter(sbml_file)
```

In this example, we want to specify fixed parameters, observables and a σ parameter. Unfortunately, the latter two are not part of the SBML standard. However, they can be provided to `amici.SbmlImporter.sbml2amici` as demonstrated in the following.

Constant parameters

Constant parameters, i.e. parameters with respect to which no sensitivities are to be computed (these are often parameters specifying a certain experimental condition) are provided as a list of parameter names.

[4]:
```
constantParameters = ['k0']
```

Observables

Specifying observables is beyond the scope of SBML. Here we define them manually.

If you are looking for a more scalable way for defining observables, then checkout PEtab. Another possibility is using SBML's `` `AssignmentRules `` <https://sbml.org/software/libsbml/5.18.0/docs/formatted/python-api/classlibsbml_1_1_assignment_rule.html>`__ to specify model outputs within the SBML file.

[5]:
```
# Define observables
observables = {
    'observable_x1': {'name': '', 'formula': 'x1'},
    'observable_x2': {'name': '', 'formula': 'x2'},
    'observable_x3': {'name': '', 'formula': 'x3'},
    'observable_x1_scaled': {'name': '', 'formula': 'scaling_x1 * x1'},
    'observable_x2_offsetted': {'name': '', 'formula': 'offset_x2 + x2'},
    'observable_x1withsigma': {'name': '', 'formula': 'x1'}
}
```

σ parameters

To specify measurement noise as a parameter, we simply provide a dictionary with (preexisting) parameter names as keys and a list of observable names as values to indicate which sigma parameter is to be used for which observable.

[6]:
```
sigmas = {'observable_x1withsigma': 'observable_x1withsigma_sigma'}
```

Generating the module

Now we can generate the python module for our model. `amici.SbmlImporter.sbml2amici` will symbolically derive the sensitivity equations, generate C++ code for model simulation, and assemble the python module. Standard logging verbosity levels can be passed to this function to see timestamped progression during code generation.

[7]:
```
import logging
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         verbose=logging.INFO,
                         observables=observables,
                         constant_parameters=constantParameters,
                         sigmas=sigmas)
```

```
2021-11-30 16:57:57.997 - amici.sbml_import - INFO - Finished gathering local SBML␣
↪symbols        + (8.05E-03s)
2021-11-30 16:57:58.047 - amici.sbml_import - INFO - Finished processing SBML parameters␣
↪         + (4.64E-02s)
2021-11-30 16:57:58.054 - amici.sbml_import - INFO - Finished processing SBML␣
↪compartments        + (2.69E-04s)
2021-11-30 16:57:58.063 - amici.sbml_import - INFO - Finished processing SBML species␣
↪initials    ++ (1.47E-03s)
2021-11-30 16:57:58.067 - amici.sbml_import - INFO - Finished processing SBML rate rules␣
↪        ++ (7.65E-05s)
2021-11-30 16:57:58.068 - amici.sbml_import - INFO - Finished processing SBML species   ␣
↪         + (9.09E-03s)
2021-11-30 16:57:58.076 - amici.sbml_import - INFO - Finished processing SBML reactions ␣
↪         + (4.32E-03s)
2021-11-30 16:57:58.086 - amici.sbml_import - INFO - Finished processing SBML rules     ␣
↪         + (6.88E-03s)
2021-11-30 16:57:58.093 - amici.sbml_import - INFO - Finished processing SBML initial␣
↪assignments + (5.51E-05s)
2021-11-30 16:57:58.097 - amici.sbml_import - INFO - Finished processing SBML species␣
↪references  + (8.19E-04s)
2021-11-30 16:57:58.105 - amici.sbml_import - INFO - Finished processing SBML events    ␣
↪         + (1.00E-04s)
2021-11-30 16:57:58.105 - amici.sbml_import - INFO - Finished importing SBML            ␣
↪           (1.19E-01s)
2021-11-30 16:57:58.180 - amici.sbml_import - INFO - Finished processing SBML␣
↪observables           (6.99E-02s)
2021-11-30 16:57:58.202 - amici.ode_export - INFO - Finished running smart_multiply     ␣
↪         + (3.40E-03s)
2021-11-30 16:57:58.210 - amici.ode_export - INFO - Finished importing SbmlImporter     ␣
↪           (1.42E-02s)
2021-11-30 16:57:58.286 - amici.ode_export - INFO - Finished simplifying Jy             ␣
↪       +++ (6.10E-02s)
```

(continues on next page)

(continued from previous page)

```
2021-11-30 16:57:58.288 - amici.ode_export - INFO - Finished computing Jy                ␣
↪         ++ (6.69E-02s)
2021-11-30 16:57:58.299 - amici.ode_export - INFO - Finished simplifying y               ␣
↪        +++ (6.08E-04s)
2021-11-30 16:57:58.300 - amici.ode_export - INFO - Finished computing y                 ␣
↪         ++ (5.38E-03s)
2021-11-30 16:57:58.308 - amici.ode_export - INFO - Finished simplifying sigmay          ␣
↪        +++ (1.19E-04s)
2021-11-30 16:57:58.310 - amici.ode_export - INFO - Finished computing sigmay            ␣
↪         ++ (5.58E-03s)
2021-11-30 16:57:58.351 - amici.ode_export - INFO - Finished writing Jy.cpp              ␣
↪          + (1.33E-01s)
2021-11-30 16:57:58.421 - amici.ode_export - INFO - Finished running smart_jacobian      ␣
↪        +++ (5.81E-02s)
2021-11-30 16:57:58.452 - amici.ode_export - INFO - Finished simplifying dJydsigma       ␣
↪        +++ (2.69E-02s)
2021-11-30 16:57:58.453 - amici.ode_export - INFO - Finished computing dJydsigma         ␣
↪         ++ (9.43E-02s)
2021-11-30 16:57:58.466 - amici.ode_export - INFO - Finished writing dJydsigma.cpp       ␣
↪          + (1.09E-01s)
2021-11-30 16:57:58.505 - amici.ode_export - INFO - Finished running smart_jacobian      ␣
↪        +++ (2.48E-02s)
2021-11-30 16:57:58.559 - amici.ode_export - INFO - Finished simplifying dJydy           ␣
↪        +++ (4.86E-02s)
2021-11-30 16:57:58.560 - amici.ode_export - INFO - Finished computing dJydy             ␣
↪         ++ (8.27E-02s)
2021-11-30 16:57:58.586 - amici.ode_export - INFO - Finished writing dJydy.cpp           ␣
↪          + (1.12E-01s)
2021-11-30 16:57:58.599 - amici.ode_export - INFO - Finished simplifying root            ␣
↪        +++ (5.50E-05s)
2021-11-30 16:57:58.600 - amici.ode_export - INFO - Finished computing root              ␣
↪         ++ (4.63E-03s)
2021-11-30 16:57:58.601 - amici.ode_export - INFO - Finished writing root.cpp            ␣
↪          + (8.96E-03s)
2021-11-30 16:57:58.624 - amici.ode_export - INFO - Finished simplifying w               ␣
↪       ++++ (7.65E-03s)
2021-11-30 16:57:58.626 - amici.ode_export - INFO - Finished computing w                 ␣
↪        +++ (1.21E-02s)
2021-11-30 16:57:58.641 - amici.ode_export - INFO - Finished running smart_jacobian      ␣
↪        +++ (1.04E-02s)
2021-11-30 16:57:58.647 - amici.ode_export - INFO - Finished simplifying dwdp            ␣
↪        +++ (1.45E-03s)
2021-11-30 16:57:58.648 - amici.ode_export - INFO - Finished computing dwdp              ␣
↪         ++ (3.77E-02s)
2021-11-30 16:57:58.658 - amici.ode_export - INFO - Finished writing dwdp.cpp            ␣
↪          + (5.28E-02s)
2021-11-30 16:57:58.680 - amici.ode_export - INFO - Finished running smart_jacobian      ␣
↪        +++ (1.15E-02s)
2021-11-30 16:57:58.687 - amici.ode_export - INFO - Finished simplifying dwdx            ␣
↪        +++ (1.95E-03s)
2021-11-30 16:57:58.687 - amici.ode_export - INFO - Finished computing dwdx              ␣
↪         ++ (2.17E-02s)
```

(continues on next page)

(continued from previous page)

```
2021-11-30 16:57:58.696 - amici.ode_export - INFO - Finished writing dwdx.cpp       ␣
↪          + (3.37E-02s)
2021-11-30 16:57:58.705 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪        +++ (1.05E-04s)
2021-11-30 16:57:58.710 - amici.ode_export - INFO - Finished simplifying dwdw       ␣
↪        +++ (6.07E-04s)
2021-11-30 16:57:58.711 - amici.ode_export - INFO - Finished computing dwdw         ␣
↪         ++ (8.67E-03s)
2021-11-30 16:57:58.713 - amici.ode_export - INFO - Finished writing dwdw.cpp       ␣
↪          + (1.25E-02s)
2021-11-30 16:57:58.729 - amici.ode_export - INFO - Finished simplifying xdot       ␣
↪       ++++ (5.40E-03s)
2021-11-30 16:57:58.729 - amici.ode_export - INFO - Finished computing xdot         ␣
↪        +++ (8.78E-03s)
2021-11-30 16:57:58.746 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪        +++ (1.30E-02s)
2021-11-30 16:57:58.752 - amici.ode_export - INFO - Finished simplifying dxdotdw    ␣
↪        +++ (2.97E-04s)
2021-11-30 16:57:58.753 - amici.ode_export - INFO - Finished computing dxdotdw      ␣
↪         ++ (3.42E-02s)
2021-11-30 16:57:58.765 - amici.ode_export - INFO - Finished writing dxdotdw.cpp    ␣
↪          + (4.89E-02s)
2021-11-30 16:57:58.777 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪        +++ (9.65E-05s)
2021-11-30 16:57:58.780 - amici.ode_export - INFO - Finished simplifying dxdotdx_
↪explicit       +++ (7.85E-05s)
2021-11-30 16:57:58.781 - amici.ode_export - INFO - Finished computing dxdotdx_explicit ␣
↪         ++ (8.68E-03s)
2021-11-30 16:57:58.782 - amici.ode_export - INFO - Finished writing dxdotdx_explicit.
↪cpp         + (1.24E-02s)
2021-11-30 16:57:58.793 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪        +++ (9.18E-05s)
2021-11-30 16:57:58.797 - amici.ode_export - INFO - Finished simplifying dxdotdp_
↪explicit       +++ (1.22E-04s)
2021-11-30 16:57:58.798 - amici.ode_export - INFO - Finished computing dxdotdp_explicit ␣
↪         ++ (8.08E-03s)
2021-11-30 16:57:58.799 - amici.ode_export - INFO - Finished writing dxdotdp_explicit.
↪cpp         + (1.36E-02s)
2021-11-30 16:57:58.816 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪       ++++ (1.92E-03s)
2021-11-30 16:57:58.821 - amici.ode_export - INFO - Finished simplifying dydx       ␣
↪       ++++ (1.55E-04s)
2021-11-30 16:57:58.822 - amici.ode_export - INFO - Finished computing dydx         ␣
↪        +++ (1.20E-02s)
2021-11-30 16:57:58.832 - amici.ode_export - INFO - Finished running smart_jacobian ␣
↪       ++++ (1.11E-04s)
2021-11-30 16:57:58.837 - amici.ode_export - INFO - Finished simplifying dydw       ␣
↪       ++++ (4.09E-04s)
2021-11-30 16:57:58.837 - amici.ode_export - INFO - Finished computing dydw         ␣
↪        +++ (9.68E-03s)
2021-11-30 16:57:58.844 - amici.ode_export - INFO - Finished simplifying dydx       ␣
↪        +++ (1.97E-04s)
```

(continues on next page)

(continued from previous page)

```
2021-11-30 16:57:58.845 - amici.ode_export - INFO - Finished computing dydx                ␣
↪         ++ (3.77E-02s)
2021-11-30 16:57:58.849 - amici.ode_export - INFO - Finished writing dydx.cpp              ␣
↪          + (4.51E-02s)
2021-11-30 16:57:58.862 - amici.ode_export - INFO - Finished running smart_jacobian        ␣
↪       ++++ (1.74E-03s)
2021-11-30 16:57:58.866 - amici.ode_export - INFO - Finished simplifying dydp              ␣
↪       ++++ (3.88E-04s)
2021-11-30 16:57:58.867 - amici.ode_export - INFO - Finished computing dydp                ␣
↪        +++ (9.93E-03s)
2021-11-30 16:57:58.872 - amici.ode_export - INFO - Finished simplifying dydp              ␣
↪        +++ (1.78E-04s)
2021-11-30 16:57:58.873 - amici.ode_export - INFO - Finished computing dydp                ␣
↪         ++ (1.84E-02s)
2021-11-30 16:57:58.876 - amici.ode_export - INFO - Finished writing dydp.cpp              ␣
↪          + (2.33E-02s)
2021-11-30 16:57:58.887 - amici.ode_export - INFO - Finished running smart_jacobian        ␣
↪        +++ (1.58E-03s)
2021-11-30 16:57:58.892 - amici.ode_export - INFO - Finished simplifying dsigmaydp         ␣
↪        +++ (2.41E-04s)
2021-11-30 16:57:58.893 - amici.ode_export - INFO - Finished computing dsigmaydp           ␣
↪         ++ (1.08E-02s)
2021-11-30 16:57:58.894 - amici.ode_export - INFO - Finished writing dsigmaydp.cpp         ␣
↪          + (1.50E-02s)
2021-11-30 16:57:58.901 - amici.ode_export - INFO - Finished writing sigmay.cpp            ␣
↪          + (2.68E-03s)
2021-11-30 16:57:58.909 - amici.ode_export - INFO - Finished computing stau                ␣
↪         ++ (1.35E-04s)
2021-11-30 16:57:58.910 - amici.ode_export - INFO - Finished writing stau.cpp              ␣
↪          + (4.10E-03s)
2021-11-30 16:57:58.916 - amici.ode_export - INFO - Finished computing deltax              ␣
↪         ++ (1.34E-04s)
2021-11-30 16:57:58.916 - amici.ode_export - INFO - Finished writing deltax.cpp            ␣
↪          + (3.34E-03s)
2021-11-30 16:57:58.924 - amici.ode_export - INFO - Finished computing deltasx             ␣
↪         ++ (1.99E-04s)
2021-11-30 16:57:58.925 - amici.ode_export - INFO - Finished writing deltasx.cpp           ␣
↪          + (3.88E-03s)
2021-11-30 16:57:58.934 - amici.ode_export - INFO - Finished writing w.cpp                 ␣
↪          + (5.23E-03s)
2021-11-30 16:57:58.945 - amici.ode_export - INFO - Finished simplifying x0                ␣
↪        +++ (7.68E-05s)
2021-11-30 16:57:58.946 - amici.ode_export - INFO - Finished computing x0                  ␣
↪         ++ (4.22E-03s)
2021-11-30 16:57:58.949 - amici.ode_export - INFO - Finished writing x0.cpp                ␣
↪          + (1.04E-02s)
2021-11-30 16:57:58.961 - amici.ode_export - INFO - Finished simplifying x0_
↪fixedParameters     +++ (4.96E-05s)
2021-11-30 16:57:58.962 - amici.ode_export - INFO - Finished computing x0_
↪fixedParameters        ++ (4.51E-03s)
2021-11-30 16:57:58.963 - amici.ode_export - INFO - Finished writing x0_fixedParameters.
↪cpp       + (8.84E-03s)
```

(continues on next page)

(continued from previous page)

```
2021-11-30 16:57:58.982 - amici.ode_export - INFO - Finished running smart_jacobian     ␣
↪         +++ (2.15E-04s)
2021-11-30 16:57:58.987 - amici.ode_export - INFO - Finished simplifying sx0            ␣
↪         +++ (1.20E-04s)
2021-11-30 16:57:58.988 - amici.ode_export - INFO - Finished computing sx0              ␣
↪          ++ (9.27E-03s)
2021-11-30 16:57:58.989 - amici.ode_export - INFO - Finished writing sx0.cpp            ␣
↪           + (2.26E-02s)
2021-11-30 16:57:58.999 - amici.ode_export - INFO - Finished running smart_jacobian     ␣
↪         +++ (3.06E-05s)
2021-11-30 16:57:59.003 - amici.ode_export - INFO - Finished running smart_jacobian     ␣
↪         +++ (2.44E-05s)
2021-11-30 16:57:59.008 - amici.ode_export - INFO - Finished simplifying sx0_
↪fixedParameters    +++ (6.08E-05s)
2021-11-30 16:57:59.009 - amici.ode_export - INFO - Finished computing sx0_
↪fixedParameters       ++ (1.28E-02s)
2021-11-30 16:57:59.009 - amici.ode_export - INFO - Finished writing sx0_fixedParameters.
↪cpp      + (1.59E-02s)
2021-11-30 16:57:59.021 - amici.ode_export - INFO - Finished writing xdot.cpp           ␣
↪           + (7.26E-03s)
2021-11-30 16:57:59.028 - amici.ode_export - INFO - Finished writing y.cpp              ␣
↪           + (2.13E-03s)
2021-11-30 16:57:59.036 - amici.ode_export - INFO - Finished simplifying x_rdata        ␣
↪         +++ (5.62E-05s)
2021-11-30 16:57:59.038 - amici.ode_export - INFO - Finished computing x_rdata          ␣
↪          ++ (5.16E-03s)
2021-11-30 16:57:59.041 - amici.ode_export - INFO - Finished writing x_rdata.cpp        ␣
↪           + (9.60E-03s)
2021-11-30 16:57:59.050 - amici.ode_export - INFO - Finished simplifying total_cl       ␣
↪         +++ (4.08E-05s)
2021-11-30 16:57:59.051 - amici.ode_export - INFO - Finished computing total_cl         ␣
↪          ++ (3.64E-03s)
2021-11-30 16:57:59.052 - amici.ode_export - INFO - Finished writing total_cl.cpp       ␣
↪           + (7.19E-03s)
2021-11-30 16:57:59.064 - amici.ode_export - INFO - Finished simplifying x_solver       ␣
↪         +++ (7.69E-05s)
2021-11-30 16:57:59.065 - amici.ode_export - INFO - Finished computing x_solver         ␣
↪          ++ (4.62E-03s)
2021-11-30 16:57:59.068 - amici.ode_export - INFO - Finished writing x_solver.cpp       ␣
↪           + (1.13E-02s)
2021-11-30 16:57:59.079 - amici.ode_export - INFO - Finished generating cpp code        ␣
↪             (8.66E-01s)
2021-11-30 16:58:07.834 - amici.ode_export - INFO - Finished compiling cpp code         ␣
↪             (8.75E+00s)
```

Importing the module and loading the model

If everything went well, we need to add the previously selected model output directory to our PYTHON_PATH and are then ready to load newly generated model:

[8]:
```
sys.path.insert(0, os.path.abspath(model_output_dir))
model_module = importlib.import_module(model_name)
```

And get an instance of our model from which we can retrieve information such as parameter names:

[9]:
```
model = model_module.getModel()

print("Model name:", model.getName())
print("Model parameters:", model.getParameterIds())
print("Model outputs:    ", model.getObservableIds())
print("Model states:     ", model.getStateIds())
```

```
Model name: model_steadystate_scaled
Model parameters: ('p1', 'p2', 'p3', 'p4', 'p5', 'scaling_x1', 'offset_x2', 'observable_
↪x1withsigma_sigma')
Model outputs:    ('observable_x1', 'observable_x2', 'observable_x3', 'observable_x1_
↪scaled', 'observable_x2_offsetted', 'observable_x1withsigma')
Model states:     ('x1', 'x2', 'x3')
```

Running simulations and analyzing results

After importing the model, we can run simulations using `amici.runAmiciSimulation`. This requires a `Model` instance and a `Solver` instance. Optionally you can provide measurements inside an `ExpData` instance, as shown later in this notebook.

[10]:
```
# Create Model instance
model = model_module.getModel()

# set timepoints for which we want to simulate the model
model.setTimepoints(np.linspace(0, 60, 60))

# Create solver instance
solver = model.getSolver()

# Run simulation using default model parameters and solver options
rdata = amici.runAmiciSimulation(model, solver)
```

[11]:
```
print('Simulation was run using model default parameters as specified in the SBML model:
↪')
print(model.getParameters())
```

```
Simulation was run using model default parameters as specified in the SBML model:
(1.0, 0.5, 0.4, 2.0, 0.1, 2.0, 3.0, 0.2)
```

Simulation results are provided as `numpy.ndarrays` in the returned dictionary:

[12]:
```
#np.set_printoptions(threshold=8, edgeitems=2)
for key, value in rdata.items():
    print('%12s: ' % key, value)
```

```
          ts:  [ 0.          1.01694915  2.03389831  3.05084746  4.06779661  5.08474576
  6.10169492  7.11864407  8.13559322  9.15254237 10.16949153 11.18644068
 12.20338983 13.22033898 14.23728814 15.25423729 16.27118644 17.28813559
 18.30508475 19.3220339  20.33898305 21.3559322  22.37288136 23.38983051
 24.40677966 25.42372881 26.44067797 27.45762712 28.47457627 29.49152542
 30.50847458 31.52542373 32.54237288 33.55932203 34.57627119 35.59322034
 36.61016949 37.62711864 38.6440678  39.66101695 40.6779661  41.69491525
 42.71186441 43.72881356 44.74576271 45.76271186 46.77966102 47.79661017
 48.81355932 49.83050847 50.84745763 51.86440678 52.88135593 53.89830508
 54.91525424 55.93220339 56.94915254 57.96610169 58.98305085 60.        ]
           x:  [[0.1        0.4        0.7       ]
 [0.57995052 0.73365809 0.0951589 ]
 [0.55996496 0.71470091 0.0694127 ]
 [0.5462855  0.68030366 0.06349394]
 [0.53561883 0.64937432 0.05923555]
 [0.52636487 0.62259567 0.05568686]
 [0.51822013 0.59943346 0.05268079]
 [0.51103767 0.57935661 0.05012037]
 [0.5047003  0.56191592 0.04793052]
 [0.49910666 0.54673518 0.0460508 ]
 [0.49416809 0.53349812 0.04443205]
 [0.48980687 0.52193767 0.04303399]
 [0.48595476 0.51182731 0.04182339]
 [0.48255176 0.50297412 0.04077267]
 [0.47954511 0.49521318 0.03985882]
 [0.47688833 0.48840304 0.03906254]
 [0.47454049 0.48242198 0.03836756]
 [0.47246548 0.47716502 0.0377601 ]
 [0.47063147 0.47254128 0.03722844]
 [0.46901037 0.46847202 0.03676259]
 [0.46757739 0.46488881 0.03635397]
 [0.46631065 0.46173207 0.03599523]
 [0.46519082 0.45894987 0.03568002]
 [0.46420083 0.45649684 0.03540285]
 [0.4633256  0.45433332 0.03515899]
 [0.4625518  0.45242457 0.03494429]
 [0.46186768 0.45074016 0.03475519]
 [0.46126282 0.44925337 0.03458856]
 [0.46072804 0.44794075 0.03444166]
 [0.46025521 0.44678168 0.03431212]
 [0.45983714 0.44575804 0.03419784]
 [0.45946749 0.44485388 0.03409701]
 [0.45914065 0.44405514 0.03400802]
 [0.45885167 0.44334947 0.03392946]
 [0.45859615 0.44272595 0.03386009]
 [0.45837021 0.44217497 0.03379883]
 [0.45817043 0.44168805 0.03374473]
 [0.45799379 0.44125772 0.03369693]
 [0.4578376  0.44087738 0.03365471]
```

(continues on next page)

(continued from previous page)

```
[0.45769949 0.44054121 0.0336174 ]
[0.45757737 0.44024405 0.03358444]
[0.45746939 0.43998137 0.03355531]
[0.45737391 0.43974917 0.03352956]
[0.45728948 0.43954389 0.03350681]
[0.45721483 0.43936242 0.0334867 ]
[0.45714882 0.43920198 0.03346892]
[0.45709045 0.43906014 0.03345321]
[0.45703884 0.43893474 0.03343932]
[0.4569932  0.43882387 0.03342704]
[0.45695285 0.43872584 0.03341618]
[0.45691717 0.43863917 0.03340658]
[0.45688561 0.43856254 0.0333981 ]
[0.45685771 0.43849478 0.0333906 ]
[0.45683304 0.43843488 0.03338397]
[0.45681123 0.4383819  0.0333781 ]
[0.45679194 0.43833507 0.03337292]
[0.45677488 0.43829365 0.03336833]
[0.4567598  0.43825703 0.03336428]
[0.45674646 0.43822466 0.0333607 ]
[0.45673467 0.43819603 0.03335753]]
         x0:  [0.1 0.4 0.7]
       x_ss:  [nan nan nan]
         sx:  None
        sx0:  None
      sx_ss:  None
          y:  [[0.1        0.4        0.7        0.2        3.4        0.1       ]
[0.57995052 0.73365809 0.0951589  1.15990103 3.73365809 0.57995052]
[0.55996496 0.71470091 0.0694127  1.11992992 3.71470091 0.55996496]
[0.5462855  0.68030366 0.06349394 1.092571   3.68030366 0.5462855 ]
[0.53561883 0.64937432 0.05923555 1.07123766 3.64937432 0.53561883]
[0.52636487 0.62259567 0.05568686 1.05272975 3.62259567 0.52636487]
[0.51822013 0.59943346 0.05268079 1.03644027 3.59943346 0.51822013]
[0.51103767 0.57935661 0.05012037 1.02207533 3.57935661 0.51103767]
[0.5047003  0.56191592 0.04793052 1.00940059 3.56191592 0.5047003 ]
[0.49910666 0.54673518 0.0460508  0.99821331 3.54673518 0.49910666]
[0.49416809 0.53349812 0.04443205 0.98833618 3.53349812 0.49416809]
[0.48980687 0.52193767 0.04303399 0.97961374 3.52193767 0.48980687]
[0.48595476 0.51182731 0.04182339 0.97190952 3.51182731 0.48595476]
[0.48255176 0.50297412 0.04077267 0.96510352 3.50297412 0.48255176]
[0.47954511 0.49521318 0.03985882 0.95909022 3.49521318 0.47954511]
[0.47688833 0.48840304 0.03906254 0.95377667 3.48840304 0.47688833]
[0.47454049 0.48242198 0.03836756 0.94908097 3.48242198 0.47454049]
[0.47246548 0.47716502 0.0377601  0.94493095 3.47716502 0.47246548]
[0.47063147 0.47254128 0.03722844 0.94126293 3.47254128 0.47063147]
[0.46901037 0.46847202 0.03676259 0.93802074 3.46847202 0.46901037]
[0.46757739 0.46488881 0.03635397 0.93515478 3.46488881 0.46757739]
[0.46631065 0.46173207 0.03599523 0.9326213  3.46173207 0.46631065]
[0.46519082 0.45894987 0.03568002 0.93038164 3.45894987 0.46519082]
[0.46420083 0.45649684 0.03540285 0.92840166 3.45649684 0.46420083]
[0.4633256  0.45433332 0.03515899 0.92665119 3.45433332 0.4633256 ]
[0.4625518  0.45242457 0.03494429 0.9251036  3.45242457 0.4625518 ]
```

(continues on next page)

(continued from previous page)

```
[0.46186768 0.45074016 0.03475519 0.92373536 3.45074016 0.46186768]
[0.46126282 0.44925337 0.03458856 0.92252564 3.44925337 0.46126282]
[0.46072804 0.44794075 0.03444166 0.92145608 3.44794075 0.46072804]
[0.46025521 0.44678168 0.03431212 0.92051041 3.44678168 0.46025521]
[0.45983714 0.44575804 0.03419784 0.91967427 3.44575804 0.45983714]
[0.45946749 0.44485388 0.03409701 0.91893498 3.44485388 0.45946749]
[0.45914065 0.44405514 0.03400802 0.91828131 3.44405514 0.45914065]
[0.45885167 0.44334947 0.03392946 0.91770333 3.44334947 0.45885167]
[0.45859615 0.44272595 0.03386009 0.91719229 3.44272595 0.45859615]
[0.45837021 0.44217497 0.03379883 0.91674042 3.44217497 0.45837021]
[0.45817043 0.44168805 0.03374473 0.91634087 3.44168805 0.45817043]
[0.45799379 0.44125772 0.03369693 0.91598758 3.44125772 0.45799379]
[0.4578376  0.44087738 0.03365471 0.9156752  3.44087738 0.4578376 ]
[0.45769949 0.44054121 0.0336174  0.91539898 3.44054121 0.45769949]
[0.45757737 0.44024405 0.03358444 0.91515474 3.44024405 0.45757737]
[0.45746939 0.43998137 0.03355531 0.91493878 3.43998137 0.45746939]
[0.45737391 0.43974917 0.03352956 0.91474782 3.43974917 0.45737391]
[0.45728948 0.43954389 0.03350681 0.91457897 3.43954389 0.45728948]
[0.45721483 0.43936242 0.0334867  0.91442966 3.43936242 0.45721483]
[0.45714882 0.43920198 0.03346892 0.91429764 3.43920198 0.45714882]
[0.45709045 0.43906014 0.03345321 0.91418091 3.43906014 0.45709045]
[0.45703884 0.43893474 0.03343932 0.91407768 3.43893474 0.45703884]
[0.4569932  0.43882387 0.03342704 0.91398641 3.43882387 0.4569932 ]
[0.45695285 0.43872584 0.03341618 0.9139057  3.43872584 0.45695285]
[0.45691717 0.43863917 0.03340658 0.91383433 3.43863917 0.45691717]
[0.45688561 0.43856254 0.0333981  0.91377123 3.43856254 0.45688561]
[0.45685771 0.43849478 0.0333906  0.91371543 3.43849478 0.45685771]
[0.45683304 0.43843488 0.03338397 0.91366609 3.43843488 0.45683304]
[0.45681123 0.4383819  0.0333781  0.91362246 3.4383819  0.45681123]
[0.45679194 0.43833507 0.03337292 0.91358388 3.43833507 0.45679194]
[0.45677488 0.43829365 0.03336833 0.91354976 3.43829365 0.45677488]
[0.4567598  0.43825703 0.03336428 0.9135196  3.43825703 0.4567598 ]
[0.45674646 0.43822466 0.0333607  0.91349292 3.43822466 0.45674646]
[0.45673467 0.43819603 0.03335753 0.91346934 3.43819603 0.45673467]]
     sigmay:  [[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
```

(continues on next page)

(continued from previous page)

```
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]
 [1.   1.   1.   1.   1.   0.2]]
          sy:  None
     ssigmay:  None
           z:  None
          rz:  None
      sigmaz:  None
          sz:  None
         srz:  None
     ssigmaz:  None
        sllh:  None
       s2llh:  None
```

(continues on next page)

(continued from previous page)

```
          J:  [[-2.04603669  0.57163267  2.        ]
 [ 0.69437133 -0.62836733  2.        ]
 [ 0.21909801  0.22836733 -3.        ]]
       xdot:  [-1.08967281e-05 -2.64534209e-05 -2.92761862e-06]
     status:  0.0
        llh:  nan
       chi2:  nan
        res:  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
       sres:  None
        FIM:  None
          w:  [[3.4        0.1        0.2        0.4        0.1        0.7
  0.01       0.02       0.16       1.4        0.7        0.1       ]
 [3.73365809 0.57995052 1.15990103 0.73365809 0.57995052 0.0951589
  0.3363426  0.21274269 0.29346324 0.1903178  0.0951589  0.1       ]
 [3.71470091 0.55996496 1.11992992 0.71470091 0.55996496 0.0694127
  0.31356076 0.20010373 0.28588036 0.13882541 0.0694127  0.1       ]
 [3.68030366 0.5462855  1.092571   0.68030366 0.5462855  0.06349394
  0.29842785 0.18582001 0.27212146 0.12698788 0.06349394 0.1       ]
 [3.64937432 0.53561883 1.07123766 0.64937432 0.53561883 0.05923555
  0.28688753 0.17390856 0.25974973 0.1184711  0.05923555 0.1       ]
 [3.62259567 0.52636487 1.05272975 0.62259567 0.52636487 0.05568686
  0.27705998 0.16385625 0.24903827 0.11137372 0.05568686 0.1       ]
 [3.59943346 0.51822013 1.03644027 0.59943346 0.51822013 0.05268079
  0.26855211 0.15531924 0.23977338 0.10536158 0.05268079 0.1       ]
 [3.57935661 0.51103767 1.02207533 0.57935661 0.51103767 0.05012037
  0.2611595  0.14803652 0.23174264 0.10024074 0.05012037 0.1       ]
 [3.56191592 0.5047003  1.00940059 0.56191592 0.5047003  0.04793052
  0.25472239 0.14179957 0.22476637 0.09586103 0.04793052 0.1       ]
 [3.54673518 0.49910666 0.99821331 0.54673518 0.49910666 0.0460508
  0.24910746 0.13643958 0.21869407 0.0921016  0.0460508  0.1       ]
 [3.53349812 0.49416809 0.98833618 0.53349812 0.49416809 0.04443205
  0.2442021  0.13181887 0.21339925 0.08886411 0.04443205 0.1       ]
 [3.52193767 0.48980687 0.97961374 0.52193767 0.48980687 0.04303399
  0.23991077 0.12782433 0.20877507 0.08606799 0.04303399 0.1       ]
 [3.51182731 0.48595476 0.97190952 0.51182731 0.48595476 0.04182339
  0.23615203 0.12436246 0.20473093 0.08364678 0.04182339 0.1       ]
 [3.50297412 0.48255176 0.96510352 0.50297412 0.48255176 0.04077267
  0.2328562  0.12135552 0.20118965 0.08154533 0.04077267 0.1       ]
```

(continues on next page)

(continued from previous page)

```
[3.49521318 0.47954511 0.95909022 0.49521318 0.47954511 0.03985882
 0.22996351 0.11873853 0.19808527 0.07971763 0.03985882 0.1       ]
[3.48840304 0.47688833 0.95377667 0.48840304 0.47688833 0.03906254
 0.22742248 0.11645686 0.19536122 0.07812507 0.03906254 0.1       ]
[3.48242198 0.47454049 0.94908097 0.48242198 0.47454049 0.03836756
 0.22518867 0.11446438 0.19296879 0.07673511 0.03836756 0.1       ]
[3.47716502 0.47246548 0.94493095 0.47716502 0.47246548 0.0377601
 0.22322363 0.112722   0.19086601 0.0755202  0.0377601  0.1       ]
[3.47254128 0.47063147 0.94126293 0.47254128 0.47063147 0.03722844
 0.22149398 0.1111964  0.18901651 0.07445688 0.03722844 0.1       ]
[3.46847202 0.46901037 0.93802074 0.46847202 0.46901037 0.03676259
 0.21997073 0.10985912 0.18738881 0.07352518 0.03676259 0.1       ]
[3.46488881 0.46757739 0.93515478 0.46488881 0.46757739 0.03635397
 0.21862862 0.10868575 0.18595552 0.07270794 0.03635397 0.1       ]
[3.46173207 0.46631065 0.9326213  0.46173207 0.46631065 0.03599523
 0.21744562 0.10765529 0.18469283 0.07199046 0.03599523 0.1       ]
[3.45894987 0.46519082 0.93038164 0.45894987 0.46519082 0.03568002
 0.2164025  0.10674963 0.18357995 0.07136003 0.03568002 0.1       ]
[3.45649684 0.46420083 0.92840166 0.45649684 0.46420083 0.03540285
 0.21548241 0.10595311 0.18259874 0.0708057  0.03540285 0.1       ]
[3.45433332 0.4633256  0.92665119 0.45433332 0.4633256  0.03515899
 0.21467061 0.10525213 0.18173333 0.07031797 0.03515899 0.1       ]
[3.45242457 0.4625518  0.9251036  0.45242457 0.4625518  0.03494429
 0.21395417 0.1046349  0.18096983 0.06988859 0.03494429 0.1       ]
[3.45074016 0.46186768 0.92373536 0.45074016 0.46186768 0.03475519
 0.21332175 0.10409116 0.18029606 0.06951039 0.03475519 0.1       ]
[3.44925337 0.46126282 0.92252564 0.44925337 0.46126282 0.03458856
 0.21276339 0.10361194 0.17970135 0.06917712 0.03458856 0.1       ]
[3.44794075 0.46072804 0.92145608 0.44794075 0.46072804 0.03444166
 0.21227033 0.10318943 0.1791763  0.06888332 0.03444166 0.1       ]
[3.44678168 0.46025521 0.92051041 0.44678168 0.46025521 0.03431212
 0.21183485 0.1028168  0.17871267 0.06862424 0.03431212 0.1       ]
[3.44575804 0.45983714 0.91967427 0.44575804 0.45983714 0.03419784
 0.21145019 0.10248805 0.17830322 0.06839569 0.03419784 0.1       ]
[3.44485388 0.45946749 0.91893498 0.44485388 0.45946749 0.03409701
 0.21111037 0.10219795 0.17794155 0.06819402 0.03409701 0.1       ]
[3.44405514 0.45914065 0.91828131 0.44405514 0.45914065 0.03400802
 0.21081014 0.10194188 0.17762206 0.06801603 0.03400802 0.1       ]
[3.44334947 0.45885167 0.91770333 0.44334947 0.45885167 0.03392946
 0.21054485 0.10171582 0.17733979 0.06785891 0.03392946 0.1       ]
[3.44272595 0.45859615 0.91719229 0.44272595 0.45859615 0.03386009
 0.21031042 0.10151621 0.17709038 0.06772018 0.03386009 0.1       ]
[3.44217497 0.45837021 0.91674042 0.44217497 0.45837021 0.03379883
 0.21010325 0.10133992 0.17686999 0.06759766 0.03379883 0.1       ]
[3.44168805 0.45817043 0.91634087 0.44168805 0.45817043 0.03374473
 0.20992015 0.1011842  0.17667522 0.06748945 0.03374473 0.1       ]
[3.44125772 0.45799379 0.91598758 0.44125772 0.45799379 0.03369693
 0.20975831 0.10104665 0.17650309 0.06739386 0.03369693 0.1       ]
[3.44087738 0.4578376  0.9156752  0.44087738 0.4578376  0.03365471
 0.20961527 0.10092512 0.17635095 0.06730942 0.03365471 0.1       ]
[3.44054121 0.45769949 0.91539898 0.44054121 0.45769949 0.0336174
 0.20948882 0.10081774 0.17621648 0.0672348  0.0336174  0.1       ]
```

(continues on next page)

(continued from previous page)

```
 [3.44024405 0.45757737 0.91515474 0.44024405 0.45757737 0.03358444
  0.20937705 0.10072286 0.17609762 0.06716887 0.03358444 0.1       ]
 [3.43998137 0.45746939 0.91493878 0.43998137 0.45746939 0.03355531
  0.20927824 0.10063901 0.17599255 0.06711061 0.03355531 0.1       ]
 [3.43974917 0.45737391 0.91474782 0.43974917 0.45737391 0.03352956
  0.20919089 0.1005649  0.17589967 0.06705912 0.03352956 0.1       ]
 [3.43954389 0.45728948 0.91457897 0.43954389 0.45728948 0.03350681
  0.20911367 0.1004994  0.17581756 0.06701361 0.03350681 0.1       ]
 [3.43936242 0.45721483 0.91442966 0.43936242 0.45721483 0.0334867
  0.2090454  0.10044151 0.17574497 0.06697339 0.0334867  0.1       ]
 [3.43920198 0.45714882 0.91429764 0.43920198 0.45714882 0.03346892
  0.20898505 0.10039033 0.17568079 0.06693784 0.03346892 0.1       ]
 [3.43906014 0.45709045 0.91418091 0.43906014 0.45709045 0.03345321
  0.20893168 0.1003451  0.17562406 0.06690641 0.03345321 0.1       ]
 [3.43893474 0.45703884 0.91407768 0.43893474 0.45703884 0.03343932
  0.2088845  0.10030511 0.1755739  0.06687863 0.03343932 0.1       ]
 [3.43882387 0.4569932  0.91398641 0.43882387 0.4569932  0.03342704
  0.20884279 0.10026976 0.17552955 0.06685407 0.03342704 0.1       ]
 [3.43872584 0.45695285 0.9139057  0.43872584 0.45695285 0.03341618
  0.20880591 0.10023851 0.17549034 0.06683236 0.03341618 0.1       ]
 [3.43863917 0.45691717 0.91383433 0.43863917 0.45691717 0.03340658
  0.2087733  0.10021088 0.17545567 0.06681317 0.03340658 0.1       ]
 [3.43856254 0.45688561 0.91377123 0.43856254 0.45688561 0.0333981
  0.20874446 0.10018646 0.17542502 0.0667962  0.0333981  0.1       ]
 [3.43849478 0.45685771 0.91371543 0.43849478 0.45685771 0.0333906
  0.20871897 0.10016486 0.17539791 0.0667812  0.0333906  0.1       ]
 [3.43843488 0.45683304 0.91366609 0.43843488 0.45683304 0.03338397
  0.20869643 0.10014577 0.17537395 0.06676793 0.03338397 0.1       ]
 [3.4383819  0.45681123 0.91362246 0.4383819  0.45681123 0.0333781
  0.2086765  0.10012889 0.17535276 0.0667562  0.0333781  0.1       ]
 [3.43833507 0.45679194 0.91358388 0.43833507 0.45679194 0.03337292
  0.20865887 0.10011396 0.17533403 0.06674583 0.03337292 0.1       ]
 [3.43829365 0.45677488 0.91354976 0.43829365 0.45677488 0.03336833
  0.20864329 0.10010077 0.17531746 0.06673667 0.03336833 0.1       ]
 [3.43825703 0.4567598  0.9135196  0.43825703 0.4567598  0.03336428
  0.20862951 0.1000891  0.17530281 0.06672856 0.03336428 0.1       ]
 [3.43822466 0.45674646 0.91349292 0.43822466 0.45674646 0.0333607
  0.20861733 0.10007878 0.17528986 0.06672139 0.0333607  0.1       ]
 [3.43819603 0.45673467 0.91346934 0.43819603 0.45673467 0.03335753
  0.20860656 0.10006966 0.17527841 0.06671506 0.03335753 0.1       ]]
  preeq_wrms:  nan
     preeq_t:  nan
preeq_numlinsteps:  None
preeq_numsteps:  [[0 0 0]]
preeq_numstepsB:  0.0
preeq_status:  [[0 0 0]]
preeq_cpu_time:  0.0
preeq_cpu_timeB:  0.0
 posteq_wrms:  nan
    posteq_t:  nan
posteq_numlinsteps:  None
posteq_numsteps:  [[0 0 0]]
```

(continues on next page)

(continued from previous page)

```
posteq_numstepsB:  0.0
posteq_status:  [[0 0 0]]
posteq_cpu_time:  0.0
posteq_cpu_timeB:  0.0
    numsteps:  [  0 100 144 165 181 191 200 207 213 218 223 228 233 237 241 245 249 252
 255 258 261 264 266 269 272 275 278 282 286 290 293 296 299 303 307 311
 314 317 321 325 328 333 337 340 342 344 346 348 350 352 354 356 358 359
 360 361 362 363 364 365]
 numrhsevals:  [  0 114 160 193 212 227 237 248 255 260 267 272 277 282 287 292 296 300
 303 306 309 312 315 318 322 325 329 333 337 342 345 348 352 358 365 369
 372 376 381 385 389 395 400 403 405 407 409 411 413 415 417 419 421 422
 424 426 427 428 429 430]
numerrtestfails:  [0 1 1 3 3 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6␣
↪6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6]
numnonlinsolvconvfails:  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0␣
↪0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
       order:  [0 5 5 5 5 5 4 5 4 5 4 4 4 4 4 4 5 5 5 5 5 5 5 5 4 4 5 5 5 4 4 4 5 5 5 4 4
 4 4 5 5 4 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4]
    cpu_time:  3.176
   numstepsB:  None
numrhsevalsB:  None
numerrtestfailsB:  None
numnonlinsolvconvfailsB:  None
   cpu_timeB:  0.0
```

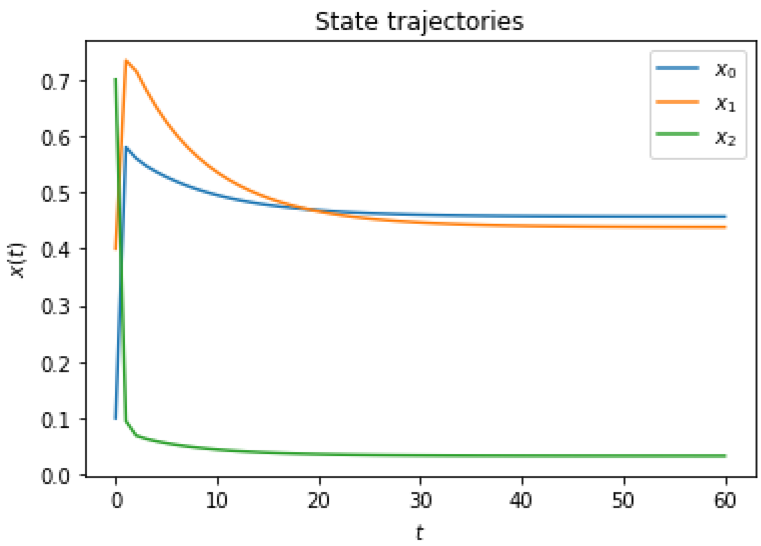
[13]:
```
print(model.getParameters())
```

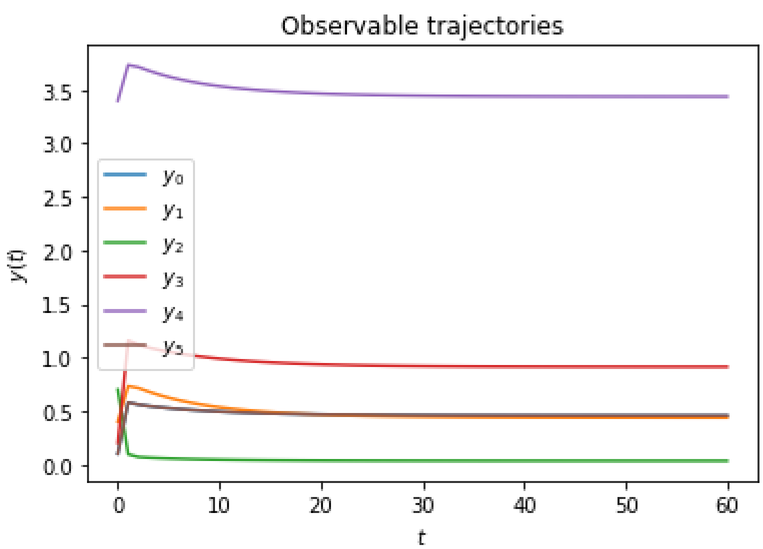
```
(1.0, 0.5, 0.4, 2.0, 0.1, 2.0, 3.0, 0.2)
```

Plotting trajectories

The simulation results above did not look too appealing. Let’s plot the trajectories of the model states and outputs them using `matplotlib.pyplot`:

[14]:
```
import amici.plotting
amici.plotting.plotStateTrajectories(rdata, model = None)
amici.plotting.plotObservableTrajectories(rdata, model = None)
```





Computing likelihood

Often model parameters need to be inferred from experimental data. This is commonly done by maximizing the likelihood of observing the data given to current model parameters. AMICI will compute this likelihood if experimental data is provided to `amici.runAmiciSimulation` as optional third argument. Measurements along with their standard deviations are provided through an `amici.ExpData` instance.

[15]:
```
# Create model instance and set time points for simulation
model = model_module.getModel()
model.setTimepoints(np.linspace(0, 10, 11))

# Create solver instance, keep default options
solver = model.getSolver()

# Run simulation without experimental data
```
(continues on next page)

(continued from previous page)

```
rdata = amici.runAmiciSimulation(model, solver)

# Create ExpData instance from simulation results
edata = amici.ExpData(rdata, 1.0, 0.0)

# Re-run simulation, this time passing "experimental data"
rdata = amici.runAmiciSimulation(model, solver, edata)

print('Log-likelihood %f' % rdata['llh'])
```

```
Log-likelihood -97.118555
```

Simulation tolerances

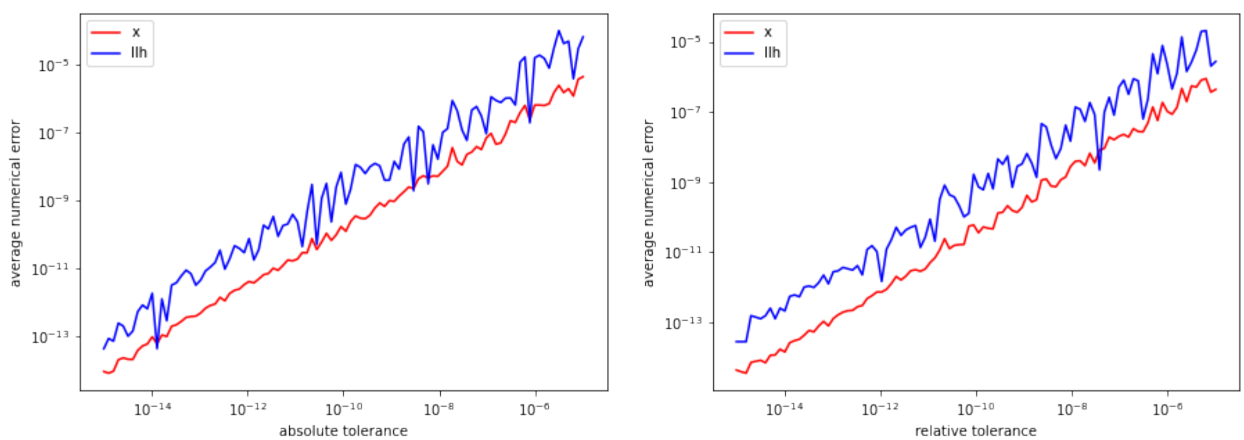
Numerical error tolerances are often critical to get accurate results. For the state variables, integration errors can be controlled using `setRelativeTolerance` and `setAbsoluteTolerance`. Similar functions exist for sensitivities, steadystates and quadratures. We initially compute a reference solution using extremely low tolerances and then assess the influence on integration error for different levels of absolute and relative tolerance.

[16]:
```
solver.setRelativeTolerance(1e-16)
solver.setAbsoluteTolerance(1e-16)
solver.setSensitivityOrder(amici.SensitivityOrder.none)
rdata_ref = amici.runAmiciSimulation(model, solver, edata)

def get_simulation_error(solver):
    rdata = amici.runAmiciSimulation(model, solver, edata)
    return np.mean(np.abs(rdata['x']-rdata_ref['x'])), np.mean(np.abs(rdata['llh']-rdata_
→ref['llh']))

def get_errors(tolfun, tols):
    solver.setRelativeTolerance(1e-16)
    solver.setAbsoluteTolerance(1e-16)
    x_errs = []
    llh_errs = []
    for tol in tols:
        getattr(solver, tolfun)(tol)
        x_err, llh_err = get_simulation_error(solver)
        x_errs.append(x_err)
        llh_errs.append(llh_err)
    return x_errs, llh_errs

atols = np.logspace(-5,-15, 100)
atol_x_errs, atol_llh_errs = get_errors('setAbsoluteTolerance', atols)

rtols = np.logspace(-5,-15, 100)
rtol_x_errs, rtol_llh_errs = get_errors('setRelativeTolerance', rtols)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

def plot_error(tols, x_errs, llh_errs, tolname, ax):
    ax.plot(tols, x_errs, 'r-', label='x')
```

(continues on next page)

(continued from previous page)

```
    ax.plot(tols, llh_errs, 'b-', label='llh')
    ax.set_xscale('log')
    ax.set_yscale('log')
    ax.set_xlabel(f'{tolname} tolerance')
    ax.set_ylabel('average numerical error')
    ax.legend()

plot_error(atols, atol_x_errs, atol_llh_errs, 'absolute', axes[0])
plot_error(rtols, rtol_x_errs, rtol_llh_errs, 'relative', axes[1])

# reset relative tolerance to default value
solver.setRelativeTolerance(1e-8)
solver.setRelativeTolerance(1e-16)
```



Sensitivity analysis

AMICI can provide first- and second-order sensitivities using the forward- or adjoint-method. The respective options are set on the Model and Solver objects.

Forward sensitivity analysis

[17]:
```
model = model_module.getModel()
model.setTimepoints(np.linspace(0, 10, 11))
model.requireSensitivitiesForAllParameters()              # sensitivities w.r.t. all␣
↪parameters
# model.setParameterList([1, 2])                          # sensitivities
# w.r.t. the specified parameters
model.setParameterScale(amici.ParameterScaling.none)        # parameters are used as-is␣
↪(not log-transformed)

solver = model.getSolver()
solver.setSensitivityMethod(amici.SensitivityMethod.forward)        # forward␣
↪sensitivity analysis
solver.setSensitivityOrder(amici.SensitivityOrder.first) # first-order sensitivities
```

(continues on next page)

(continued from previous page)

```
rdata = amici.runAmiciSimulation(model, solver)

# print sensitivity-related results
for key, value in rdata.items():
    if key.startswith('s'):
        print('%12s: ' % key, value)
```

```
          sx:  [[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.00747250e-01  1.19873139e-01 -9.44167985e-03]
  [-1.02561396e-01 -1.88820454e-01  1.01855972e-01]
  [ 4.66193077e-01 -2.86365372e-01  2.39662449e-02]
  [ 4.52560294e-02  1.14631370e-01 -3.34067919e-02]
  [ 4.00672911e-01  1.92564093e-01  4.98877759e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.23007240e-01  1.53979022e-01 -1.26885280e-02]
  [-1.33426939e-01 -3.15955239e-01  9.49575030e-02]
  [ 5.03470377e-01 -3.52731535e-01  2.81567412e-02]
  [ 3.93630714e-02  1.10770683e-01 -1.05673869e-02]
  [ 5.09580304e-01  4.65255489e-01  9.24843702e-02]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.14278104e-01  1.63465064e-01 -1.03268418e-02]
  [-1.60981967e-01 -4.00490452e-01  7.54810648e-02]
  [ 4.87746419e-01 -3.76014315e-01  2.30919334e-02]
  [ 4.28733680e-02  1.15473583e-01 -6.63571687e-03]
  [ 6.05168647e-01  7.07226039e-01  1.23870914e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

 [[-2.05888038e-01  1.69308689e-01 -7.93085660e-03]
  [-1.84663809e-01 -4.65451966e-01  5.95026117e-02]
  [ 4.66407064e-01 -3.87612079e-01  1.76410128e-02]
  [ 4.52451104e-02  1.19865712e-01 -4.73313094e-03]
  [ 6.90798449e-01  9.20396633e-01  1.49475827e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

(continues on next page)

(continued from previous page)

```
[[-1.98803165e-01  1.73327268e-01 -6.03008179e-03]
 [-2.04303740e-01 -5.16111388e-01  4.68785776e-02]
 [ 4.47070326e-01 -3.94304029e-01  1.32107437e-02]
 [ 4.69732048e-02  1.22961727e-01 -3.35899442e-03]
 [ 7.68998995e-01  1.10844286e+00  1.70889328e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

[[-1.92789113e-01  1.75978657e-01 -4.54517629e-03]
 [-2.20500138e-01 -5.55540705e-01  3.68776526e-02]
 [ 4.30424855e-01 -3.97907706e-01  9.75257113e-03]
 [ 4.82793652e-02  1.24952071e-01 -2.30991637e-03]
 [ 8.40805131e-01  1.27504628e+00  1.89020151e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

[[-1.87672774e-01  1.77588334e-01 -3.38318222e-03]
 [-2.33807210e-01 -5.86081383e-01  2.89236334e-02]
 [ 4.16201399e-01 -3.99295277e-01  7.06598588e-03]
 [ 4.92546648e-02  1.26089711e-01 -1.50412006e-03]
 [ 9.06806543e-01  1.42334018e+00  2.04522708e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

[[-1.83320440e-01  1.78410042e-01 -2.47240692e-03]
 [-2.44690164e-01 -6.09568485e-01  2.25774266e-02]
 [ 4.04061655e-01 -3.99063012e-01  4.97908386e-03]
 [ 4.99612484e-02  1.26581014e-01 -8.85891342e-04]
 [ 9.67473970e-01  1.55589415e+00  2.17895305e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

[[-1.79620591e-01  1.78640114e-01 -1.75822439e-03]
 [-2.53540123e-01 -6.27448857e-01  1.75019839e-02]
 [ 3.93704970e-01 -3.97656641e-01  3.35895484e-03]
 [ 5.04492282e-02  1.26586733e-01 -4.13401240e-04]
 [ 1.02322336e+00  1.67481439e+00  2.29524046e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]

[[-1.76478441e-01  1.78430281e-01 -1.19867662e-03]
 [-2.60686971e-01 -6.40868686e-01  1.34365068e-02]
 [ 3.84873835e-01 -3.95414931e-01  2.10369522e-03]
 [ 5.07601805e-02  1.26231631e-01 -5.46465317e-05]
 [ 1.07443160e+00  1.78183962e+00  2.39710937e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

(continues on next page)

(continued from previous page)

```
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00]]]
        sx0:  [[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]]
      sx_ss:  [[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]]
     sigmay:  [[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]
[1.  1.  1.  1.  1.  0.2]]
         sy:  [[[ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e-01
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

[[-2.00747250e-01  1.19873139e-01 -9.44167985e-03 -4.01494500e-01
   1.19873139e-01 -2.00747250e-01]
 [-1.02561396e-01 -1.88820454e-01  1.01855972e-01 -2.05122791e-01
  -1.88820454e-01 -1.02561396e-01]
 [ 4.66193077e-01 -2.86365372e-01  2.39662449e-02  9.32386154e-01
  -2.86365372e-01  4.66193077e-01]
 [ 4.52560294e-02  1.14631370e-01 -3.34067919e-02  9.05120589e-02
```

(continues on next page)

(continued from previous page)

```
   1.14631370e-01  4.52560294e-02]
  [ 4.00672911e-01  1.92564093e-01  4.98877759e-02  8.01345822e-01
    1.92564093e-01  4.00672911e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.80072436e-01
    0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00]]

 [[-2.23007240e-01  1.53979022e-01 -1.26885280e-02 -4.46014480e-01
    1.53979022e-01 -2.23007240e-01]
  [-1.33426939e-01 -3.15955239e-01  9.49575030e-02 -2.66853878e-01
   -3.15955239e-01 -1.33426939e-01]
  [ 5.03470377e-01 -3.52731535e-01  2.81567412e-02  1.00694075e+00
   -3.52731535e-01  5.03470377e-01]
  [ 3.93630714e-02  1.10770683e-01 -1.05673869e-02  7.87261427e-02
    1.10770683e-01  3.93630714e-02]
  [ 5.09580304e-01  4.65255489e-01  9.24843702e-02  1.01916061e+00
    4.65255489e-01  5.09580304e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.60534516e-01
    0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00]]

 [[-2.14278104e-01  1.63465064e-01 -1.03268418e-02 -4.28556209e-01
    1.63465064e-01 -2.14278104e-01]
  [-1.60981967e-01 -4.00490452e-01  7.54810648e-02 -3.21963935e-01
   -4.00490452e-01 -1.60981967e-01]
  [ 4.87746419e-01 -3.76014315e-01  2.30919334e-02  9.75492839e-01
   -3.76014315e-01  4.87746419e-01]
  [ 4.28733680e-02  1.15473583e-01 -6.63571687e-03  8.57467361e-02
    1.15473583e-01  4.28733680e-02]
  [ 6.05168647e-01  7.07226039e-01  1.23870914e-01  1.21033729e+00
    7.07226039e-01  6.05168647e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.46870655e-01
    0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
    0.00000000e+00  0.00000000e+00]]

 [[-2.05888038e-01  1.69308689e-01 -7.93085660e-03 -4.11776077e-01
    1.69308689e-01 -2.05888038e-01]
  [-1.84663809e-01 -4.65451966e-01  5.95026117e-02 -3.69327617e-01
   -4.65451966e-01 -1.84663809e-01]
  [ 4.66407064e-01 -3.87612079e-01  1.76410128e-02  9.32814128e-01
   -3.87612079e-01  4.66407064e-01]
  [ 4.52451104e-02  1.19865712e-01 -4.73313094e-03  9.04902208e-02
    1.19865712e-01  4.52451104e-02]
```

(continues on next page)

(continued from previous page)

```
 [ 6.90798449e-01  9.20396633e-01  1.49475827e-01  1.38159690e+00
   9.20396633e-01  6.90798449e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.36280366e-01
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

[[-1.98803165e-01  1.73327268e-01 -6.03008179e-03 -3.97606330e-01
   1.73327268e-01 -1.98803165e-01]
 [-2.04303740e-01 -5.16111388e-01  4.68785776e-02 -4.08607480e-01
  -5.16111388e-01 -2.04303740e-01]
 [ 4.47070326e-01 -3.94304029e-01  1.32107437e-02  8.94140651e-01
  -3.94304029e-01  4.47070326e-01]
 [ 4.69732048e-02  1.22961727e-01 -3.35899442e-03  9.39464097e-02
   1.22961727e-01  4.69732048e-02]
 [ 7.68998995e-01  1.10844286e+00  1.70889328e-01  1.53799799e+00
   1.10844286e+00  7.68998995e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.27091252e-01
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

[[-1.92789113e-01  1.75978657e-01 -4.54517629e-03 -3.85578227e-01
   1.75978657e-01 -1.92789113e-01]
 [-2.20500138e-01 -5.55540705e-01  3.68776526e-02 -4.41000277e-01
  -5.55540705e-01 -2.20500138e-01]
 [ 4.30424855e-01 -3.97907706e-01  9.75257113e-03  8.60849709e-01
  -3.97907706e-01  4.30424855e-01]
 [ 4.82793652e-02  1.24952071e-01 -2.30991637e-03  9.65587304e-02
   1.24952071e-01  4.82793652e-02]
 [ 8.40805131e-01  1.27504628e+00  1.89020151e-01  1.68161026e+00
   1.27504628e+00  8.40805131e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.18989205e-01
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

[[-1.87672774e-01  1.77588334e-01 -3.38318222e-03 -3.75345548e-01
   1.77588334e-01 -1.87672774e-01]
 [-2.33807210e-01 -5.86081383e-01  2.89236334e-02 -4.67614420e-01
  -5.86081383e-01 -2.33807210e-01]
 [ 4.16201399e-01 -3.99295277e-01  7.06598588e-03  8.32402797e-01
  -3.99295277e-01  4.16201399e-01]
 [ 4.92546648e-02  1.26089711e-01 -1.50412006e-03  9.85093296e-02
   1.26089711e-01  4.92546648e-02]
 [ 9.06806543e-01  1.42334018e+00  2.04522708e-01  1.81361309e+00
```

(continues on next page)

(continued from previous page)

```
   1.42334018e+00  9.06806543e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.11829985e-01
   0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

 [[-1.83320440e-01  1.78410042e-01 -2.47240692e-03 -3.66640879e-01
   1.78410042e-01 -1.83320440e-01]
  [-2.44690164e-01 -6.09568485e-01  2.25774266e-02 -4.89380329e-01
  -6.09568485e-01 -2.44690164e-01]
  [ 4.04061655e-01 -3.99063012e-01  4.97908386e-03  8.08123310e-01
  -3.99063012e-01  4.04061655e-01]
  [ 4.99612484e-02  1.26581014e-01 -8.85891342e-04  9.99224969e-02
   1.26581014e-01  4.99612484e-02]
  [ 9.67473970e-01  1.55589415e+00  2.17895305e-01  1.93494794e+00
   1.55589415e+00  9.67473970e-01]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  5.05500234e-01
   0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

 [[-1.79620591e-01  1.78640114e-01 -1.75822439e-03 -3.59241183e-01
   1.78640114e-01 -1.79620591e-01]
  [-2.53540123e-01 -6.27448857e-01  1.75019839e-02 -5.07080247e-01
  -6.27448857e-01 -2.53540123e-01]
  [ 3.93704970e-01 -3.97656641e-01  3.35895484e-03  7.87409940e-01
  -3.97656641e-01  3.93704970e-01]
  [ 5.04492282e-02  1.26586733e-01 -4.13401240e-04  1.00898456e-01
   1.26586733e-01  5.04492282e-02]
  [ 1.02322336e+00  1.67481439e+00  2.29524046e-01  2.04644672e+00
   1.67481439e+00  1.02322336e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  4.99901907e-01
   0.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]

 [[-1.76478441e-01  1.78430281e-01 -1.19867662e-03 -3.52956882e-01
   1.78430281e-01 -1.76478441e-01]
  [-2.60686971e-01 -6.40868686e-01  1.34365068e-02 -5.21373942e-01
  -6.40868686e-01 -2.60686971e-01]
  [ 3.84873835e-01 -3.95414931e-01  2.10369522e-03  7.69747670e-01
  -3.95414931e-01  3.84873835e-01]
  [ 5.07601805e-02  1.26231631e-01 -5.46465317e-05  1.01520361e-01
   1.26231631e-01  5.07601805e-02]
  [ 1.07443160e+00  1.78183962e+00  2.39710937e-01  2.14886320e+00
   1.78183962e+00  1.07443160e+00]
```

(continues on next page)

(continued from previous page)

```
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  4.94949118e-01
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]]]
    ssigmay:  [[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
```

(continues on next page)

(continued from previous page)

```
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]

[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1.]]]
```

(continues on next page)

(continued from previous page)

```
     sigmaz:  None
         sz:  None
        srz:  None
    ssigmaz:  None
       sllh:  [nan nan nan nan nan nan nan nan]
      s2llh:  None
     status:  0.0
       sres:  [[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0.]
```

(continues on next page)

(continued from previous page)

```
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0.]]
```

Adjoint sensitivity analysis

[18]:
```
# Set model options
model = model_module.getModel()
p_orig = np.array(model.getParameters())
p_orig[list(model.getParameterIds()).index('observable_x1withsigma_sigma')] = 0.1 #␣
↪Change default parameter
model.setParameters(p_orig)
model.setParameterScale(amici.ParameterScaling.none)
model.setTimepoints(np.linspace(0, 10, 21))

solver = model.getSolver()
solver.setMaxSteps(10**4) # Set maximum number of steps for the solver

# simulate time-course to get artificial data
rdata = amici.runAmiciSimulation(model, solver)
edata = amici.ExpData(rdata, 1.0, 0)
edata.fixedParameters = model.getFixedParameters()
# set sigma to 1.0 except for observable 5, so that p[7] is used instead
# (if we have sigma parameterized, the corresponding ExpData entries must NaN, otherwise␣
↪they will override the parameter)
edata.setObservedDataStdDev(rdata['t']*0+np.nan,
                            list(model.getObservableIds()).index('observable_x1withsigma
↪'))

# enable sensitivities
solver.setSensitivityOrder(amici.SensitivityOrder.first) # First-order ...
solver.setSensitivityMethod(amici.SensitivityMethod.adjoint)        # ... adjoint␣
↪sensitivities
```

(continues on next page)

(continued from previous page)

```
model.requireSensitivitiesForAllParameters()              # ... w.r.t. all parameters

# compute adjoint sensitivities
rdata = amici.runAmiciSimulation(model, solver, edata)
#print(rdata['sigmay'])
print('Log-likelihood: %f\nGradient: %s' % (rdata['llh'], rdata['sllh']))
```

```
Log-likelihood: -1190.452734
Gradient: [-8.18063367e+01 -7.40378749e+01  1.87640047e+02  2.07890554e+01
  2.62573207e+02  1.77402064e-01  1.15646253e+01  2.11221869e+04]
```

Finite differences gradient check

Compare AMICI-computed gradient with finite differences

[19]:
```
from scipy.optimize import check_grad

def func(x0, symbol='llh', x0full=None, plist=[], verbose=False):
    p = x0[:]
    if len(plist):
        p = x0full[:]
        p[plist] = x0
    verbose and print('f: p=%s' % p)

    old_parameters = model.getParameters()
    solver.setSensitivityOrder(amici.SensitivityOrder.none)
    model.setParameters(p)
    rdata = amici.runAmiciSimulation(model, solver, edata)

    model.setParameters(old_parameters)

    res = np.sum(rdata[symbol])
    verbose and print(res)
    return res

def grad(x0, symbol='llh', x0full=None, plist=[], verbose=False):
    p = x0[:]
    if len(plist):
        model.setParameterList(plist)
        p = x0full[:]
        p[plist] = x0
    else:
        model.requireSensitivitiesForAllParameters()
    verbose and print('g: p=%s' % p)

    old_parameters = model.getParameters()
    solver.setSensitivityMethod(amici.SensitivityMethod.forward)
    solver.setSensitivityOrder(amici.SensitivityOrder.first)
    model.setParameters(p)
    rdata = amici.runAmiciSimulation(model, solver, edata)
```

(continues on next page)

(continued from previous page)

```
    model.setParameters(old_parameters)

    res = rdata['s%s' % symbol]
    if not isinstance(res, float):
        if len(res.shape) == 3:
            res = np.sum(res, axis=(0, 2))
    verbose and print(res)
    return res

epsilon = 1e-4
err_norm = check_grad(func, grad, p_orig, 'llh', epsilon=epsilon)
print('sllh: |error|_2: %f' % err_norm)
# assert err_norm < 1e-6
print()

for ip in range(model.np()):
    plist = [ip]
    p = p_orig.copy()
    err_norm = check_grad(func, grad, p[plist], 'llh', p, [ip], epsilon=epsilon)
    print('sllh: p[%d]: |error|_2: %f' % (ip, err_norm))

print()
for ip in range(model.np()):
    plist = [ip]
    p = p_orig.copy()
    err_norm = check_grad(func, grad, p[plist], 'y', p, [ip], epsilon=epsilon)
    print('sy: p[%d]: |error|_2: %f' % (ip, err_norm))

print()
for ip in range(model.np()):
    plist = [ip]
    p = p_orig.copy()
    err_norm = check_grad(func, grad, p[plist], 'x', p, [ip], epsilon=epsilon)
    print('sx: p[%d]: |error|_2: %f' % (ip, err_norm))

print()
for ip in range(model.np()):
    plist = [ip]
    p = p_orig.copy()
    err_norm = check_grad(func, grad, p[plist], 'sigmay', p, [ip], epsilon=epsilon)
    print('ssigmay: p[%d]: |error|_2: %f' % (ip, err_norm))
```

```
sllh: |error|_2: 31.850873

sllh: p[0]: |error|_2: 0.006287
sllh: p[1]: |error|_2: 0.016510
sllh: p[2]: |error|_2: 0.017028
sllh: p[3]: |error|_2: 0.009608
sllh: p[4]: |error|_2: 0.083404
sllh: p[5]: |error|_2: 0.000280
```

(continues on next page)

(continued from previous page)

```
sllh: p[6]: |error|_2: 0.001050
sllh: p[7]: |error|_2: 31.850739

sy: p[0]: |error|_2: 0.002974
sy: p[1]: |error|_2: 0.002717
sy: p[2]: |error|_2: 0.001308
sy: p[3]: |error|_2: 0.000939
sy: p[4]: |error|_2: 0.006106
sy: p[5]: |error|_2: 0.000000
sy: p[6]: |error|_2: 0.000000
sy: p[7]: |error|_2: 0.000000

sx: p[0]: |error|_2: 0.001033
sx: p[1]: |error|_2: 0.001076
sx: p[2]: |error|_2: 0.000121
sx: p[3]: |error|_2: 0.000439
sx: p[4]: |error|_2: 0.001569
sx: p[5]: |error|_2: 0.000000
sx: p[6]: |error|_2: 0.000000
sx: p[7]: |error|_2: 0.000000

ssigmay: p[0]: |error|_2: 0.000000
ssigmay: p[1]: |error|_2: 0.000000
ssigmay: p[2]: |error|_2: 0.000000
ssigmay: p[3]: |error|_2: 0.000000
ssigmay: p[4]: |error|_2: 0.000000
ssigmay: p[5]: |error|_2: 0.000000
ssigmay: p[6]: |error|_2: 0.000000
ssigmay: p[7]: |error|_2: 0.000000
```

[20]:

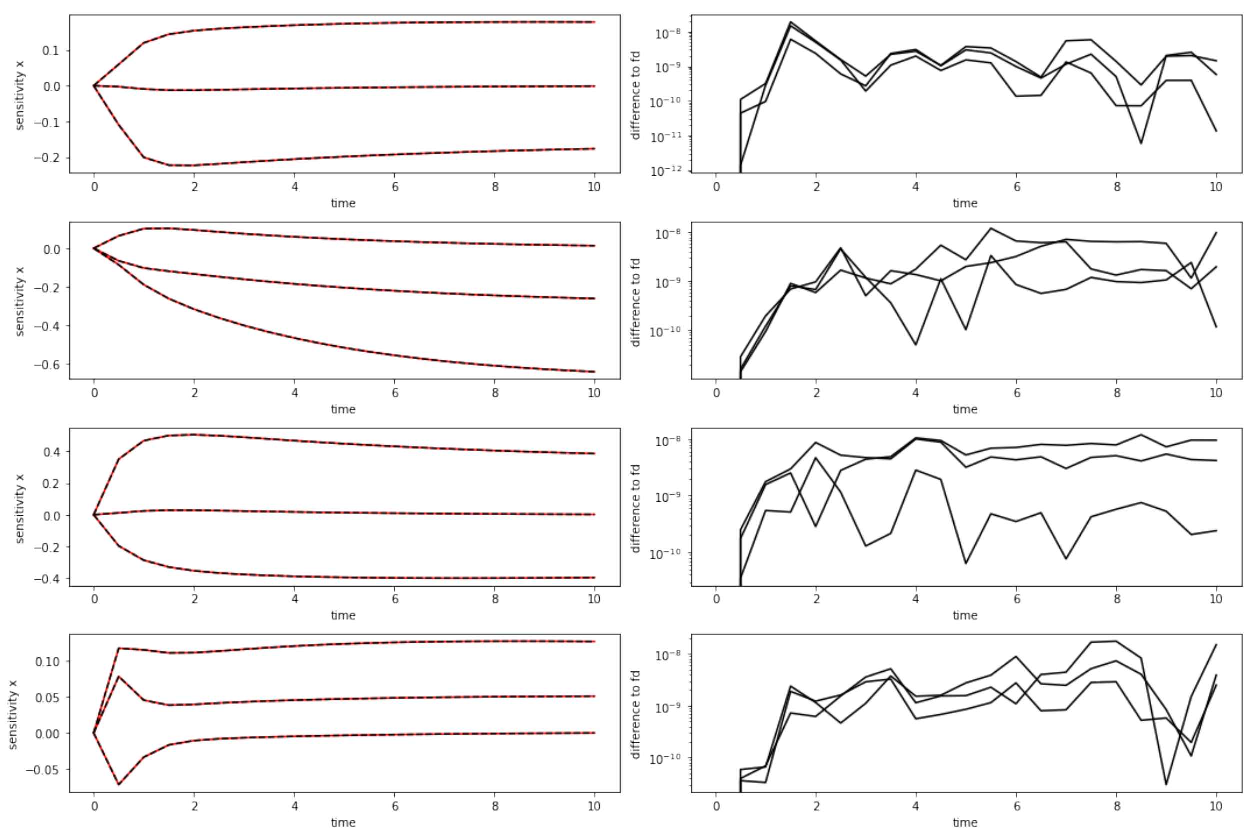
```
eps=1e-4
op=model.getParameters()


solver.setSensitivityMethod(amici.SensitivityMethod.forward) # forward sensitivity␣
↪analysis
solver.setSensitivityOrder(amici.SensitivityOrder.first) # first-order sensitivities
model.requireSensitivitiesForAllParameters()
solver.setRelativeTolerance(1e-12)
rdata = amici.runAmiciSimulation(model, solver, edata)

def fd(x0, ip, eps, symbol='llh'):
    p = list(x0[:])
    old_parameters = model.getParameters()
    solver.setSensitivityOrder(amici.SensitivityOrder.none)
    p[ip]+=eps
    model.setParameters(p)
    rdata_f = amici.runAmiciSimulation(model, solver, edata)
    p[ip]-=2*eps
    model.setParameters(p)
    rdata_b = amici.runAmiciSimulation(model, solver, edata)
```

(continues on next page)

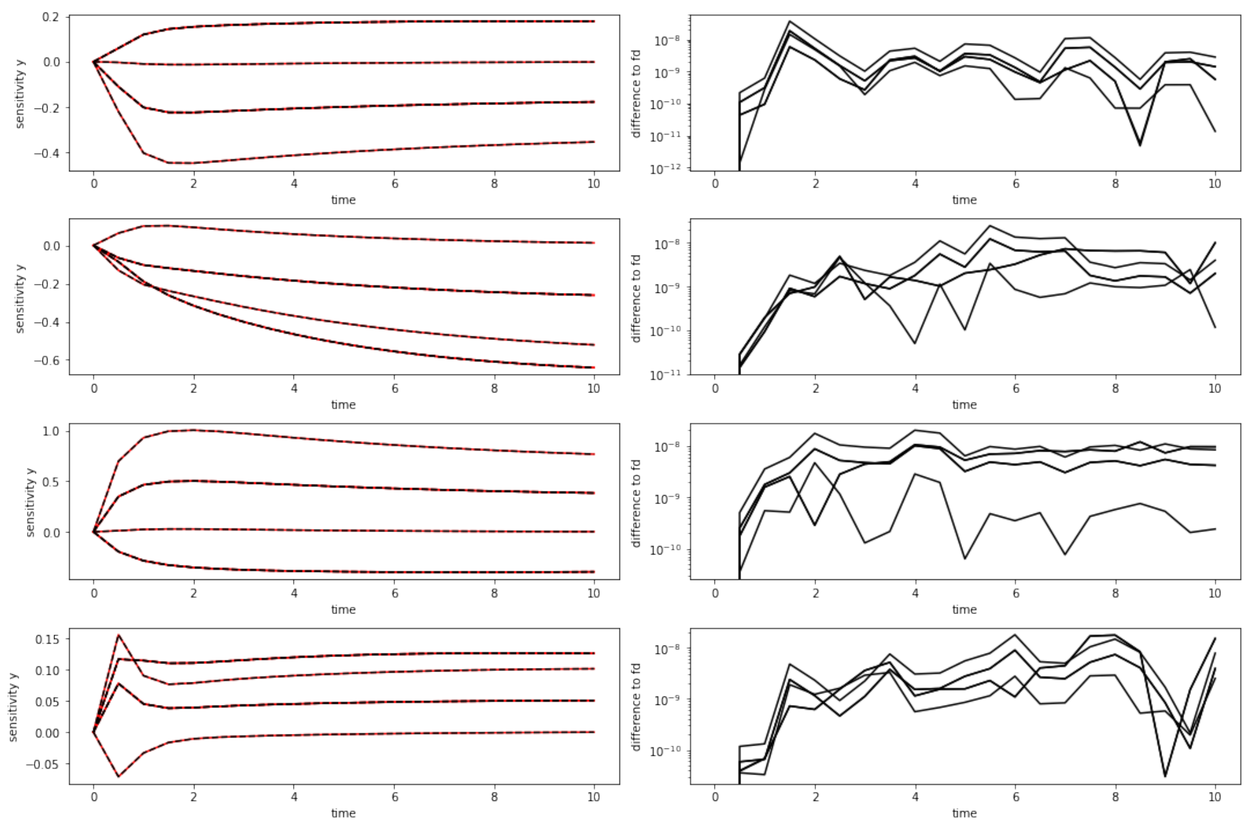
(continued from previous page)

```
    model.setParameters(old_parameters)
    return (rdata_f[symbol]-rdata_b[symbol])/(2*eps)

def plot_sensitivities(symbol, eps):
    fig, axes = plt.subplots(4,2, figsize=(15,10))
    for ip in range(4):
        fd_approx = fd(model.getParameters(), ip, eps, symbol=symbol)

        axes[ip,0].plot(edata.getTimepoints(), rdata[f's{symbol}'][:,ip,:], 'r-')
        axes[ip,0].plot(edata.getTimepoints(), fd_approx, 'k--')
        axes[ip,0].set_ylabel(f'sensitivity {symbol}')
        axes[ip,0].set_xlabel('time')


        axes[ip,1].plot(edata.getTimepoints(), np.abs(rdata[f's{symbol}'][:,ip,:]-fd_approx), 'k-')
        axes[ip,1].set_ylabel('difference to fd')
        axes[ip,1].set_xlabel('time')
        axes[ip,1].set_yscale('log')

    plt.tight_layout()
    plt.show()
```

[21]:
```
plot_sensitivities('x', eps)
```



[22]:
```
plot_sensitivities('y', eps)
```



Export as DataFrame

Experimental data and simulation results can both be exported as pandas Dataframe to allow for an easier inspection of numeric values

[23]:
```
# run the simulation
rdata = amici.runAmiciSimulation(model, solver, edata)
```

[24]:
```
# look at the ExpData as DataFrame
df = amici.getDataObservablesAsDataFrame(model, [edata])
df
```

[24]:
```
   time datatype  t_presim   k0  k0_preeq  k0_presim  observable_x1  \
0   0.0     data       0.0  1.0       NaN        NaN      -1.191094
1   0.5     data       0.0  1.0       NaN        NaN      -1.599410
2   1.0     data       0.0  1.0       NaN        NaN       1.522095
3   1.5     data       0.0  1.0       NaN        NaN       2.455856
4   2.0     data       0.0  1.0       NaN        NaN      -0.600864
5   2.5     data       0.0  1.0       NaN        NaN       1.422341
6   3.0     data       0.0  1.0       NaN        NaN      -0.672523
7   3.5     data       0.0  1.0       NaN        NaN       2.278515
8   4.0     data       0.0  1.0       NaN        NaN       0.078411
9   4.5     data       0.0  1.0       NaN        NaN       0.074017
```

(continues on next page)

(continued from previous page)

```
10   5.0     data       0.0  1.0       NaN       NaN       0.537820
11   5.5     data       0.0  1.0       NaN       NaN       0.498204
12   6.0     data       0.0  1.0       NaN       NaN       2.345616
13   6.5     data       0.0  1.0       NaN       NaN       0.995109
14   7.0     data       0.0  1.0       NaN       NaN       1.363276
15   7.5     data       0.0  1.0       NaN       NaN       0.190180
16   8.0     data       0.0  1.0       NaN       NaN      -0.362771
17   8.5     data       0.0  1.0       NaN       NaN       0.884408
18   9.0     data       0.0  1.0       NaN       NaN      -1.554260
19   9.5     data       0.0  1.0       NaN       NaN       0.492781
20  10.0     data       0.0  1.0       NaN       NaN      -0.964663

    observable_x2  observable_x3  observable_x1_scaled  \
0        0.281469       0.033354              1.645144
1        1.140165       1.368902              1.535490
2        2.020846       1.109229              0.594295
3        0.961785       0.052925              1.839501
4        0.021845       0.220294             -0.110990
5        0.045064       1.361070              0.945899
6        0.873615       1.560601              2.677974
7       -0.005105       0.799689              1.240916
8        0.979531       1.745795              0.858134
9        0.226267       0.604246             -0.091356
10       1.233766       0.194859             -0.345351
11       3.218945       0.190730              0.462838
12       0.956306      -1.770517              0.761754
13      -0.151091       0.805937              2.929645
14       1.622225       0.030980             -0.082327
15       0.962294       1.425918              1.106591
16      -0.251576       0.540826              1.507657
17       0.930074       0.510605              2.012371
18       1.449906      -1.477985              0.871124
19       0.700873       0.242116              0.279648
20      -0.485664       2.087815              2.328707

    observable_x2_offsetted  observable_x1withsigma  observable_x1_std  \
0                  3.900680               -0.518642                1.0
1                  5.678167                0.694825                1.0
2                  3.230374                1.275482                1.0
3                  2.783796                0.085480                1.0
4                  5.003351                2.020780                1.0
5                  4.589646               -0.593535                1.0
6                  2.187698                1.054870                1.0
7                  4.268726                0.491755                1.0
8                  4.609529                0.620844                1.0
9                  3.652345                0.261807                1.0
10                 4.477322                3.378227                1.0
11                 5.946970                1.310100                1.0
12                 3.894865                1.546055                1.0
13                 4.348557                0.234915                1.0
14                 3.761730                0.724608                1.0
15                 4.175984                0.245791                1.0
```

(continues on next page)

(continued from previous page)

```
16             3.871633             2.260562                1.0
17             2.593086            -0.654355                1.0
18             5.497123             0.232772                1.0
19             4.236658            -0.889212                1.0
20             4.787617             0.395106                1.0

    observable_x2_std  observable_x3_std  observable_x1_scaled_std  \
0                 1.0                1.0                       1.0
1                 1.0                1.0                       1.0
2                 1.0                1.0                       1.0
3                 1.0                1.0                       1.0
4                 1.0                1.0                       1.0
5                 1.0                1.0                       1.0
6                 1.0                1.0                       1.0
7                 1.0                1.0                       1.0
8                 1.0                1.0                       1.0
9                 1.0                1.0                       1.0
10                1.0                1.0                       1.0
11                1.0                1.0                       1.0
12                1.0                1.0                       1.0
13                1.0                1.0                       1.0
14                1.0                1.0                       1.0
15                1.0                1.0                       1.0
16                1.0                1.0                       1.0
17                1.0                1.0                       1.0
18                1.0                1.0                       1.0
19                1.0                1.0                       1.0
20                1.0                1.0                       1.0

    observable_x2_offsetted_std  observable_x1withsigma_std
0                           1.0                         NaN
1                           1.0                         NaN
2                           1.0                         NaN
3                           1.0                         NaN
4                           1.0                         NaN
5                           1.0                         NaN
6                           1.0                         NaN
7                           1.0                         NaN
8                           1.0                         NaN
9                           1.0                         NaN
10                          1.0                         NaN
11                          1.0                         NaN
12                          1.0                         NaN
13                          1.0                         NaN
14                          1.0                         NaN
15                          1.0                         NaN
16                          1.0                         NaN
17                          1.0                         NaN
18                          1.0                         NaN
19                          1.0                         NaN
20                          1.0                         NaN
```

[25]:
```
# from the exported dataframe, we can actually reconstruct a copy of the ExpData instance
reconstructed_edata = amici.getEdataFromDataFrame(model, df)
```

[26]:
```
# look at the States in rdata as DataFrame
amici.getResidualsAsDataFrame(model, [edata], [rdata])
```

[26]:
```
    time  t_presim   k0  k0_preeq  k0_presim  observable_x1  observable_x2  \
0    0.0       0.0  1.0       NaN        NaN       1.291094       0.118531
1    0.5       0.0  1.0       NaN        NaN       2.138777       0.455486
2    1.0       0.0  1.0       NaN        NaN       0.942023       1.287558
3    1.5       0.0  1.0       NaN        NaN       1.885457       0.231133
4    2.0       0.0  1.0       NaN        NaN       1.161398       0.693991
5    2.5       0.0  1.0       NaN        NaN       0.869285       0.653687
6    3.0       0.0  1.0       NaN        NaN       1.219394       0.191652
7    3.5       0.0  1.0       NaN        NaN       1.737155       0.671214
8    4.0       0.0  1.0       NaN        NaN       0.457870       0.328229
9    4.5       0.0  1.0       NaN        NaN       0.457522       0.411248
10   5.0       0.0  1.0       NaN        NaN       0.010728       0.609084
11   5.5       0.0  1.0       NaN        NaN       0.024710       2.606212
12   6.0       0.0  1.0       NaN        NaN       1.826626       0.354703
13   6.5       0.0  1.0       NaN        NaN       0.479809       0.742320
14   7.0       0.0  1.0       NaN        NaN       0.851446       1.040670
15   7.5       0.0  1.0       NaN        NaN       0.318388       0.389765
16   8.0       0.0  1.0       NaN        NaN       0.868271       0.815679
17   8.5       0.0  1.0       NaN        NaN       0.381793       0.373840
18   9.0       0.0  1.0       NaN        NaN       2.054162       0.901024
19   9.5       0.0  1.0       NaN        NaN       0.004569       0.158864
20  10.0       0.0  1.0       NaN        NaN       1.459612       1.021246

    observable_x3  observable_x1_scaled  observable_x2_offsetted  \
0        0.666646              1.445144                 0.500680
1        1.177412              0.456756                 1.993488
2        1.012805              0.565850                 0.502913
3        0.023151              0.698703                 0.946856
4        0.150600              1.232059                 1.287515
5        1.294769              0.160213                 0.890895
6        1.496868              1.584233                 1.494264
7        0.738182              0.158196                 0.602617
8        1.686301              0.214427                 0.958227
9        0.546593              1.154432                 0.014831
10       0.138899              1.399534                 0.852641
11       0.136330              0.582991                 2.334237
12       1.823477              0.276225                 0.293262
13       0.754307              1.899046                 0.757328
14       0.019418              1.105987                 0.180175
15       1.376658              0.089456                 0.603455
16       0.492622              0.496656                 0.307529
17       0.463381              1.007140                 0.963148
18       1.524300              0.128680                 1.948242
19       0.196646              0.715052                 0.694649
20       2.043129              1.338809                 1.252036

    observable_x1withsigma
```

(continues on next page)

(continued from previous page)

```
0            6.186425
1            1.554579
2            6.954092
3            4.849190
4           14.602456
5           11.465911
6            5.079991
7            0.496051
8            0.845638
9            2.697316
10          28.511357
11           7.871853
12          10.270655
13           2.803840
14           2.127777
15           2.627767
16          17.550614
17          11.569706
18           2.671303
19          13.865618
20           0.998435
```

[27]:

```
# look at the Observables in rdata as DataFrame
amici.getSimulationObservablesAsDataFrame(model, [edata], [rdata])
```

[27]:

```
    time    datatype  t_presim   k0  k0_preeq  k0_presim  observable_x1  \
0    0.0  simulation       0.0  1.0       NaN        NaN       0.100000
1    0.5  simulation       0.0  1.0       NaN        NaN       0.539367
2    1.0  simulation       0.0  1.0       NaN        NaN       0.580072
3    1.5  simulation       0.0  1.0       NaN        NaN       0.570399
4    2.0  simulation       0.0  1.0       NaN        NaN       0.560535
5    2.5  simulation       0.0  1.0       NaN        NaN       0.553056
6    3.0  simulation       0.0  1.0       NaN        NaN       0.546871
7    3.5  simulation       0.0  1.0       NaN        NaN       0.541360
8    4.0  simulation       0.0  1.0       NaN        NaN       0.536280
9    4.5  simulation       0.0  1.0       NaN        NaN       0.531538
10   5.0  simulation       0.0  1.0       NaN        NaN       0.527091
11   5.5  simulation       0.0  1.0       NaN        NaN       0.522914
12   6.0  simulation       0.0  1.0       NaN        NaN       0.518989
13   6.5  simulation       0.0  1.0       NaN        NaN       0.515299
14   7.0  simulation       0.0  1.0       NaN        NaN       0.511830
15   7.5  simulation       0.0  1.0       NaN        NaN       0.508568
16   8.0  simulation       0.0  1.0       NaN        NaN       0.505500
17   8.5  simulation       0.0  1.0       NaN        NaN       0.502615
18   9.0  simulation       0.0  1.0       NaN        NaN       0.499902
19   9.5  simulation       0.0  1.0       NaN        NaN       0.497350
20  10.0  simulation       0.0  1.0       NaN        NaN       0.494949

    observable_x2  observable_x3  observable_x1_scaled  \
0        0.400000       0.700000              0.200000
1        0.684679       0.191491              1.078734
2        0.733287       0.096424              1.160145
```

(continues on next page)

(continued from previous page)

```
3       0.730652       0.076076              1.140799
4       0.715836       0.069694              1.121069
5       0.698751       0.066301              1.106112
6       0.681963       0.063733              1.093741
7       0.666109       0.061506              1.082720
8       0.651302       0.059495              1.072561
9       0.637515       0.057653              1.063076
10      0.624681       0.055960              1.054183
11      0.612733       0.054400              1.045829
12      0.601603       0.052960              1.037978
13      0.591229       0.051629              1.030598
14      0.581555       0.050399              1.023660
15      0.572529       0.049259              1.017136
16      0.564103       0.048203              1.011000
17      0.556234       0.047224              1.005231
18      0.548881       0.046315              0.999804
19      0.542008       0.045471              0.994700
20      0.535581       0.044686              0.989898

    observable_x2_offsetted  observable_x1withsigma  observable_x1_std  \
0                  3.400000                0.100000                1.0
1                  3.684679                0.539367                1.0
2                  3.733287                0.580072                1.0
3                  3.730652                0.570399                1.0
4                  3.715836                0.560535                1.0
5                  3.698751                0.553056                1.0
6                  3.681963                0.546871                1.0
7                  3.666109                0.541360                1.0
8                  3.651302                0.536280                1.0
9                  3.637515                0.531538                1.0
10                 3.624681                0.527091                1.0
11                 3.612733                0.522914                1.0
12                 3.601603                0.518989                1.0
13                 3.591229                0.515299                1.0
14                 3.581555                0.511830                1.0
15                 3.572529                0.508568                1.0
16                 3.564103                0.505500                1.0
17                 3.556234                0.502615                1.0
18                 3.548881                0.499902                1.0
19                 3.542008                0.497350                1.0
20                 3.535581                0.494949                1.0

    observable_x2_std  observable_x3_std  observable_x1_scaled_std  \
0                 1.0                1.0                       1.0
1                 1.0                1.0                       1.0
2                 1.0                1.0                       1.0
3                 1.0                1.0                       1.0
4                 1.0                1.0                       1.0
5                 1.0                1.0                       1.0
6                 1.0                1.0                       1.0
7                 1.0                1.0                       1.0
8                 1.0                1.0                       1.0
```

(continues on next page)

(continued from previous page)

```
9                  1.0                1.0                        1.0
10                 1.0                1.0                        1.0
11                 1.0                1.0                        1.0
12                 1.0                1.0                        1.0
13                 1.0                1.0                        1.0
14                 1.0                1.0                        1.0
15                 1.0                1.0                        1.0
16                 1.0                1.0                        1.0
17                 1.0                1.0                        1.0
18                 1.0                1.0                        1.0
19                 1.0                1.0                        1.0
20                 1.0                1.0                        1.0

    observable_x2_offsetted_std  observable_x1withsigma_std
0                           1.0                         0.1
1                           1.0                         0.1
2                           1.0                         0.1
3                           1.0                         0.1
4                           1.0                         0.1
5                           1.0                         0.1
6                           1.0                         0.1
7                           1.0                         0.1
8                           1.0                         0.1
9                           1.0                         0.1
10                          1.0                         0.1
11                          1.0                         0.1
12                          1.0                         0.1
13                          1.0                         0.1
14                          1.0                         0.1
15                          1.0                         0.1
16                          1.0                         0.1
17                          1.0                         0.1
18                          1.0                         0.1
19                          1.0                         0.1
20                          1.0                         0.1
```

[28]:
```
# look at the States in rdata as DataFrame
amici.getSimulationStatesAsDataFrame(model, [edata], [rdata])
```

[28]:
```
    time  t_presim   k0  k0_preeq  k0_presim        x1        x2        x3
0    0.0       0.0  1.0       NaN        NaN  0.100000  0.400000  0.700000
1    0.5       0.0  1.0       NaN        NaN  0.539367  0.684679  0.191491
2    1.0       0.0  1.0       NaN        NaN  0.580072  0.733287  0.096424
3    1.5       0.0  1.0       NaN        NaN  0.570399  0.730652  0.076076
4    2.0       0.0  1.0       NaN        NaN  0.560535  0.715836  0.069694
5    2.5       0.0  1.0       NaN        NaN  0.553056  0.698751  0.066301
6    3.0       0.0  1.0       NaN        NaN  0.546871  0.681963  0.063733
7    3.5       0.0  1.0       NaN        NaN  0.541360  0.666109  0.061506
8    4.0       0.0  1.0       NaN        NaN  0.536280  0.651302  0.059495
9    4.5       0.0  1.0       NaN        NaN  0.531538  0.637515  0.057653
10   5.0       0.0  1.0       NaN        NaN  0.527091  0.624681  0.055960
```

(continues on next page)

(continued from previous page)

```
11   5.5        0.0  1.0       NaN        NaN  0.522914  0.612733  0.054400
12   6.0        0.0  1.0       NaN        NaN  0.518989  0.601603  0.052960
13   6.5        0.0  1.0       NaN        NaN  0.515299  0.591229  0.051629
14   7.0        0.0  1.0       NaN        NaN  0.511830  0.581555  0.050399
15   7.5        0.0  1.0       NaN        NaN  0.508568  0.572529  0.049259
16   8.0        0.0  1.0       NaN        NaN  0.505500  0.564103  0.048203
17   8.5        0.0  1.0       NaN        NaN  0.502615  0.556234  0.047224
18   9.0        0.0  1.0       NaN        NaN  0.499902  0.548881  0.046315
19   9.5        0.0  1.0       NaN        NaN  0.497350  0.542008  0.045471
20  10.0        0.0  1.0       NaN        NaN  0.494949  0.535581  0.044686
```

Using PEtab

This notebook illustrates how to use PEtab with AMICI.

[1]:
```
from amici.petab_import import import_petab_problem
from amici.petab_objective import simulate_petab
import petab

import os
```

We use an example model from the benchmark collection:

[2]:
```
!git clone --depth 1 https://github.com/Benchmarking-Initiative/Benchmark-Models-PEtab.
↪git tmp/benchmark-models || (cd tmp/benchmark-models && git pull)
```

```
Cloning into 'tmp/benchmark-models'...
remote: Enumerating objects: 142, done.
remote: Counting objects: 100% (142/142), done.
remote: Compressing objects: 100% (122/122), done.
remote: Total 142 (delta 41), reused 104 (delta 18), pack-reused 0
Receiving objects: 100% (142/142), 648.29 KiB | 1.23 MiB/s, done.
Resolving deltas: 100% (41/41), done.
```

[3]:
```
folder_base = "tmp/benchmark-models/Benchmark-Models/"
!ls -l $folder_base
```

```
total 68
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Alkan_SciSignal2018
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Beer_MolBioSystems2014
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Boehm_JProteomeRes2014
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Borghans_BiophysChem1997
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Brannmark_JBC2010
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Bruno_JExpBio2016
-rwxr-xr-x 1 yannik yannik  654 Mär 17 15:27 checkBenchmarkModels.py
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Chen_MSB2009
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Crauste_CellSystems2017
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Elowitz_Nature2000
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Fiedler_BMC2016
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Fujita_SciSignal2010
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Perelson_Science1996
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Rahman_MBS2016
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Sneyd_PNAS2002
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Weber_BMC2015
drwxr-xr-x 2 yannik yannik 4096 Mär 17 15:27 Zheng_PNAS2012
```

We import a model to PEtab from a provided yaml file:

```
[4]: model_name = "Boehm_JProteomeRes2014"
yaml_file = os.path.join(folder_base, model_name, model_name + ".yaml")
petab_problem = petab.Problem.from_yaml(yaml_file)
```

Next, we import the model to amici, compile it and obtain a function handle:

```
[5]: amici_model = import_petab_problem(petab_problem)
```

```
2020-03-17 15:27:27.586 - amici.petab_import - INFO - Importing model ...
2020-03-17 15:27:27.593 - amici.petab_import - INFO - Model name is 'Boehm_
↪JProteomeRes2014'. Writing model code to '/home/yannik/amici/python/examples/amici_
↪models/Boehm_JProteomeRes2014'.
2020-03-17 15:27:27.598 - amici.petab_import - INFO - Species: 8
2020-03-17 15:27:27.599 - amici.petab_import - INFO - Global parameters: 9
2020-03-17 15:27:27.599 - amici.petab_import - INFO - Reactions: 9
2020-03-17 15:27:27.715 - amici.petab_import - INFO - Observables: 3
2020-03-17 15:27:27.715 - amici.petab_import - INFO - Sigmas: 3
2020-03-17 15:27:27.722 - amici.petab_import - DEBUG - Adding output parameters to model:
↪ OrderedDict([('noiseParameter1_pSTAT5A_rel', None), ('noiseParameter1_pSTAT5B_rel',␣
↪None), ('noiseParameter1_rSTAT5A_rel', None)])
2020-03-17 15:27:27.725 - amici.petab_import - DEBUG - Condition table: (1, 1)
2020-03-17 15:27:27.726 - amici.petab_import - DEBUG - Fixed parameters are []
2020-03-17 15:27:27.728 - amici.petab_import - INFO - Overall fixed parameters: 0
2020-03-17 15:27:27.729 - amici.petab_import - INFO - Variable parameters: 12
2020-03-17 15:27:27.735 - amici.sbml_import - INFO - Finished processing SBML parameters␣
↪             (1.25E-03s)
2020-03-17 15:27:27.749 - amici.sbml_import - INFO - Finished processing SBML species   ␣
↪             (1.26E-02s)
2020-03-17 15:27:27.829 - amici.sbml_import - INFO - Finished processing SBML reactions ␣
↪             (7.41E-02s)
2020-03-17 15:27:27.833 - amici.sbml_import - INFO - Finished processing SBML␣
↪compartments           (4.23E-04s)
2020-03-17 15:27:27.898 - amici.sbml_import - INFO - Finished processing SBML rules     ␣
↪             (6.47E-02s)
2020-03-17 15:27:28.012 - amici.sbml_import - INFO - Finished processing SBML␣
↪observables            (6.77E-02s)
2020-03-17 15:27:28.139 - amici.ode_export - INFO - Finished writing J.cpp              ␣
↪             (1.14E-01s)
2020-03-17 15:27:28.160 - amici.ode_export - INFO - Finished writing JB.cpp             ␣
↪             (2.04E-02s)
2020-03-17 15:27:28.167 - amici.ode_export - INFO - Finished writing JDiag.cpp          ␣
↪             (6.41E-03s)
2020-03-17 15:27:28.187 - amici.ode_export - INFO - Finished writing JSparse.cpp        ␣
↪             (1.91E-02s)
2020-03-17 15:27:28.217 - amici.ode_export - INFO - Finished writing JSparseB.cpp       ␣
↪             (2.73E-02s)
```

(continues on next page)

(continued from previous page)

```
2020-03-17 15:27:28.236 - amici.ode_export - INFO - Finished writing Jy.cpp               ␣
↪             (1.65E-02s)
2020-03-17 15:27:28.344 - amici.ode_export - INFO - Finished writing dJydsigmay.cpp       ␣
↪             (1.07E-01s)
2020-03-17 15:27:28.389 - amici.ode_export - INFO - Finished writing dJydy.cpp            ␣
↪             (3.99E-02s)
2020-03-17 15:27:28.466 - amici.ode_export - INFO - Finished writing dwdp.cpp             ␣
↪             (7.61E-02s)
2020-03-17 15:27:28.473 - amici.ode_export - INFO - Finished writing dwdx.cpp             ␣
↪             (5.87E-03s)
2020-03-17 15:27:28.497 - amici.ode_export - INFO - Finished writing dxdotdw.cpp          ␣
↪             (2.32E-02s)
2020-03-17 15:27:28.533 - amici.ode_export - INFO - Finished writing dxdotdp_explicit.
↪cpp             (3.38E-02s)
2020-03-17 15:27:28.756 - amici.ode_export - INFO - Finished writing dydx.cpp             ␣
↪             (1.98E-01s)
2020-03-17 15:27:28.910 - amici.ode_export - INFO - Finished writing dydp.cpp             ␣
↪             (1.53E-01s)
2020-03-17 15:27:28.926 - amici.ode_export - INFO - Finished writing dsigmaydp.cpp        ␣
↪             (1.40E-02s)
2020-03-17 15:27:28.931 - amici.ode_export - INFO - Finished writing sigmay.cpp           ␣
↪             (2.46E-03s)
2020-03-17 15:27:28.950 - amici.ode_export - INFO - Finished writing w.cpp                ␣
↪             (1.55E-02s)
2020-03-17 15:27:28.967 - amici.ode_export - INFO - Finished writing x0.cpp               ␣
↪             (1.57E-02s)
2020-03-17 15:27:28.975 - amici.ode_export - INFO - Finished writing x0_fixedParameters.
↪cpp           (4.78E-03s)
2020-03-17 15:27:29.027 - amici.ode_export - INFO - Finished writing sx0.cpp              ␣
↪             (5.01E-02s)
2020-03-17 15:27:29.069 - amici.ode_export - INFO - Finished writing sx0_fixedParameters.
↪cpp          (3.14E-02s)
2020-03-17 15:27:29.104 - amici.ode_export - INFO - Finished writing xdot.cpp             ␣
↪             (3.43E-02s)
2020-03-17 15:27:29.129 - amici.ode_export - INFO - Finished writing y.cpp                ␣
↪             (2.16E-02s)
2020-03-17 15:27:29.136 - amici.ode_export - INFO - Finished writing x_rdata.cpp          ␣
↪             (4.95E-03s)
2020-03-17 15:27:29.138 - amici.ode_export - INFO - Finished writing total_cl.cpp         ␣
↪             (6.59E-04s)
2020-03-17 15:27:29.147 - amici.ode_export - INFO - Finished writing x_solver.cpp         ␣
↪             (7.72E-03s)
2020-03-17 15:27:29.166 - amici.ode_export - INFO - Finished generating cpp code          ␣
↪             (1.14E+00s)
2020-03-17 15:27:46.200 - amici.ode_export - INFO - Finished compiling cpp code           ␣
↪             (1.70E+01s)
2020-03-17 15:27:46.204 - amici.petab_import - INFO - Finished Importing PEtab model      ␣
↪             (1.86E+01s)
2020-03-17 15:27:46.209 - amici.petab_import - INFO - Successfully loaded model Boehm_
↪JProteomeRes2014 from /home/yannik/amici/python/examples/amici_models/Boehm_
↪JProteomeRes2014.
```

```
running build_ext
building 'Boehm_JProteomeRes2014._Boehm_JProteomeRes2014' extension
swigging swig/Boehm_JProteomeRes2014.i to swig/Boehm_JProteomeRes2014_wrap.cpp
swig -python -c++ -modern -outdir Boehm_JProteomeRes2014 -I/home/yannik/amici/python/
↪sdist/amici/swig -I/home/yannik/amici/python/sdist/amici/include -o swig/Boehm_
↪JProteomeRes2014_wrap.cpp swig/Boehm_JProteomeRes2014.i
creating build
creating build/temp.linux-x86_64-3.7
creating build/temp.linux-x86_64-3.7/swig
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c swig/Boehm_JProteomeRes2014_wrap.cpp -o build/temp.linux-x86_64-
↪3.7/swig/Boehm_JProteomeRes2014_wrap.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdw.cpp -o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_dxdotdw.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_total_cl.cpp -o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_total_cl.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_x_rdata.cpp -o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_x_rdata.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdp_implicit_colptrs.cpp -o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_implicit_colptrs.o -std=c++14
```

(continues on next page)

(continued from previous page)

```
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dsigmaydp.cpp -o build/temp.linux-x86_64-
↪3.7/Boehm_JProteomeRes2014_dsigmaydp.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_y.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_y.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dydp.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dydp.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_w.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_w.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JSparseB_rowvals.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_JSparseB_rowvals.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
```

(continues on next page)

(continued from previous page)

```
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdw_rowvals.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_dxdotdw_rowvals.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dwdx_rowvals.cpp -o build/temp.linux-x86_
↪64-3.7/Boehm_JProteomeRes2014_dwdx_rowvals.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_x0.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_x0.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dwdx.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dwdx.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dJydy_colptrs.cpp -o build/temp.linux-x86_
↪64-3.7/Boehm_JProteomeRes2014_dJydy_colptrs.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JSparseB.cpp -o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_JSparseB.o -std=c++14
```

(continues on next page)

(continued from previous page)

```
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JSparseB_colptrs.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_JSparseB_colptrs.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdp_explicit_colptrs.cpp -o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_explicit_colptrs.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_sx0_fixedParameters.cpp -o build/temp.
↪linux-x86_64-3.7/Boehm_JProteomeRes2014_sx0_fixedParameters.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JSparse_rowvals.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_JSparse_rowvals.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdp_explicit.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_explicit.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
```

(continues on next page)

(continued from previous page)

```
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dJydy.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dJydy.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dwdp_colptrs.cpp -o build/temp.linux-x86_
↪64-3.7/Boehm_JProteomeRes2014_dwdp_colptrs.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_x0_fixedParameters.cpp -o build/temp.
↪linux-x86_64-3.7/Boehm_JProteomeRes2014_x0_fixedParameters.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdw_colptrs.cpp -o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_dxdotdw_colptrs.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dJydsigmay.cpp -o build/temp.linux-x86_64-
↪3.7/Boehm_JProteomeRes2014_dJydsigmay.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdp_implicit_rowvals.cpp -o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_implicit_rowvals.o -std=c++14
```

(continues on next page)

(continued from previous page)

```
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dwdp.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dwdp.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_sx0.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_sx0.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JB.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_JB.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dwdx_colptrs.cpp -o build/temp.linux-x86_
↪64-3.7/Boehm_JProteomeRes2014_dwdx_colptrs.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c wrapfunctions.cpp -o build/temp.linux-x86_64-3.7/wrapfunctions.o␣
↪-std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
```

(continues on next page)

(continued from previous page)

```
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_x_solver.cpp -o build/temp.linux-x86_64-3.
→7/Boehm_JProteomeRes2014_x_solver.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
→for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_JSparse.cpp -o build/temp.linux-x86_64-3.
→7/Boehm_JProteomeRes2014_JSparse.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
→for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_xdot.cpp -o build/temp.linux-x86_64-3.7/
→Boehm_JProteomeRes2014_xdot.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
→for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_dJydy_rowvals.cpp -o build/temp.linux-x86_
→64-3.7/Boehm_JProteomeRes2014_dJydy_rowvals.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
→for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_dwdp_rowvals.cpp -o build/temp.linux-x86_
→64-3.7/Boehm_JProteomeRes2014_dwdp_rowvals.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
→for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
→DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
→examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
→include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
→python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
→ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
→include/python3.7m -c Boehm_JProteomeRes2014_JSparse_colptrs.cpp -o build/temp.linux-
→x86_64-3.7/Boehm_JProteomeRes2014_JSparse_colptrs.o -std=c++14
```

(continues on next page)

(continued from previous page)

```
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_J.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_J.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dydx.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dydx.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_JDiag.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_JDiag.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_Jy.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_Jy.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_sigmay.cpp -o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_sigmay.o -std=c++14
cc1plus: warning: command line option ‘-Wstrict-prototypes’ is valid for C/ObjC but not␣
↪for C++
```

(continues on next page)

(continued from previous page)

```
gcc -pthread -B /home/yannik/anaconda3/compiler_compat -Wl,--sysroot=/ -Wsign-compare -
↪DNDEBUG -g -fwrapv -O3 -Wall -Wstrict-prototypes -fPIC -I/home/yannik/amici/python/
↪examples/amici_models/Boehm_JProteomeRes2014 -I/home/yannik/amici/python/sdist/amici/
↪include -I/home/yannik/amici/python/sdist/amici/ThirdParty/gsl -I/home/yannik/amici/
↪python/sdist/amici/ThirdParty/sundials/include -I/home/yannik/amici/python/sdist/amici/
↪ThirdParty/SuiteSparse/include -I/usr/include/hdf5/serial -I/home/yannik/anaconda3/
↪include/python3.7m -c Boehm_JProteomeRes2014_dxdotdp_explicit_rowvals.cpp -o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_explicit_rowvals.o -std=c++14
cc1plus: warning: command line option '-Wstrict-prototypes' is valid for C/ObjC but not␣
↪for C++
g++ -pthread -shared -B /home/yannik/anaconda3/compiler_compat -L/home/yannik/anaconda3/
↪lib -Wl,-rpath=/home/yannik/anaconda3/lib -Wl,--no-as-needed -Wl,--sysroot=/ build/
↪temp.linux-x86_64-3.7/swig/Boehm_JProteomeRes2014_wrap.o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dxdotdw.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_
↪total_cl.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_x_rdata.o build/temp.
↪linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_implicit_colptrs.o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_dsigmaydp.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_y.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dydp.o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_w.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_JSparseB_rowvals.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_
↪dxdotdw_rowvals.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dwdx_rowvals.o␣
↪build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_x0.o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dwdx.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dJydy_
↪colptrs.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_JSparseB.o build/temp.
↪linux-x86_64-3.7/Boehm_JProteomeRes2014_JSparseB_colptrs.o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dxdotdp_explicit_colptrs.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_sx0_fixedParameters.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_JSparse_rowvals.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_
↪dxdotdp_explicit.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dJydy.o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dwdp_colptrs.o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_x0_fixedParameters.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_dxdotdw_colptrs.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_
↪dJydsigmay.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_implicit_
↪rowvals.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dwdp.o build/temp.linux-
↪x86_64-3.7/Boehm_JProteomeRes2014_sx0.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_JB.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dwdx_colptrs.
↪o build/temp.linux-x86_64-3.7/wrapfunctions.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_x_solver.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_JSparse.
↪o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_xdot.o build/temp.linux-x86_64-3.
↪7/Boehm_JProteomeRes2014_dJydy_rowvals.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_dwdp_rowvals.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_
↪JSparse_colptrs.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_J.o build/temp.
↪linux-x86_64-3.7/Boehm_JProteomeRes2014_dydx.o build/temp.linux-x86_64-3.7/Boehm_
↪JProteomeRes2014_JDiag.o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_Jy.o build/
↪temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_sigmay.o build/temp.linux-x86_64-3.7/
↪Boehm_JProteomeRes2014_dxdotdp_explicit_rowvals.o -L/usr/lib/x86_64-linux-gnu/hdf5/
↪serial -L/home/yannik/amici/python/sdist/amici/libs -lamici -lsundials -lsuitesparse -
↪lcblas -lhdf5_hl_cpp -lhdf5_hl -lhdf5_cpp -lhdf5 -o /home/yannik/amici/python/examples/
↪amici_models/Boehm_JProteomeRes2014/Boehm_JProteomeRes2014/_Boehm_JProteomeRes2014.
↪cpython-37m-x86_64-linux-gnu.so
```

That's it. Now, we can use the model to perform simulations. For more involved purposes, consider using the objective function provided by pyPESTO. For simple simulations, a function `simulate_petab` is available:

[6]:
```
simulate_petab(petab_problem, amici_model)
```

[6]:
```
{'llh': -138.22199570334107,
 'sllh': None,
 'rdatas': [<amici.numpy.ReturnDataView at 0x7f030e1a0fd0>]}
```

This performs a simulation at the nominal parameters. Parameters can also be directly specified, both scaled and unscaled:

[7]:
```
parameters = {
    x_id: x_val for x_id, x_val in
    zip(petab_problem.x_ids, petab_problem.x_nominal_scaled)
}
simulate_petab(petab_problem, amici_model, problem_parameters=parameters, scaled_
↪parameters=True)
```

[7]:
```
{'llh': -138.22199570334107,
 'sllh': None,
 'rdatas': [<amici.numpy.ReturnDataView at 0x7f030e198590>]}
```

For further information, see the documentation.

AMICI Python example "Experimental Conditions"

In this example we will explore some more options for the initialization of experimental conditions, including how to reset initial conditions based on changing values for `fixedParameters` as well as an additional presimulation phase on top of preequilibration. This notebook is expected to run from the `python/example_presimulation` directory.

[1]:
```
# SBML model we want to import
sbml_file = 'model_presimulation.xml'
# Name of the model that will also be the name of the python module
model_name = 'model_presimulation'
# Directory to which the generated model code is written
model_output_dir = model_name

import libsbml
import amici
import amici.plotting
import os
import sys
import importlib
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pprint import pprint
```

Model Loading

Here we load a simple model of protein phosphorylation that can be inhibited by a drug. This model was created using PySB (see `createModel.py`)

[2]:
```
sbml_reader = libsbml.SBMLReader()
sbml_doc = sbml_reader.readSBML(sbml_file)
sbml_model = sbml_doc.getModel()

print('Species:')
pprint([(s.getId(),s.getName()) for s in sbml_model.getListOfSpecies()])

print('\nReactions:')
for reaction in sbml_model.getListOfReactions():
    reactants = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfReactants()])
    products  = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfProducts()])
    reversible = '<' if reaction.getReversible() else ''
    print('%3s: %10s %1s->%10s\t\t[%s]' % (reaction.getName(),
                        reactants,
                        reversible,
                         products,
                        libsbml.formulaToL3String(reaction.getKineticLaw().getMath())))

print('Parameters:')
pprint([(p.getId(),p.getName()) for p in sbml_model.getListOfParameters()])
```

```
Species:
[('__s0', "PROT(kin=None, drug=None, phospho='u')"),
 ('__s1', 'DRUG(bound=None)'),
 ('__s2', 'KIN(bound=None)'),
 ('__s3', "DRUG(bound=1) ._br_PROT(kin=None, drug=1, phospho='u')"),
 ('__s4', "KIN(bound=1) ._br_PROT(kin=1, drug=None, phospho='u')"),
 ('__s5', "PROT(kin=None, drug=None, phospho='p')")]

Reactions:
PROT_DRUG_bind:   __s0 +  __s1 <->      __s3             [-(koff_prot_drug * __s3) + kon_
↪prot_drug * __s0 * __s1]
PROT_KIN_bind:  __s0 +  __s2  ->       __s4              [kon_prot_kin * __s0 * __s2]
PROT_KIN_phospho:       __s4  -> __s2 +  __s5            [kphospho_prot_kin * __s4]
PROT_dephospho:       __s5  ->       __s0           [kdephospho_prot * __s5]
Parameters:
[('initProt', 'initProt'),
 ('initDrug', 'initDrug'),
 ('initKin', 'initKin'),
 ('pPROT_obs', 'pPROT_obs'),
 ('PROT_0', 'PROT_0'),
 ('DRUG_0', 'DRUG_0'),
 ('KIN_0', 'KIN_0'),
 ('kon_prot_drug', 'kon_prot_drug'),
 ('koff_prot_drug', 'koff_prot_drug'),
 ('kon_prot_kin', 'kon_prot_kin'),
 ('kphospho_prot_kin', 'kphospho_prot_kin'),
```

(continues on next page)

(continued from previous page)

```
 ('kdephospho_prot', 'kdephospho_prot'),
 ('__obs0', 'pPROT'),
 ('__obs1', 'tPROT')]
```

[3]:
```
# Create an SbmlImporter instance for our SBML model
sbml_importer = amici.SbmlImporter(sbml_file)
```

For this example we want specify the initial drug and kinase concentrations as experimental conditions. Accordingly we specify them as `fixedParameters`. The meaning of `fixedParameters` is defined in the Glossary, which we display here for convenience.

[4]:
```
from IPython.display import IFrame
IFrame('https://amici.readthedocs.io/en/latest/glossary.html#term-fixed-parameters',␣
↪width=600, height=175)
```

[4]:
```
<IPython.lib.display.IFrame at 0x10ce15b50>
```

[5]:
```
fixedParameters = ['DRUG_0','KIN_0']
```

The SBML model specifies a single observable named `pPROT` which describes the fraction of phosphorylated Protein. We load this observable using amici.assignmentRules2observables.

[6]:
```
# Retrieve model output names and formulae from AssignmentRules and remove the␣
↪respective rules
observables = amici.assignmentRules2observables(
        sbml_importer.sbml, # the libsbml model object
        filter_function=lambda variable: variable.getName() == 'pPROT'
    )
print('Observables:')
pprint(observables)
```

```
Observables:
{'__obs0': {'formula': '__s5', 'name': 'pPROT'}}
```

Now the model is ready for compilation using sbml2amici. Note that we here pass `fixedParameters` as arguments to `constant_parameters`, which ensures that amici is aware that we want to have them as `fixedParameters`:

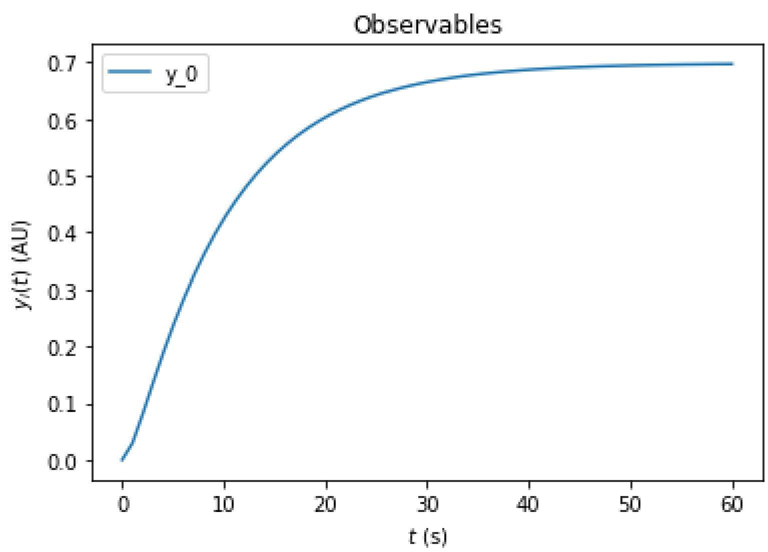
[7]:
```
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         verbose=False,
                         observables=observables,
                         constant_parameters=fixedParameters)
# load the generated module
model_module = amici.import_model_module(model_name, model_output_dir)
```

To simulate the model we need to create an instance via the `getModel()` method in the generated model module.

[8]:
```
# Create Model instance
model = model_module.getModel()

# Create solver instance
solver = model.getSolver()
```

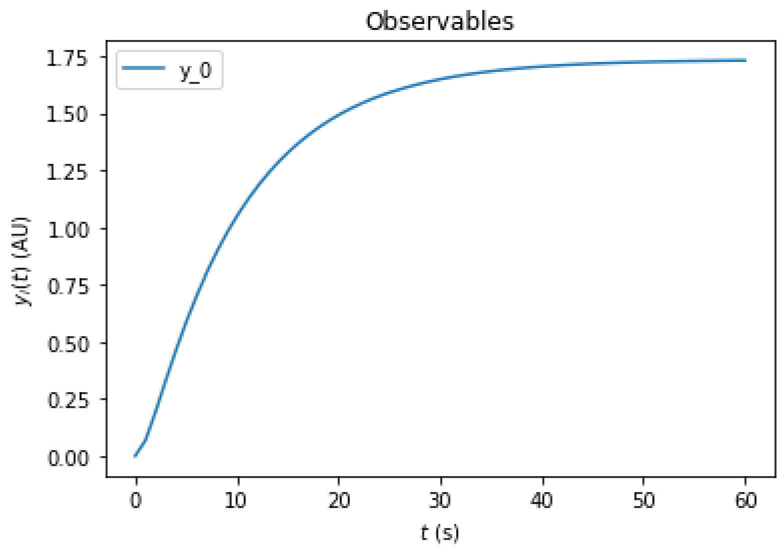
The only thing we need to simulate the model is a timepoint vector, which can be specified using the setTimepoints method. If we do not specify any additional options, the default values for `fixedParameters` and `parameters` that were specified in the SBML file will be used.

[9]:
```
# Run simulation using default model parameters and solver options
model.setTimepoints(np.linspace(0, 60, 60))
rdata = amici.runAmiciSimulation(model, solver)
amici.plotting.plotObservableTrajectories(rdata)
```



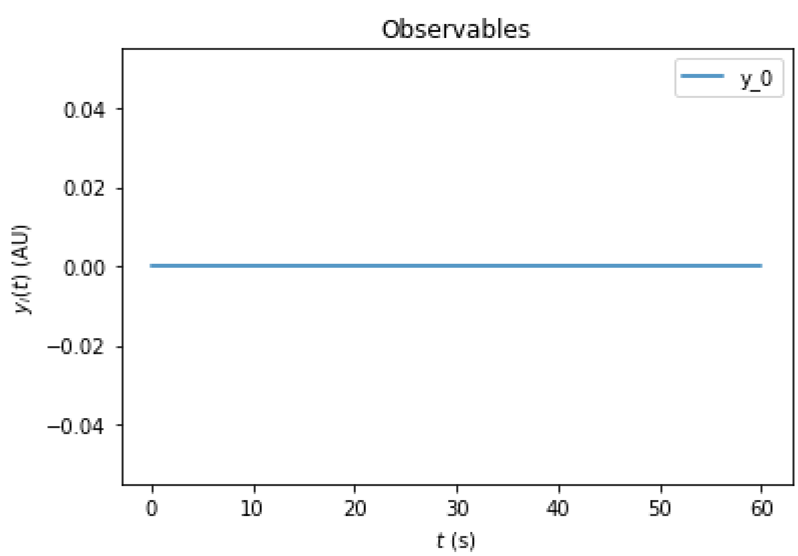
Simulation options can be specified either in the Model or in an ExpData instance. The `ExpData` instance can also carry experimental data. To initialize an `ExpData` instance from simulation results, amici offers some convenient constructors. In the following we will initialize an `ExpData` object from simulation results, but add noise with standard deviation `0.1` and specify the standard deviation accordingly. Moreover, we will specify custom values for `DRUG_0=0` and `KIN_0=2`. If `fixedParameter` is specified in an `ExpData` instance, runAmiciSimulation will use those parameters instead of the ones specified in the `Model` instance.

[10]:
```
edata = amici.ExpData(rdata, 0.1, 0.0)
edata.fixedParameters = [0,2]
rdata = amici.runAmiciSimulation(model, solver, edata)
amici.plotting.plotObservableTrajectories(rdata)
```



For many biological systems, it is reasonable to assume that they start in a steady state. In this example we want to specify an experiment where a pretreatment with a drug is performed *before* the kinase is added. We assume that the pretreatment is sufficiently long such that the system reaches steadystate before the kinase is added. To implement this in amici, we can specify `fixedParametersPreequilibration` in the `ExpData` object. This automatically adds a preequilibration phase where the model is run to steadystate, before regular simulation starts. Here we set `DRUG_0=3` and `KIN_0=0` for the preequilibration. This means that there is no kinase available in the preequilibration phase.

[11]:
```
edata.fixedParametersPreequilibration = [3,0]
rdata = amici.runAmiciSimulation(model, solver, edata)
amici.plotting.plotObservableTrajectories(rdata)
```

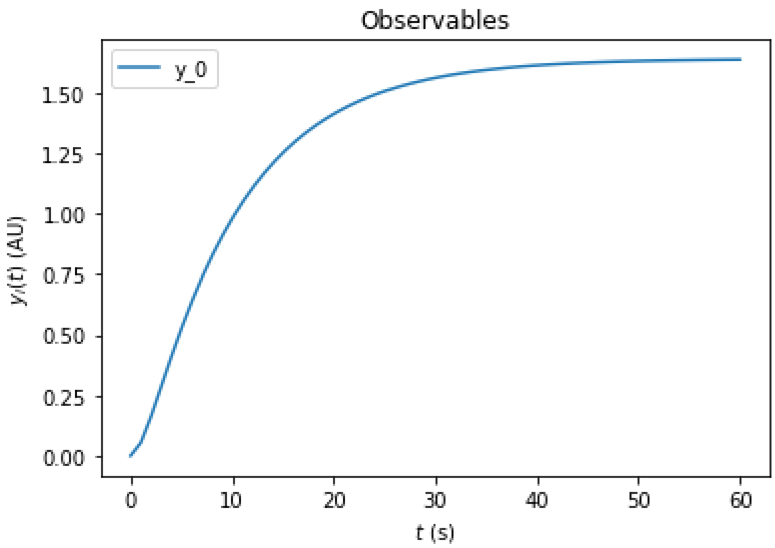


The resulting trajectory is definitely not what one may expect. The problem is that the `DRUG_0` and `KIN_0` set initial conditions for species in the model. By default these initial conditions are only applied at the very beginning of the simulation, i.e., before the preequilibration. Accordingly, the `fixedParameters` that we specified do not have any effect. To fix this, we need to set the `reinitializeFixedParameterInitialStates` attribue to `True`, to spefify that AMICI reinitializes all states that have `fixedParameter`-dependent initial states.

[12]:
```
edata.reinitializeFixedParameterInitialStates = True
```

With this option activated, the kinase concentration will be reinitialized after the preequilibration and we will see the expected change in fractional phosphorylation:

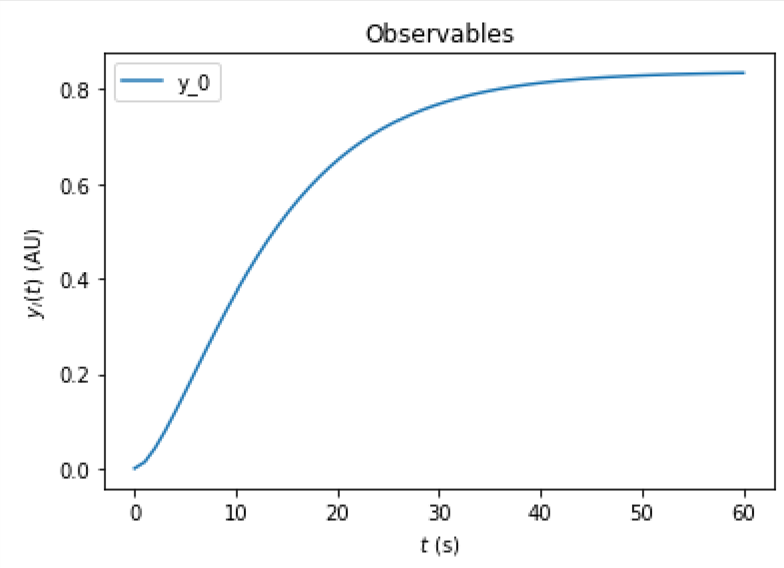
[13]:
```
rdata = amici.runAmiciSimulation(model, solver, edata)
amici.plotting.plotObservableTrajectories(rdata)
```



On top of preequilibration, we can also specify presimulation. This option can be used to specify pretreatments where the system is not assumed to reach steadystate. Presimulation can be activated by specifying `t_presim` and `edata.fixedParametersPresimulation`. If both `fixedParametersPresimulation` and `fixedParametersPreequilibration` are specified, preequilibration will be performed first, followed by presimulation, followed by regular simulation. For this example we specify `DRUG_0=10` and `KIN_0=0` for the presimulation and `DRUG_0=10` and `KIN_0=2` for the regular simulation. We do not overwrite the `DRUG_0=3` and `KIN_0=0` that was previously specified for preequilibration.

[14]:
```
edata.t_presim = 10
edata.fixedParametersPresimulation = [10.0, 0.0]
edata.fixedParameters = [10.0, 2.0]
print(edata.fixedParametersPreequilibration)
print(edata.fixedParametersPresimulation)
print(edata.fixedParameters)
rdata = amici.runAmiciSimulation(model, solver, edata)
amici.plotting.plotObservableTrajectories(rdata)
```

```
(3.0, 0.0)
(10.0, 0.0)
(10.0, 2.0)
```



AMICI documentation example of the steady state solver logic

This is an example to document the internal logic of the steady state solver, which is used in preequilibration and postequilibration.

Steady states of dynamical system

Not every dynamical system needs to run into a steady state. Instead, it may exhibit

- continuous growth, e.g.,

$$\dot{x} = x, \quad x_0 = 1$$

- a finite-time blow up, e.g.,

$$\dot{x} = x^2, \quad x_0 = 1$$

- oscillations, e.g.,

$$\ddot{x} = -x, \quad x_0 = 1$$

- chaotic behaviour, e.g., the Lorentz attractor

If the considered dynamical system has a steady state for positive times, then integrating the ODE long enough will equilibrate the system to this steady state. However, this may be computationally more demanding than other approaches and may fail, if the maximum number of integration steps is exceeded before reaching the steady state.

In general, Newton's method will find the steady state faster than forward simulation. However, it only converges if started close enough to the steady state. Moreover, it will not work, if the dynamical system has conserved quantities which were not removed prior to steady state computation: Conserved quantities will cause singularities in the Jacobian of the right hand side of the system, such that the linear problem within each step of Newton's method can not be solved.

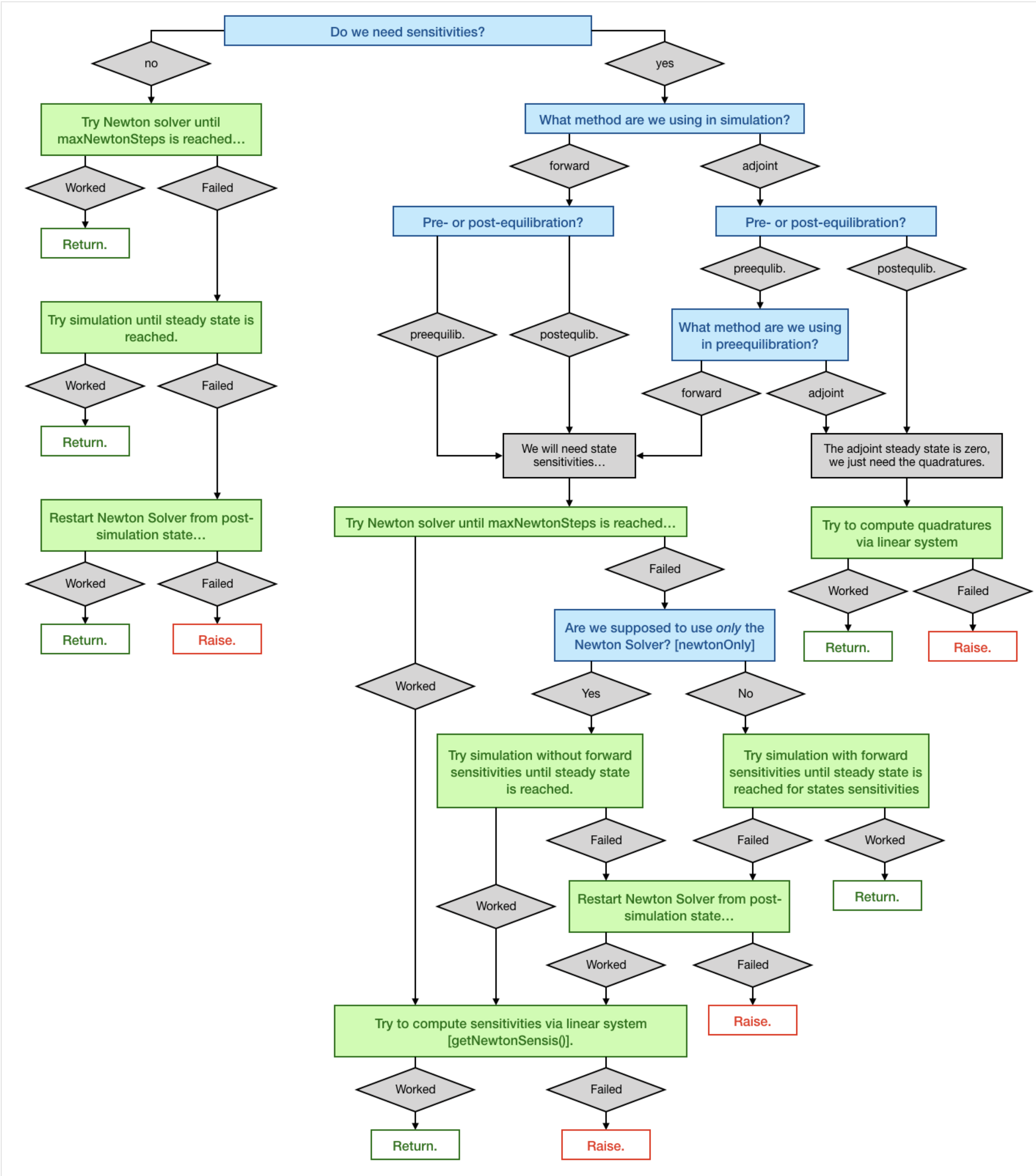
Logic of the steady state solver

If AMICI has to equilibrate a dynamical system, it can do this either via simulating until the right hand side of the system becomes small, or it can try to find the steady state directly by Newton's method. Amici decides automatically which approach is chosen and how forward or adjoint sensitivities are computed, if requested. However, the user can influence this behavior, if prior knowledge about the dynamical is available.

The logic which AMICI will follow to equilibrate the system works as follows:

[1]:
```
from IPython.display import Image
fig = Image(filename=('../../../documentation/gfx/steadystate_solver_workflow.png'))
fig
```

[1]:


Do we need sensitivities?
no
yes
Try Newton solver until maxNewtonSteps is reached…
What method are we using in simulation?
Worked
Failed
forward
adjoint
Return.
Pre- or post-equilibration?
Pre- or post-equilibration?
preequlib.
postequlib.
Try simulation until steady state is reached.
What method are we using in preequilibration?
preequilib.
postequlib.
Worked
Failed
forward
adjoint
Return.
We will need state sensitivities…
The adjoint steady state is zero, we just need the quadratures.
Restart Newton Solver from post-simulation state…
Try Newton solver until maxNewtonSteps is reached…
Try to compute quadratures via linear system
Worked
Failed
Failed
Worked
Failed
Return.
Raise.
Are we supposed to use only the Newton Solver? [newtonOnly]
Return.
Raise.
Worked
Yes
No
Try simulation without forward sensitivities until steady state is reached.
Try simulation with forward sensitivities until steady state is reached for states sensitivities
Failed
Failed
Worked
Worked
Restart Newton Solver from post-simulation state…
Return.
Worked
Failed
Try to compute sensitivities via linear system [getNewtonSensis()].
Raise.
Worked
Failed
Return.
Raise.

The example model

We will use the example model `model_constant_species.xml`, which has conserved species. Those are automatically removed in the SBML import of AMICI, but they can also be kept in the model to demonstrate the failure of Newton's method due to a singular right hand side Jacobian.

[2]:
```
import libsbml
import importlib
import amici
import os
import sys
import numpy as np
import matplotlib.pyplot as plt

# SBML model we want to import
sbml_file = 'model_constant_species.xml'

# Name of the models that will also be the name of the python module
model_name = 'model_constant_species'
model_reduced_name = model_name + '_reduced'

# Directories to which the generated model code is written
model_output_dir = model_name
model_reduced_output_dir = model_reduced_name

# Read the model and give some output
sbml_reader = libsbml.SBMLReader()
sbml_doc = sbml_reader.readSBML(sbml_file)
sbml_model = sbml_doc.getModel()
dir(sbml_doc)

print('Species: ', [s.getId() for s in sbml_model.getListOfSpecies()])

print('\nReactions:')
for reaction in sbml_model.getListOfReactions():
    reactants = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfReactants()])
    products  = ' + '.join(['%s %s'%(int(r.getStoichiometry()) if r.getStoichiometry() >␣
↪1 else '', r.getSpecies()) for r in reaction.getListOfProducts()])
    reversible = '<' if reaction.getReversible() else ''
    print('%3s: %10s %1s->%10s\t\t[%s]' % (reaction.getId(),
                        reactants,
                        reversible,
                        products,
                        libsbml.formulaToL3String(reaction.getKineticLaw().getMath())))
```

```
Species:  ['substrate', 'enzyme', 'complex', 'product']

Reactions:
creation:              -> substrate              [compartment * (synthesis_substrate + k_
↪create)]
binding:  substrate +  enzyme <->    complex             [compartment * (k_bind *␣
↪substrate * enzyme - k_unbind * complex)]
conversion:    complex  -> enzyme +  product             [compartment * k_convert *␣
↪complex]
```

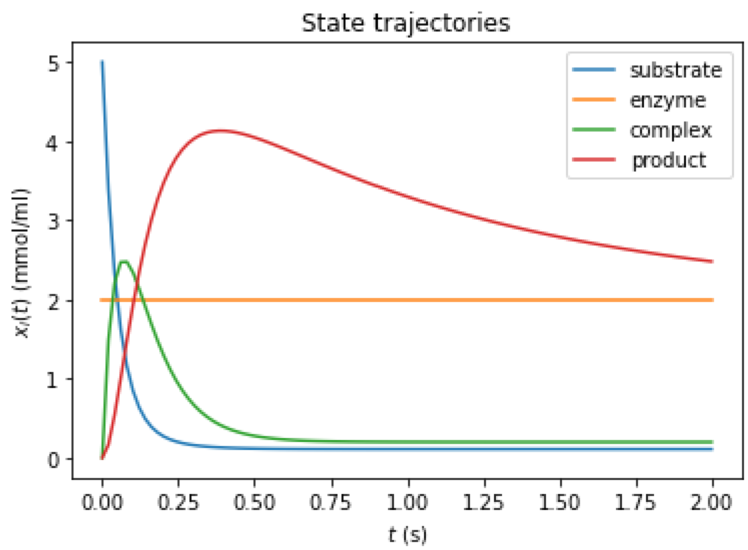
(continues on next page)

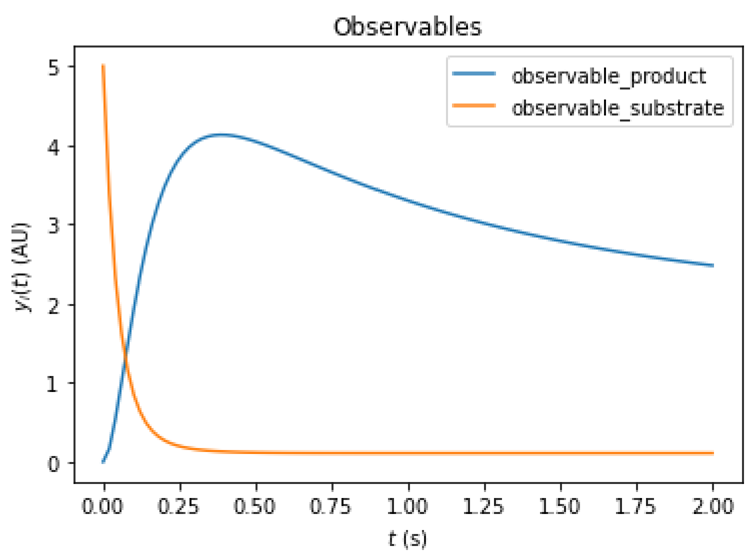
(continued from previous page)

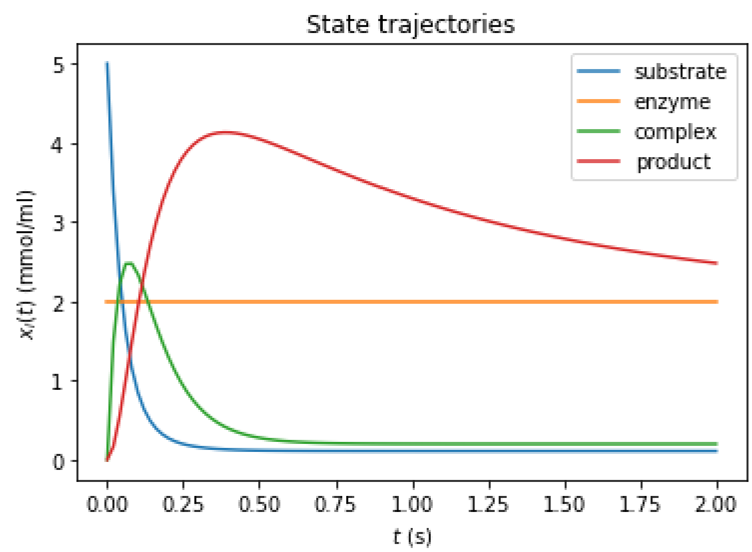
```
decay:    product  ->                   [compartment * k_decay * product]
```

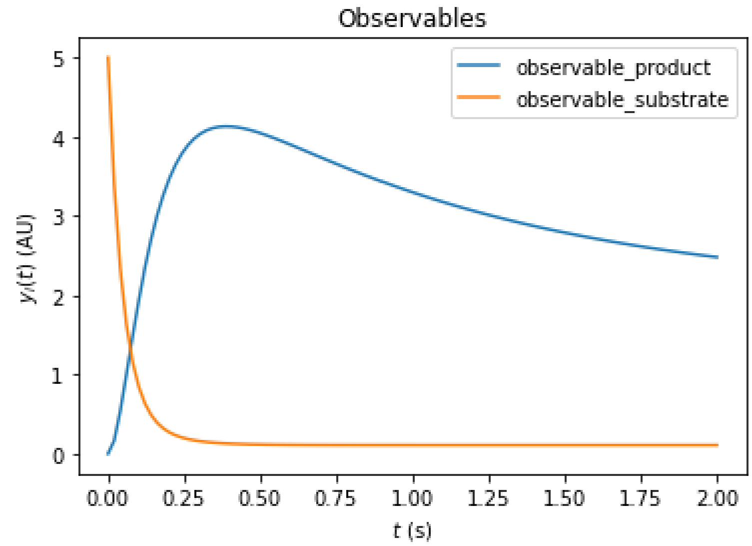
[3]:
```
# Create an SbmlImporter instance for our SBML model
sbml_importer = amici.SbmlImporter(sbml_file)

# specify observables and constant parameters
constantParameters = ['synthesis_substrate', 'init_enzyme']
observables = {
    'observable_product': {'name': '', 'formula': 'product'},
    'observable_substrate': {'name': '', 'formula': 'substrate'},
}
sigmas = {'observable_product': 1.0, 'observable_substrate': 1.0}

# import the model
sbml_importer.sbml2amici(model_reduced_name,
                         model_reduced_output_dir,
                         observables=observables,
                         constant_parameters=constantParameters,
                         sigmas=sigmas)
sbml_importer.sbml2amici(model_name,
                         model_output_dir,
                         observables=observables,
                         constant_parameters=constantParameters,
                         sigmas=sigmas,
                         compute_conservation_laws=False)
```

[4]:
```
# import the models and run some test simulations
model_reduced_module = amici.import_model_module(model_reduced_name, os.path.
→abspath(model_reduced_output_dir))
model_reduced = model_reduced_module.getModel()

model_module = amici.import_model_module(model_name, os.path.abspath(model_output_dir))
model = model_module.getModel()


# simulate model with conservation laws
model_reduced.setTimepoints(np.linspace(0, 2, 100))
solver_reduced = model_reduced.getSolver()
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced)

# simulate model without conservation laws
model.setTimepoints(np.linspace(0, 2, 100))
solver = model.getSolver()
rdata = amici.runAmiciSimulation(model, solver)

# plot trajectories
import amici.plotting
amici.plotting.plotStateTrajectories(rdata_reduced, model=model_reduced)
amici.plotting.plotObservableTrajectories(rdata_reduced, model=model_reduced)

amici.plotting.plotStateTrajectories(rdata, model=model)
amici.plotting.plotObservableTrajectories(rdata, model=model)
```


State trajectories
substrate
enzyme
complex
product
$x_i(t)$ (mmol/ml)
t (s)
0
1
2
3
4
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Observables
observable_product
observable_substrate
$y_i(t)$ (AU)
t (s)
0
1
2
3
4
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00





Inferring the steady state of the system (postequilibration)

First, we want to demonstrate that Newton's method will fail with the unreduced model due to a singular right hand side Jacobian.

[5]:
```
# Call postequilibration by setting an infinity timepoint
model.setTimepoints(np.full(1, np.inf))

# set the solver
solver = model.getSolver()
solver.setNewtonMaxSteps(10)
solver.setMaxSteps(1000)
rdata = amici.runAmiciSimulation(model, solver)

#np.set_printoptions(threshold=8, edgeitems=2)
```

(continues on next page)

(continued from previous page)

```
for key, value in rdata.items():
    print('%12s: ' % key, value)
```

```
          ts:  [inf]
           x:  [[0.11       2.         0.2        2.00000002]]
          x0:  [5. 2. 0. 0.]
        x_ss:  [nan nan nan nan]
          sx:  None
         sx0:  None
       sx_ss:  None
           y:  [[2.00000002 0.11      ]]
      sigmay:  [[1. 1.]]
          sy:  None
     ssigmay:  None
           z:  None
          rz:  None
      sigmaz:  None
          sz:  None
         srz:  None
     ssigmaz:  None
        sllh:  None
       s2llh:  None
           J:  [[-20.    0.   20.    0. ]
 [ -1.1   0.    1.1   0. ]
 [  1.    0.  -11.   10. ]
 [  0.    0.    0.   -1. ]]
        xdot:  [ 0.00000000e+00  0.00000000e+00  2.22044605e-16 -2.24170307e-08]
      status:  0.0
         llh:  nan
        chi2:  nan
         res:  [0. 0.]
        sres:  None
         FIM:  None
           w:  [[2.         2.         2.         2.00000002]]
  preeq_wrms:  nan
     preeq_t:  nan
preeq_numlinsteps:  None
preeq_numsteps:  [[0 0 0]]
preeq_numstepsB:  12.0
preeq_status:  [[0 0 0]]
preeq_cpu_time:  0.0
preeq_cpu_timeB:  0.0
 posteq_wrms:  0.5604257578208488
    posteq_t:  19.2252094591474
posteq_numlinsteps:  None
posteq_numsteps:  [[  0 417   0]]
posteq_numstepsB:  0.0
posteq_status:  [[-3  1  0]]
posteq_cpu_time:  2.315
posteq_cpu_timeB:  0.0
    numsteps:  None
 numrhsevals:  None
```

(continues on next page)

(continued from previous page)

```
numerrtestfails:  None
numnonlinsolvconvfails:  None
       order:  None
    cpu_time:  0.0
   numstepsB:  None
numrhsevalsB:  None
numerrtestfailsB:  None
numnonlinsolvconvfailsB:  None
   cpu_timeB:  0.0
```

The fields `posteq_status` and `posteq_numsteps` in rdata tells us how postequilibration worked:

- the first entry informs us about the status/number of steps in Newton's method (here 0, as Newton's method did not work)
- the second entry tells us, the status/how many integration steps were taken until steady state was reached
- the third entry informs us about the status/number of Newton steps in the second launch, after simulation

The status is encoded as an Integer flag with the following meanings:

- `1`: Successful run
- `0`: Did not run
- `-1`: Error: No further specification is given, the error message should give more information.
- `-2`: Error: The method did not converge to a steady state within the maximum number of steps (Newton's method or simulation).
- `-3`: Error: The Jacobian of the right hand side is singular (only Newton's method)
- `-4`: Error: The damping factor in Newton's method was reduced until it met the lower bound without success (Newton's method only)
- `-5`: Error: The model was simulated past the timepoint `t=1e100` without finding a steady state. Therefore, it is likely that the model has not steady state for the given parameter vector.

Here, only the second entry of `posteq_status` contains a positive integer: The first run of Newton's method failed due to a Jacobian, which oculd not be factorized, but the second run (simulation) contains the entry 1 (success). The third entry is 0, thus Newton's method was not launched for a second time. More information can be found in`posteq_numsteps`: Also here, only the second entry contains a positive integer, which is smaller than the maximum number of steps taken (<1000). Hence steady state was reached via simulation, which corresponds to the simulated time written to `posteq_time`.

We want to demonstrate a complete failure if inferring the steady state by reducing the number of integration steps to a lower value:

[6]:
```
# reduce maxsteps for integration
solver.setMaxSteps(100)
rdata = amici.runAmiciSimulation(model, solver)
print('Status of postequilibration:', rdata['posteq_status'])
print('Number of steps employed in postequilibration:', rdata['posteq_numsteps'])
```

```
Status of postequilibration: [[-3 -2 -3]]
Number of steps employed in postequilibration: [[  0 100   0]]
```

```
[Warning] AMICI:simulation: AMICI simulation failed:
Steady state computation failed. First run of Newton solver failed: RHS could not be␣
↪factorized. Simulation to steady state failed: No convergence was achieved. Second run␣
↪of Newton solver failed: RHS could not be factorized.
```

(continues on next page)

(continued from previous page)

```
Error occured in:
0          0x1060f7913 amici::SteadystateProblem::handleSteadyStateFailure(amici::Solver␣
↪const*, amici::Model*) + 531
1          0x1060f6b3c amici::SteadystateProblem::findSteadyState(amici::Solver*, amici::
↪NewtonSolver*, amici::Model*, int) + 332
2          0x1060f6882 amici::SteadystateProblem::workSteadyStateProblem(amici::Solver*,␣
↪amici::Model*, int) + 322
3          0x1060a4615 amici::AmiciApplication::runAmiciSimulation(amici::Solver&, amici:
↪:ExpData const*, amici::Model&, bool) + 405
4
```

However, the same logic works, if we use the reduced model. For sufficiently many Newton steps, postequilibration is achieved by Newton's method in the first run. In this specific example, the steady state is found within one step.

[7]:
```
# Call postequilibration by setting an infinity timepoint
model_reduced.setTimepoints(np.full(1, np.inf))

# set the solver
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setMaxSteps(100)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced)

print('Status of postequilibration:', rdata_reduced['posteq_status'])
print('Number of steps employed in postequilibration:', rdata_reduced['posteq_numsteps'])
```

```
Status of postequilibration: [[1 0 0]]
Number of steps employed in postequilibration: [[2 0 0]]
```

Postequilibration with sensitivities

Equilibration is possible with forward and adjoint sensitivity analysis. As for the main simulation part, adjoint sensitivity analysis yields less information than forward sensitivity analysis, since no state sensitivities are computed. However, it has a better scaling behavior towards large model sizes.

Postequilibration with forward sensitivities

If forward sensitivity analysis is used, then state sensitivities at the timepoint `np.inf` will be computed. This can be done in (currently) two different ways:

1. If the Jacobian $\nabla_x f$ of the right hand side f is not (close to) singular, the most efficient approach will be solving the linear system of equations, which defines the steady state sensitivities:

$$0 = \dot{s}^x = (\nabla_x f) s^x + \frac{\partial f}{\partial \theta} \qquad \Rightarrow \qquad (\nabla_x f) s^x = - \frac{\partial f}{\partial \theta}$$

   This approach will always be chosen by AMICI, if the option `model.SteadyStateSensitivityMode` is set to `SteadyStateSensitivityMode.newtonOnly`. Furthermore, it will also be chosen, if the steady state was found by Newton's method, as in this case, the Jacobian is at least not singular (but may still be poorly conditioned). A check for the condition number of the Jacobian is currently missing, but will soon be implemented.

2. If the Jacobian is poorly conditioned or singular, then the only way to obtain a reliable result will be integrating the state variables with state sensitivities until the norm of the right hand side becomes small. This

approach will be chosen by AMICI, if the steady state was found by simulation and the option `model.SteadyStateSensitivityMode` is set to `SteadyStateSensitivityMode.simulationFSA`. This approach is numerically more stable, but the computation time for large models may be substantial.

Side remark:

A possible third way may consist in a (relaxed) Richardson iteration type approach, which interprets the entries of the right hand side f as residuals and minimizes the squared residuals $\|f\|^2$ by a Levenberg-Marquart-type algorithm. This approach would also work for poorly conditioned (and even for singular Jacobians if additional constraints are implemented as Lagrange multipliers) while being faster than a long forward simulation.

We want to demonstrate both possibilities to find the steady state sensitivities, as well as the failure of their computation if the Jacobian is singular and the `newtonOnly` setting was used.

[8]:
```
# Call simulation with singular Jacobian and integrateIfNewtonFails mode
model.setTimepoints(np.full(1, np.inf))
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.
↪integrateIfNewtonFails)
solver = model.getSolver()
solver.setNewtonMaxSteps(10)
solver.setSensitivityMethod(amici.SensitivityMethod.forward)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
solver.setMaxSteps(10000)
rdata = amici.runAmiciSimulation(model, solver)

print('Status of postequilibration:', rdata['posteq_status'])
print('Number of steps employed in postequilibration:', rdata['posteq_numsteps'])
print('Computed state sensitivities:')
print(rdata['sx'][0,:,:])
```

```
Status of postequilibration: [[-3  1  0]]
Number of steps employed in postequilibration: [[   0 1026    0]]
Computed state sensitivities:
[[-1.10000000e-02  0.00000000e+00 -6.70507402e-18 -1.20114408e-11]
 [ 1.00000000e-02  0.00000000e+00 -8.22965063e-19  1.20114329e-11]
 [-1.00000000e-03  0.00000000e+00 -2.00000000e-02 -2.40228711e-11]
 [ 5.50000000e-02  0.00000000e+00  1.00000000e-01  9.99999999e-01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -2.00000004e+00]]
```

[9]:
```
# Call simulation with singular Jacobian and newtonOnly mode (will fail)
model.setTimepoints(np.full(1, np.inf))
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.newtonOnly)
solver = model.getSolver()
solver.setSensitivityMethod(amici.SensitivityMethod.forward)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
solver.setMaxSteps(10000)
rdata = amici.runAmiciSimulation(model, solver)

print('Status of postequilibration:', rdata['posteq_status'])
print('Number of steps employed in postequilibration:', rdata['posteq_numsteps'])
print('Computed state sensitivities:')
print(rdata['sx'][0,:,:])
```

```
Status of postequilibration: [[-2 -1  1]]
Number of steps employed in postequilibration: [[  0 543   0]]
Computed state sensitivities:
```

(continues on next page)

(continued from previous page)

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
[Warning] AMICI:simulation: AMICI simulation failed:
Steady state sensitvitiy computation failed due to unsuccessful factorization of RHS␣
↪Jacobian
Error occured in:
0          0x1060f698b amici::SteadystateProblem::workSteadyStateProblem(amici::Solver*,␣
↪amici::Model*, int) + 587
1          0x1060a4615 amici::AmiciApplication::runAmiciSimulation(amici::Solver&, amici:
↪:ExpData const*, amici::Model&, bool) + 405
2          0x1060a4474 amici::runAmiciSimulation(amici::Solver&, amici::ExpData const*,␣
↪amici::Model&, bool) + 36
3          0x106061005 _wrap_runAmiciSimulation(_object*, _object*) + 549
4          0x1021b2309 cfunction_call_varargs + 320
5
```

[10]:
```
# Call postequilibration by setting an infinity timepoint
model_reduced.setTimepoints(np.full(1, np.inf))
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.newtonOnly)
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.forward)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
solver_reduced.setMaxSteps(1000)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced)

print('Status of postequilibration:', rdata_reduced['posteq_status'])
print('Number of steps employed in postequilibration:', rdata_reduced['posteq_numsteps'])
print('Computed state sensitivities:')
print(rdata_reduced['sx'][0,:,:])
```

```
Status of postequilibration: [[1 0 0]]
Number of steps employed in postequilibration: [[2 0 0]]
Computed state sensitivities:
[[-1.1e-02  0.0e+00 -0.0e+00 -0.0e+00]
 [ 1.0e-02  0.0e+00 -0.0e+00 -0.0e+00]
 [-1.0e-03  0.0e+00 -2.0e-02 -0.0e+00]
 [ 5.5e-02  0.0e+00  1.0e-01  1.0e+00]
 [-0.0e+00  0.0e+00 -0.0e+00 -2.0e+00]]
```

Postequilibration with adjoint sensitivities

Postequilibration also works with adjoint sensitivities. In this case, it is exploited that the ODE of the adjoint state p will always have the steady state 0, since it's a linear ODE:

$$\frac{d}{dt}p(t) = J(x^*, \theta)^T p(t),$$

where x^* denotes the steady state of the system state. Since the Eigenvalues of the Jacobian are negative and since the Jacobian at steady state is a fixed matrix, this system has a simple algebraic solution:

$$p(t) = e^{tJ(x^*,\theta)^T} p_{\text{end}}.$$

As a consequence, the quadratures in adjoint computation also reduce to a matrix-vector product:

$$Q(x, \theta) = Q(x^*, \theta) = p_{\text{integral}} * \frac{\partial f}{\partial \theta}$$

with

$$p_{\text{integral}} = \int_0^\infty p(s)ds = (J(x^*, \theta)^T)^{-1} p_{\text{end}}.$$

However, this solution is given in terms of a linear system of equations defined by the transposed Jacobian of the right hand side. Hence, if the (transposed) Jacobian is singular, it is not applicable. In this case, standard integration must be carried out.

[11]:
```
# Call adjoint postequilibration by setting an infinity timepoint
# and create an edata object, which is needed for adjoint computation
edata = amici.ExpData(2, 0, 0, np.array([float('inf')]))
edata.setObservedData([1.8] * 2)
edata.fixedParameters = np.array([3., 5.])

model_reduced.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.newtonOnly)
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
solver_reduced.setMaxSteps(1000)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

print('Status of postequilibration:', rdata_reduced['posteq_status'])
print('Number of steps employed in postequilibration:', rdata_reduced['posteq_numsteps'])
print('Number of backward steps employed in postequilibration:', rdata_reduced['posteq_
↪numstepsB'])
print('Computed gradient:', rdata_reduced['sllh'])
```

```
Status of postequilibration: [[1 0 0]]
Number of steps employed in postequilibration: [[2 0 0]]
Number of backward steps employed in postequilibration: 0.0
Computed gradient: [-1.85900e-02  1.69000e-02 -1.69000e-03 -3.16282e+00  1.60000e+01]
```

If we carry out the same computation with a system that has a singular Jacobian, then `posteq_numstepsB` will not be `0` any more (which indicates that the linear system solve was used to compute backward postequilibration). Now, integration is carried out and hence `posteq_numstepsB > 0`

```
[12]: # Call adjoint postequilibration with model with singular Jacobian
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.newtonOnly)
solver = model.getSolver()
solver.setNewtonMaxSteps(10)
solver.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
rdata = amici.runAmiciSimulation(model, solver, edata)

print('Status of postequilibration:', rdata['posteq_status'])
print('Number of steps employed in postequilibration:', rdata['posteq_numsteps'])
print('Number of backward steps employed in postequilibration:', rdata['posteq_numstepsB
↪'])
print('Computed gradient:', rdata['sllh'])
```

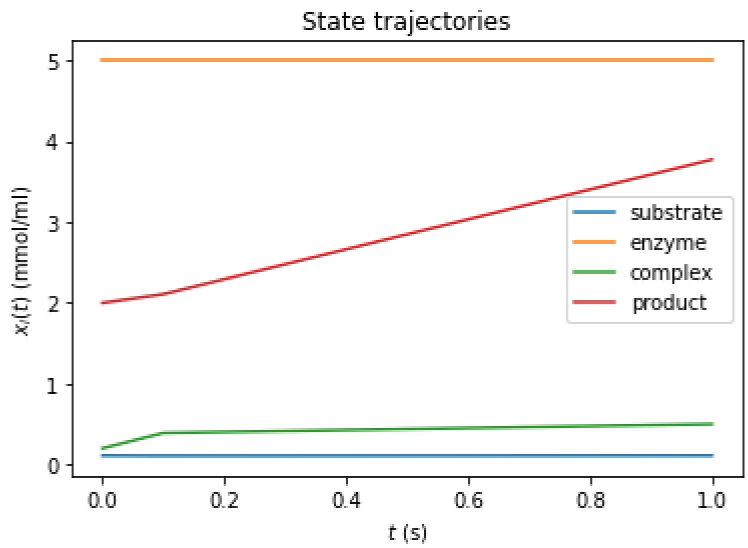
```
Status of postequilibration: [[-3 -1  1]]
Number of steps employed in postequilibration: [[  0 479   0]]
Number of backward steps employed in postequilibration: 3076.0
Computed gradient: [-1.85899987e-02  1.68999988e-02 -1.69000055e-03 -3.16282001e+00
  1.60000000e+01]
```

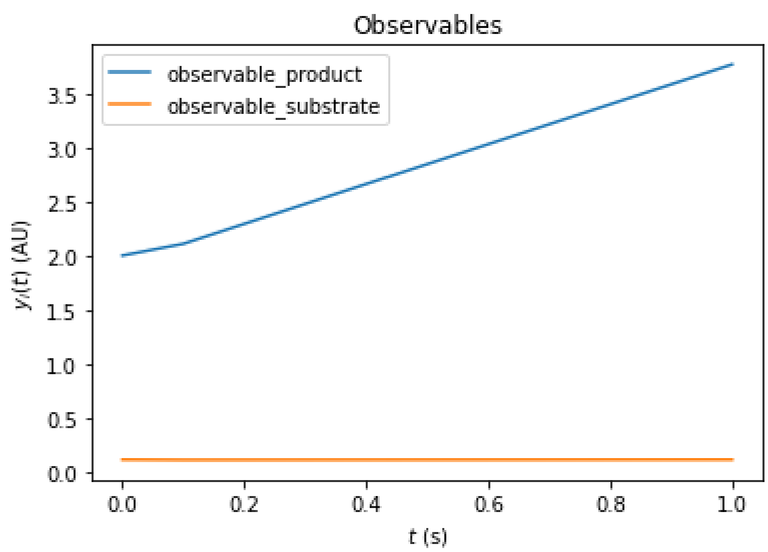
Preequilibrating the model

Sometimes, we want to launch a solver run from a steady state which was inferred numerically, i.e., the system was preequilibrated. In order to do this with AMICI, we need to pass an ExpData object, which contains fixed parameter for the actual simulation and for preequilibration of the model.

```
[13]: # create edata, with 3 timepoints and 2 observables:
edata = amici.ExpData(2, 0, 0,
                      np.array([0., 0.1, 1.]))
edata.setObservedData([1.8] * 6)
edata.fixedParameters = np.array([3., 5.])
edata.fixedParametersPreequilibration = np.array([0., 2.])
edata.reinitializeFixedParameterInitialStates = True
```

```
[14]: # create the solver object and run the simulation
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

amici.plotting.plotStateTrajectories(rdata_reduced, model = model_reduced)
amici.plotting.plotObservableTrajectories(rdata_reduced, model = model_reduced)
```





We can also combine pre- and postequilibration.

[15]:
```
# Change the last timepoint to an infinity timepoint.
edata.setTimepoints(np.array([0., 0.1, float('inf')]))

# run the simulation
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)
```

Preequilibration with sensitivities

Beyond the need for an ExpData object, the steady state solver logic in preequilibration is the same as in postequilibration, also if sensitivities are requested. The computation will fail for singular Jacobians, if SteadyStateSensitivityMode is set to newtonOnly, or if not enough steps can be taken. However, if forward simulation with steady state sensitivities is allowed, or if the Jacobian is not singular, it will work.

Prequilibration with forward sensitivities

[16]:
```
# No postquilibration this time.
edata.setTimepoints(np.array([0., 0.1, 1.]))

# create the solver object and run the simulation, singular Jacobian, enforce Newton␣
↪solver for sensitivities
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.newtonOnly)
solver = model.getSolver()
solver.setNewtonMaxSteps(10)
solver.setSensitivityMethod(amici.SensitivityMethod.forward)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
rdata = amici.runAmiciSimulation(model, solver, edata)

for key, value in rdata.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
```

```
          preeq_wrms:  0.5604257578208488
             preeq_t:  19.2252094591474
   preeq_numlinsteps:  None
      preeq_numsteps:  [[  0 417   0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[-3  1  0]]
      preeq_cpu_time:  1.723
     preeq_cpu_timeB:  0.0
```

```
[Warning] AMICI:simulation: AMICI simulation failed:
Steady state sensitivitiy computation failed due to unsuccessful factorization of RHS␣
↪Jacobian
Error occured in:
0          0x1060f698b amici::SteadystateProblem::workSteadyStateProblem(amici::Solver*,␣
↪amici::Model*, int) + 587
1          0x1060a456f amici::AmiciApplication::runAmiciSimulation(amici::Solver&, amici:
↪:ExpData const*, amici::Model&, bool) + 239
2          0x1060a4474 amici::runAmiciSimulation(amici::Solver&, amici::ExpData const*,␣
↪amici::Model&, bool) + 36
3          0x106061005 _wrap_runAmiciSimulation(_object*, _object*) + 549
4          0x1021b2309 cfunction_call_varargs + 320
5
```

[17]:
```
# Singluar Jacobian, use simulation
model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.
↪integrateIfNewtonFails)
solver = model.getSolver()
```

(continues on next page)

(continued from previous page)

```
solver.setNewtonMaxSteps(10)
solver.setSensitivityMethod(amici.SensitivityMethod.forward)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
rdata = amici.runAmiciSimulation(model, solver, edata)

for key, value in rdata.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
```

```
          preeq_wrms:  0.9920376238481097
             preeq_t:  21.270502326483026
   preeq_numlinsteps:  None
      preeq_numsteps:  [[   0 1026    0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[-3  1  0]]
      preeq_cpu_time:  12.439
     preeq_cpu_timeB:  0.0
```

[18]:
```
# Non-singular Jacobian, use Newton solver
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.forward)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

for key, value in rdata_reduced.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
```

```
          preeq_wrms:  0.0
             preeq_t:  nan
   preeq_numlinsteps:  None
      preeq_numsteps:  [[2 0 0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[1 0 0]]
      preeq_cpu_time:  0.036
     preeq_cpu_timeB:  0.0
```

Prequilibration with adjoint sensitivities

When using preequilibration, adjoint sensitivity analysis can be used for simulation. This is a particularly interesting case: Standard adjoint sensitivity analysis requires the initial state sensitivities `sx0` to work, at least if data is given for finite (i.e., not exclusively postequilibration) timepoints: For each parameter, a contribution to the gradient is given by the scalar product of the corresponding state sensitivity vector at timepoint $t = 0$, (column in `sx0`), with the adjoint state ($p(t = 0)$). Hence, the matrix `sx0` is needed. This scalar product "closes the loop" from forward to adjoint simulation.

By default, if adjoint sensitivity analysis is called with preequilibration, the initial state sensitivities are computed in just the same way as if this way done for forward sensitivity analysis. The only difference in the internal logic is that, if the steady state gets inferred via simulation, a separate solver object is used in order to ensure that the steady state simulation does not interfere with the snapshotting of the forward trajectory from the actual time course.

However, also an adjoint version of preequilibration is possible: In this case, the "loop" from forward to adjoint simulation needs no closure: The simulation time is extended by preequilibration: forward from $t = -\infty$ to $t = 0$, and after adjoint simulation also backward from $t = 0$ to $t = -\infty$. Similar to adjoint postequilibration, the steady state of the adjoint state (at $t = -\infty$) is $p = 0$, hence the scalar product (at $t = -\infty$) for the initial state sensitivities of preequilibration with the adjoint state vanishes. Instead, this gradient contribution is covered by additional quadratures $\int_{-\infty}^{0} p(s)ds \cdot \frac{\partial f}{\partial \theta}$. In order to compute these quadratures correctly, the adjoint state from the main adjoint simulation must be passed on to the initial adjoint state of backward preequilibration.

However, as the adjoint state must be passed on from backward computation to preequilibration, it is currently not allowed to alter (reinitialize) states of the model at $t = 0$, unless these states are constant, as otherwise this alteration would lead to a discontinuity in the adjoints state as well and hence to an incorrect gradient.

[19]:
```
# Non-singular Jacobian, use Newton solver and adjoints with initial state sensitivities
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

for key, value in rdata_reduced.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
print('Gradient:', rdata_reduced['sllh'])
```

```
          preeq_wrms:  0.0
             preeq_t:  nan
   preeq_numlinsteps:  None
      preeq_numsteps:  [[2 0 0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[1 0 0]]
      preeq_cpu_time:  0.039
     preeq_cpu_timeB:  0.0
Gradient: [-0.05528395  0.0461776  -0.03354519 -2.34602219  6.314481  ]
```

[20]:
```
# Non-singular Jacobian, use simulation solver and adjoints with initial state␣
↪sensitivities
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(0)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

for key, value in rdata_reduced.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
print('Gradient:', rdata_reduced['sllh'])
```

```
          preeq_wrms:  0.8470065245264354
             preeq_t:  19.213162474372176
   preeq_numlinsteps:  None
      preeq_numsteps:  [[  0 426   0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[-2  1  0]]
      preeq_cpu_time:  1.753
     preeq_cpu_timeB:  0.0
Gradient: [-0.05528395  0.0461776  -0.03354519 -2.34602226  6.3144812 ]
```

[21]:
```
# Non-singular Jacobian, use Newton solver and adjoints with fully adjoint␣
↪preequilibration
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(10)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver_reduced.setSensitivityMethodPreequilibration(amici.SensitivityMethod.adjoint)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

for key, value in rdata_reduced.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
print('Gradient:', rdata_reduced['sllh'])
```

```
          preeq_wrms:  0.0
             preeq_t:  nan
   preeq_numlinsteps:  None
      preeq_numsteps:  [[2 0 0]]
     preeq_numstepsB:  0.0
        preeq_status:  [[1 0 0]]
      preeq_cpu_time:  0.042
     preeq_cpu_timeB:  0.009
Gradient: [-0.05528395  0.0461776  -0.03354519 -2.34602219  6.314481  ]
```

As for postquilibration, adjoint preequilibration has an analytic solution (via the linear system), which will be preferred. If used for models with singular Jacobian, numerical integration will be carried out, which is indicated by preeq_numstepsB.

[22]:
```
# Non-singular Jacobian, use Newton solver and adjoints with fully adjoint␣
↪preequilibration
solver = model.getSolver()
solver.setNewtonMaxSteps(10)
solver.setSensitivityMethod(amici.SensitivityMethod.adjoint)
solver.setSensitivityMethodPreequilibration(amici.SensitivityMethod.adjoint)
solver.setSensitivityOrder(amici.SensitivityOrder.first)
rdata = amici.runAmiciSimulation(model, solver, edata)

for key, value in rdata.items():
    if key[0:6] == 'preeq_':
        print('%20s: ' % key, value)
print('Gradient:', rdata['sllh'])
```

```
          preeq_wrms:  0.9986067660342685
             preeq_t:  36.94272314329062
   preeq_numlinsteps:  None
      preeq_numsteps:  [[  0 417   0]]
     preeq_numstepsB:  1371.0
        preeq_status:  [[-3  1  0]]
      preeq_cpu_time:  2.488
     preeq_cpu_timeB:  5.016
Gradient: [-0.05528395  0.04617759 -0.03354518 -2.34602224  6.3144811 ]
```

Controlling the error tolerances in pre- and postequilibration

When solving ODEs or DAEs, AMICI uses the default logic of CVODES and IDAS to control error tolerances. This means that error weights are computed based on the absolute error tolerances and the product of current state variables of the system and their respective relative error tolerances. If this error combination is then controlled.

The respective tolerances for equilibrating a system with AMICI can be controlled by the user via the getter/setter functions `[get|set][Absolute|Relative]ToleranceSteadyState[Sensi]`:

[23]:
```
# Non-singular Jacobian, use simulaiton
model_reduced.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.
↪integrateIfNewtonFails)
solver_reduced = model_reduced.getSolver()
solver_reduced.setNewtonMaxSteps(0)
solver_reduced.setSensitivityMethod(amici.SensitivityMethod.forward)
solver_reduced.setSensitivityOrder(amici.SensitivityOrder.first)

# run with lax tolerances
solver_reduced.setRelativeToleranceSteadyState(1e-2)
solver_reduced.setAbsoluteToleranceSteadyState(1e-3)
solver_reduced.setRelativeToleranceSteadyStateSensi(1e-2)
solver_reduced.setAbsoluteToleranceSteadyStateSensi(1e-3)
rdata_reduced_lax = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

# run with strict tolerances
solver_reduced.setRelativeToleranceSteadyState(1e-12)
solver_reduced.setAbsoluteToleranceSteadyState(1e-16)
solver_reduced.setRelativeToleranceSteadyStateSensi(1e-12)
solver_reduced.setAbsoluteToleranceSteadyStateSensi(1e-16)
rdata_reduced_strict = amici.runAmiciSimulation(model_reduced, solver_reduced, edata)

# compare ODE outputs
print('\nODE solver steps, which were necessary to reach steady state:')
print('lax tolerances: ', rdata_reduced_lax['preeq_numsteps'])
print('strict tolerances: ', rdata_reduced_strict['preeq_numsteps'])

print('\nsimulation time corresponding to steady state:')
print(rdata_reduced_lax['preeq_t'])
print(rdata_reduced_strict['preeq_t'])

print('\ncomputation time to reach steady state:')
print(rdata_reduced_lax['preeq_cpu_time'])
print(rdata_reduced_strict['preeq_cpu_time'])
```

```

ODE solver steps, which were necessary to reach steady state:
lax tolerances:  [[   0 733    0]]
strict tolerances:  [[    0 1031    0]]

simulation time correpsonding to steady state:
6.002011407974004
31.0689293433781

computation time to reach steady state:
```

(continues on next page)

(continued from previous page)

```
7.646
7.837
```

Debugging simulation failures

Objective: Demonstrate common simulation failures and give some hints for interpreting, debugging, and fixing them.

[1]:
```
%matplotlib inline
import os
import amici
from amici.petab_import import import_petab_problem
from amici.petab_objective import simulate_petab, RDATAS, EDATAS
from amici.plotting import plot_state_trajectories, plot_jacobian
import petab
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from contextlib import suppress

try:
    import benchmark_models_petab
except ModuleNotFoundError:
    # install `benchmark_models_petab` if necessary
    %pip install -q -e "git+https://github.com/Benchmarking-Initiative/Benchmark-Models-
↪PEtab.git@master#subdirectory=src/python&egg=benchmark_models_petab"
    try:
        import benchmark_models_petab
    except ModuleNotFoundError:
        print("** Please restart the kernel. **")
```

Overview

In the following, we will simulate models contained in the PEtab Benchmark Collection to demonstrate a number of simulation failures to analyze and fix them. We use the PEtab format, as it makes model import and simulation much easier, but everything illustrated here, also applies to plain SBML or PySB import.

Note that, due to numerical issues, the examples below may not be fully reproducible on every system.

If any simulation failures occur, they will be printed via Python logging.

Programmatically, simulation success can be checked via `ReturnDataView.status`. In case of a successful simulation, and only then, this value corresponds to `amici.AMICI_SUCCESS`. In case of a simulation error, all quantities in `ReturnData`/`ReturnDataView` will be reported up to the time of failure, the rest will be `NaN`. The likelihood and it's gradient will always be `NaN` in case of failure.

AMICI_TOO_MUCH_WORK - mxstep steps taken before reaching tout

Let's run a simulation:

[2]:
```
petab_problem = benchmark_models_petab.get_problem("Fujita_SciSignal2010")
amici_model = import_petab_problem(petab_problem, verbose=False, force_compile=False)

np.random.seed(2991)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_TOO_MUCH_WORK', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS']
```

```
2023-01-25 16:03:24.023 - amici.swig_wrappers - DEBUG - [condition_step_03_0][CVODES:
↪CVode:TOO_MUCH_WORK] AMICI ERROR: in module CVODES in function CVode : At t = 3031.8,␣
↪mxstep steps taken before reaching tout.
2023-01-25 16:03:24.024 - amici.swig_wrappers - ERROR - [condition_step_03_0][FORWARD_
↪FAILURE] AMICI forward simulation failed at t = 3031.8: AMICI failed to integrate the␣
↪forward problem
```

```
Status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_TOO_MUCH_WORK',
↪'AMICI_SUCCESS', 'AMICI_SUCCESS']
```

What happened?

AMICI failed to integrate the forward problem. The problem occurred for only one simulation condition, `condition_step_03_0`. The issue occurred at $t = 3031.8$, where the CVODES reached the maximum number of steps.

How to address?

The number of steps the solver has to take is closely related to the chosen error tolerance. More accurate results, more steps. Therefore, this problem can be solved in two ways:

1. Increasing the maximum number of steps via amici.Solver.setMaxSteps. Note that this will increase the time required for simulation, and that simulation may still fail eventually. Sometimes it may be preferable to not increase this limit but rather fail fast. Also note that increasing the number of allowed steps increase RAM requirements (even if fewer steps are actually taken), so don't set this to ridiculously large values in order to avoid this error.
2. Reducing the number of steps CVODES has to take. This is determined by the required error tolerance. There are various solver error tolerances than can be adjusted. The most relevant ones are those controlled via amici.Solver.setRelativeTolerance() and amici.Solver.setAbsoluteTolerance().

So, let's fix that:

[3]:
```python
# let's increase the allowed number of steps by 10x:
print("Increasing allowed number of steps ...")
amici_solver = amici_model.getSolver()
amici_solver.setMaxSteps(10 * amici_solver.getMaxSteps())

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver
)

print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
print("Simulations finished succesfully.")
print()


# let's relax the relative error tolerance by a factor of 50
print("Relaxing relative error tolerance ...")
amici_solver = amici_model.getSolver()
amici_solver.setRelativeTolerance(50 * amici_solver.getRelativeTolerance())

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
print("Simulations finished succesfully.")
```

```
Increasing allowed number of steps ...
Status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS']
Simulations finished succesfully.

Relaxing relative error tolerance ...
Status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS']
Simulations finished succesfully.
```

Internal t = [...] and h = [...] are such that t + h = t on the next step

Let's run a simulation:

```
[4]: petab_problem = benchmark_models_petab.get_problem("Crauste_CellSystems2017")
amici_model = import_petab_problem(petab_problem, verbose=False)

np.random.seed(1)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_TOO_MUCH_WORK']
```

```
2023-01-25 16:03:24.617 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.618 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.620 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.621 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.623 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.624 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.626 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.627 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
```

(continues on next page)

(continued from previous page)

```
2023-01-25 16:03:24.629 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.630 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : Internal t = 0.330112 and h␣
↪= 2.06577e-17 are such that t + h = t on the next step. The solver will continue␣
↪anyway.
2023-01-25 16:03:24.633 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪WARNING] AMICI ERROR: in module CVODES in function CVode : The above warning has been␣
↪issued mxhnil times and will not be issued again for this problem.
2023-01-25 16:03:24.635 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:TOO_
↪MUCH_WORK] AMICI ERROR: in module CVODES in function CVode : At t = 0.330112, mxstep␣
↪steps taken before reaching tout.
2023-01-25 16:03:24.636 - amici.swig_wrappers - ERROR - [model1_data1][FORWARD_FAILURE]␣
↪AMICI forward simulation failed at t = 0.330112: AMICI failed to integrate the forward␣
↪problem
```

```
Status: ['AMICI_TOO_MUCH_WORK']
```

What happened?

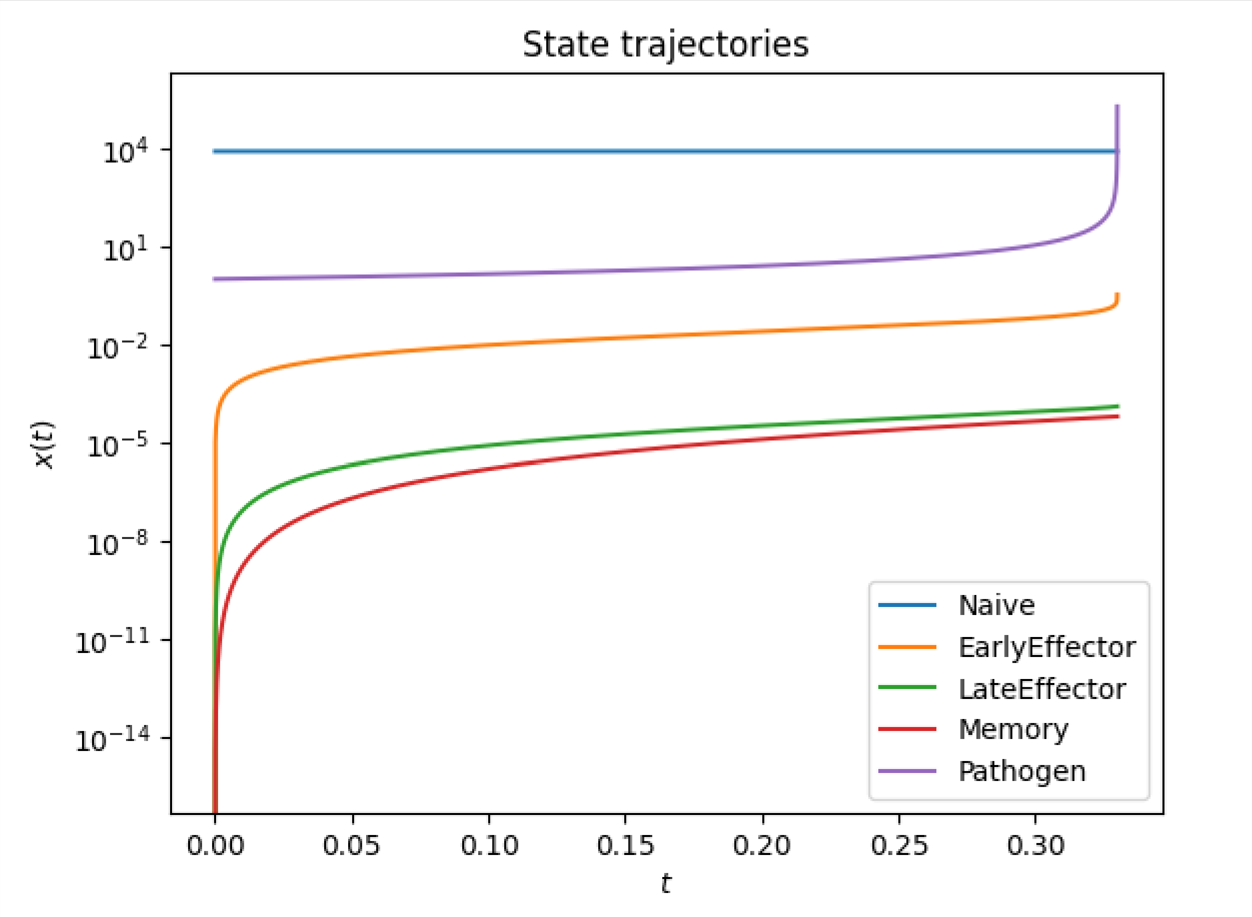
The forward simulation failed because AMICI the solver exceeded the maximum number of steps. Unlike in the previous case of `mxstep steps taken before reaching tout` (see above), here we got several additional warnings that the current step size h is numerically zero.

How to address?

The warning `Internal t = [...] and h = [...] are such that t + h = t on the next step` tells us that the solver is not able to move forward. The solver may be able to recover from that, but not always.

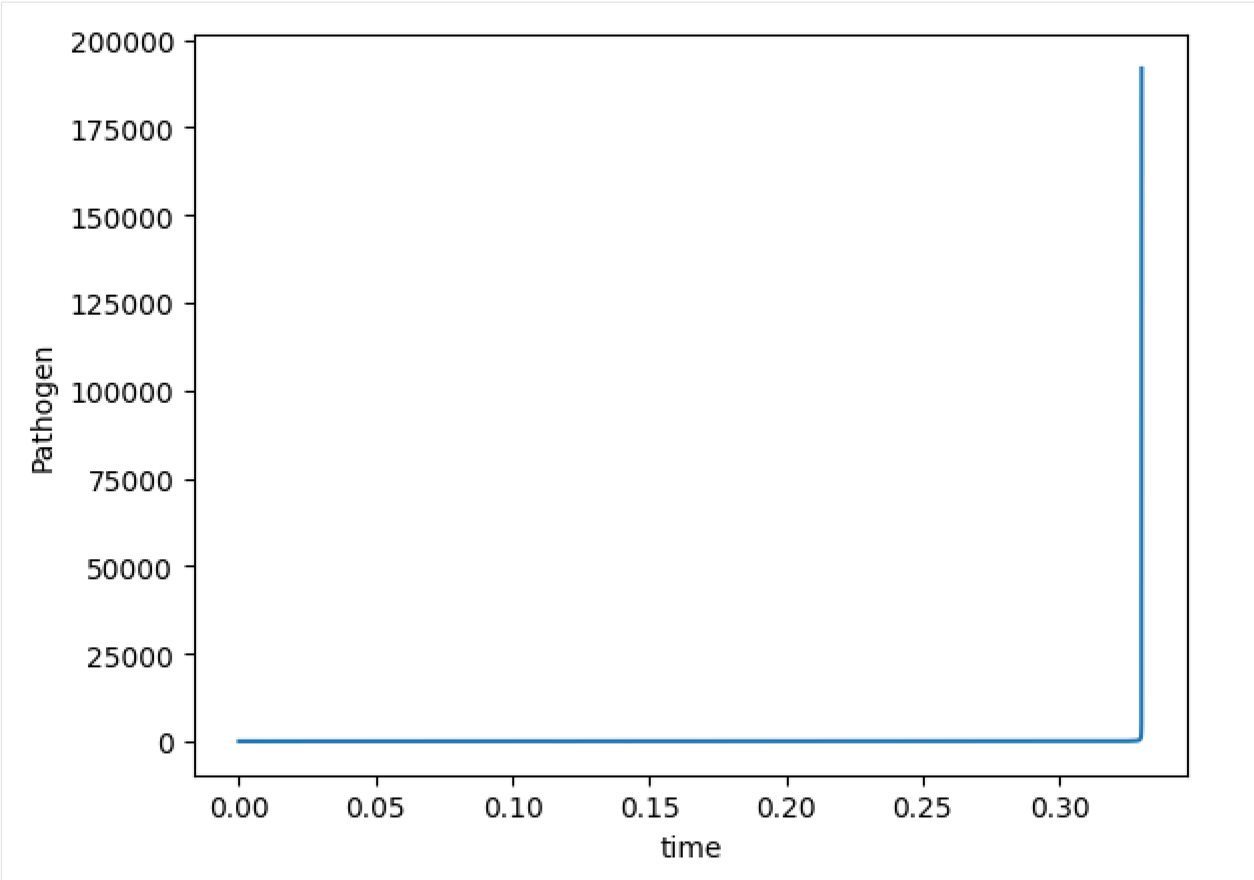
Let's look at the state trajectories to see what's going on. Such a tiny step size is usually related to very fast dynamics. We repeat the simulation with additional timepoints before the point of failure:

[5]:
```
# Create a copy of this simulation condition
edata = amici.ExpData(res[EDATAS][0])
edata.setTimepoints(np.linspace(0, 0.33011, 5000))
amici_solver = amici_model.getSolver()
rdata = amici.runAmiciSimulation(amici_model, amici_solver, edata)

# Visualize state trajectories
plot_state_trajectories(rdata, model=amici_model)
plt.yscale("log")
```



We can see a steep increase for `Pathogen` just before the error occurs. Let's zoom in:

[6]:
```
plt.plot(rdata.t, rdata.by_id("Pathogen"))
plt.xlabel("time")
plt.ylabel("Pathogen");
```



The solver is unable to handle such a steep increase. There is not much we can do. Increasing the tolerances will let the solver proceed a bit further, but this is usually not enough. Most likely there is a problem in the model or in the choice of parameter values.

the error test failed repeatedly or with |h| = hmin

Let's run a simulation:

[7]:
```
petab_problem = benchmark_models_petab.get_problem("Fujita_SciSignal2010")
amici_model = import_petab_problem(petab_problem, verbose=False)

np.random.seed(4920)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True
)
```

(continues on next page)

(continued from previous page)

```
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_SUCCESS', 'AMICI_ERR_FAILURE', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS
↪', 'AMICI_SUCCESS']
```

```
2023-01-25 16:03:25.895 - amici.swig_wrappers - DEBUG - [condition_step_00_3][CVODES:
↪CVode:ERR_FAILURE] AMICI ERROR: in module CVODES in function CVode : At t = 429.232␣
↪and h = 7.75194e-05, the error test failed repeatedly or with |h| = hmin.
2023-01-25 16:03:25.896 - amici.swig_wrappers - ERROR - [condition_step_00_3][FORWARD_
↪FAILURE] AMICI forward simulation failed at t = 429.232: AMICI failed to integrate the␣
↪forward problem
```

```
Status: ['AMICI_SUCCESS', 'AMICI_ERR_FAILURE', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS']
```

What happened?

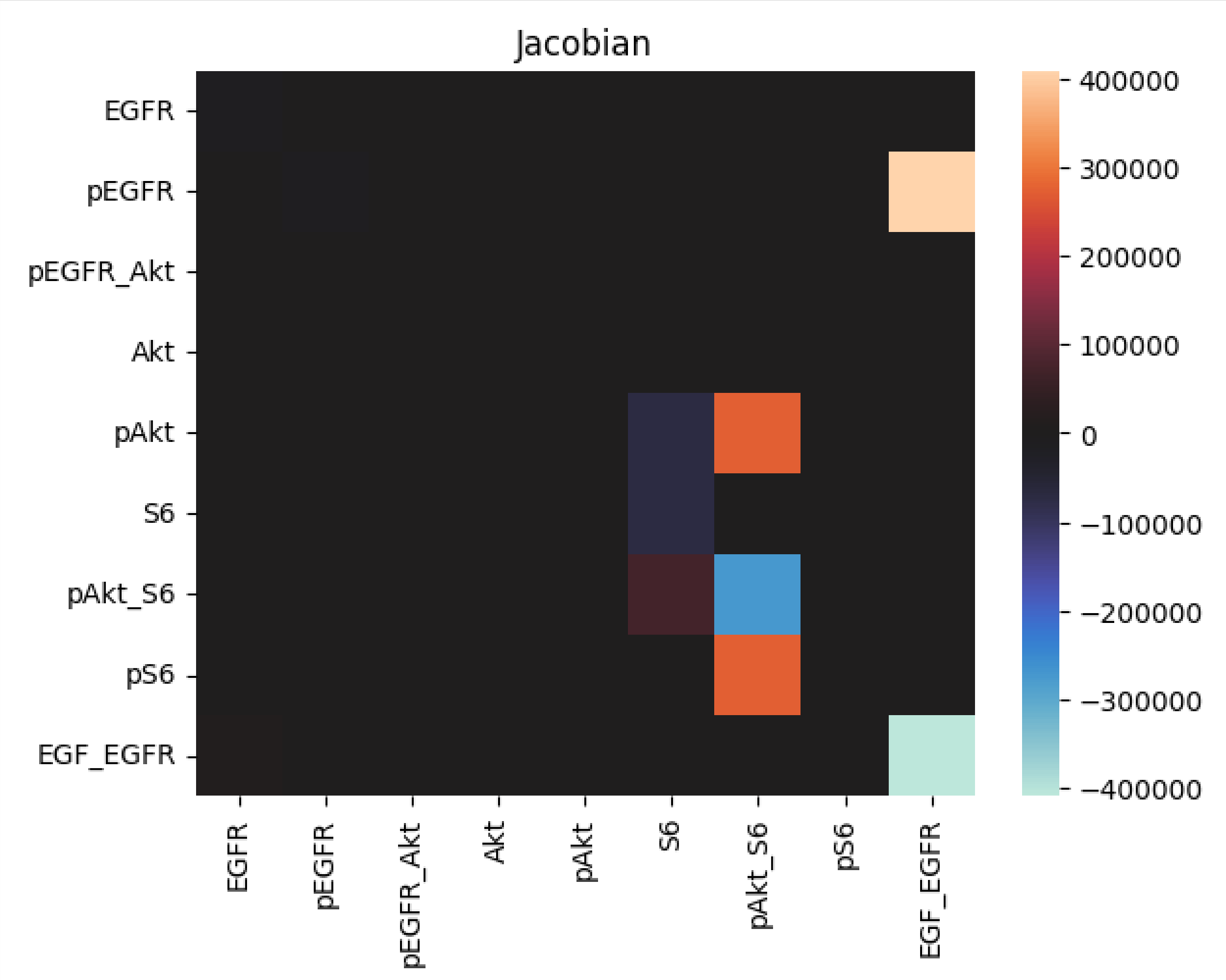
AMICI failed to integrate the forward problem. The problem occurred for only one simulation condition, `condition_step_00_3`. The issue occurred at $t = 429.232$, where the error test failed. This means, the solver is unable to take a step of non-zero size without violating the choosen error tolerances.

How to address?

The step size is computed based on the Jacobian. Inspecting `ReturnData.J` shows us that we have rather large values in the Jacobian:

[8]:
```
rdata = res[RDATAS][1]

# Show Jacobian as heatmap
plot_jacobian(rdata)

print(f"largest absolute Jacobian value: {np.max(np.abs(rdata.J)):.3g}")
```

```
largest absolute Jacobian value: 4.09e+05
```



In this case, the default relative error tolerance may be too high and lead too large absolute errors.

Let's retry simulation using stricter tolerances:

[9]:
```python
# set stricter relative error tolerance
amici_solver = amici_model.getSolver()
amici_solver.setRelativeTolerance(amici_solver.getRelativeTolerance() / 10)

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
print("Simulations finished succesfully.")
```

```
Status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS']
Simulations finished succesfully.
```

Cvode routine CVode returned a root after reinitialization

Let's run a simulation:

[10]:
```
petab_problem = benchmark_models_petab.get_problem("Weber_BMC2015")
amici_model = import_petab_problem(petab_problem, verbose=False, force_compile=False)

np.random.seed(4)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_ERROR', 'AMICI_SUCCESS']
```

```
2023-01-25 16:03:26.743 - amici.swig_wrappers - ERROR - [model1_data2+model1_
↪data1][OTHER] AMICI simulation failed: Cvode routine CVode returned a root after␣
↪reinitialization. The initial step-size after the event or heaviside function is too␣
↪small. To fix this, increase absolute and relative tolerances! failed with error code 2
Error occurred in:
2       0x7ff62c2b99d8 amici::CvodeException::CvodeException(int, char const*) + 24
3       0x7ff62c2a904f /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/documentation/amici_models/Weber_BMC2015/Weber_BMC2015/_Weber_BMC2015.cpython-
↪38-x86_64-linux-gnu.so(+0x4804f) [0x7ff62c2a904f]
4       0x7ff62c2e9ba6 amici::CVodeSolver::solve(double, int) const + 86
5       0x7ff63bbfd226 amici::Solver::run(double) const + 198
6       0x7ff63bbc38d6 amici::ForwardProblem::workForwar
```

```
Status: ['AMICI_ERROR', 'AMICI_SUCCESS']
```

What happened?

The simulation failed because the initial step-size after an event or heaviside function was too small. The error occured during simulation of condition `model1_data1` after successful preequilibration (`model1_data2`).

How to address?

The error message already suggests a fix for this situation, so let's try increasing the relative tolerance:

[11]:
```
amici_solver = amici_model.getSolver()
amici_solver.setRelativeTolerance(200 * amici_solver.getRelativeTolerance())

np.random.seed(4)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
```

(continues on next page)

(continued from previous page)

```
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
```

```
Status: ['AMICI_SUCCESS', 'AMICI_SUCCESS']
```

AMICI encountered a NaN / Inf value for [...]

Let's run a simulation:

[12]:

```
petab_problem = benchmark_models_petab.get_problem("Borghans_BiophysChem1997")
amici_model = import_petab_problem(petab_problem, verbose=False)

np.random.seed(18)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_FIRST_RHSFUNC_ERR']
```

```
2023-01-25 16:03:27.184 - amici.swig_wrappers - WARNING - [model1_data1][AMICI:NaN]␣
↪AMICI encountered a NaN value for xdot[2] (A_state)
2023-01-25 16:03:27.185 - amici.swig_wrappers - WARNING - [model1_data1][AMICI:NaN]␣
↪AMICI encountered a NaN value for w[6] (flux_v7_v_6)
2023-01-25 16:03:27.187 - amici.swig_wrappers - DEBUG - [model1_data1][CVODES:CVode:
↪OTHER] AMICI ERROR: in module CVODES in function CVode : The right-hand side routine␣
↪failed at the first call.
2023-01-25 16:03:27.188 - amici.swig_wrappers - ERROR - [model1_data1][FORWARD_FAILURE]␣
↪AMICI forward simulation failed at t = 0: AMICI failed to integrate the forward problem
```

```
Status: ['AMICI_FIRST_RHSFUNC_ERR']
```

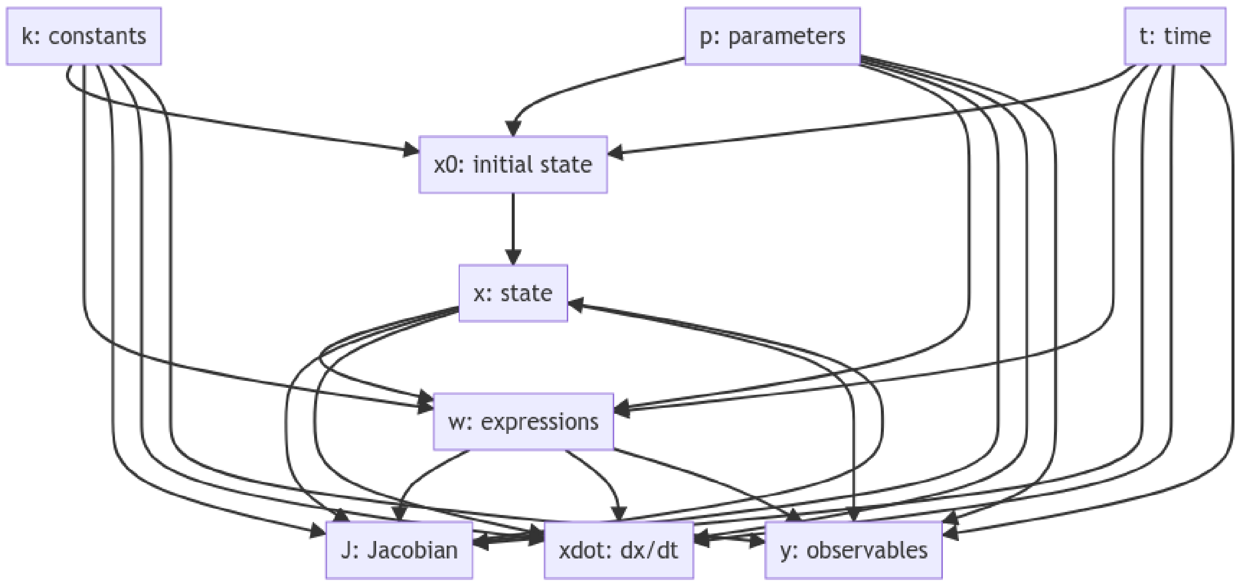
What happened?

The forward simulation failed because AMICI encountered a `NaN` value when simulating condition `model1_data1`.

Then NaNs occurred in $\dot{x}$ and w (model expressions, such as reaction fluxes or assignment rules). Furthermore, the failure occurred at the first call, so at $t = t_0$ (here: $t = 0$).

How to address?

The NaN in $\dot{x}$ is most likely a consequence of the one in w. (A subset of) the dependency tree looks something like:



Always look for the most basic (furthest up) model quantities. In cases where there non-finite values occur in expressions further down, rerunning the simulation after calling Model.setAlwaysCheckFinite(True) may give some further hints on where the issue originates.

The NaN in w occurred for flux_v7_v_6 (see error log), i.e., when computing the reaction flux for reaction v7_v_6. As w only depends on (t, p, k, x) and no non-finite values have been reported for those, the issue has to be in the respective flux equation.

Let's look at that expression. This can either be done by inspecting the underlying SBML model (e.g., using COPASI), or by checking the generated model code:

[13]:
```
# model name and source code location
model_name = amici_model.module.__package__
model_src_dir = Path(amici_model.module.__file__).parents[1]

# find the problematic expression in the model source code
!grep flux_v7_v_6 {model_src_dir}/{model_name}_w.cpp
```

```
    flux_v7_v_6 = 1.0*std::pow(A_state, 2)*Vd*std::pow(Z_state, n_par)/((std::pow(A_
↪state, 2) + std::pow(Kp, 2))*(std::pow(Kd, n_par) + std::pow(Z_state, n_par)));  //␣
↪w[6]
```

What could go wrong? We can obtain NaN from any of these symbols symbols being NaN, or through division by zero.

Let's let's check the denominator first:

$$(A\_state^2 + Kp^2) * (Kd^{n\_par} + Z\_state^{n\_par})$$

A_state and Z_state are state variables, Kd, K_p, and n_par are parameters.

As the error occurred at $t = t_0$, let's ensure the initial state is non-zero and finite:

[14]:
```
rdata = res[RDATAS][0]
edata = res[EDATAS][0]
# check initial states
x0 = dict(zip(amici_model.getStateIds(), rdata.x0))
print(f"{x0=}")
```

```
x0={'Z_state': 0.6869701913398437, 'Y_state': 0.2977237418558598, 'A_state': 0.
↪1116031306650328}
```

The initial states are fine - the first multiplicand is non-zero, as x_0 was non-zero.

So let's check the parameter values occurring in the second multiplicand:

[15]:
```
# we have to account for the chosen parameter scale
from itertools import starmap
unscaled_parameter = dict(zip(
    amici_model.getParameterIds(),
    starmap(amici.getUnscaledParameter, zip(edata.parameters, edata.pscale)),
))
print(dict((p, unscaled_parameter[p]) for p in ('Kd', 'Kp', 'n_par')))
```

```
{'Kd': 0.028491925689008366, 'Kp': 1002.513636749445, 'n_par': 7816.430091706722}
```

Considering that `n_par` occurrs as exponent, it's magnitude looks pretty high. This term is very likely causing the problem - let's check:

[16]:
```
print(f"{x0['Z_state']**unscaled_parameter['n_par'] + unscaled_parameter['Kd']**unscaled_
↪parameter['n_par']=}")
```

```
x0['Z_state']**unscaled_parameter['n_par'] + unscaled_parameter['Kd']**unscaled_
↪parameter['n_par']=0.0
```

Indeed, no way we can fix this for the given model. This was most likely an unrealistic parameter value, originating from a too high upper parameter bound for `n_par`. Therefore, if this error occurs during optimization, a first step could be adapting the respective parameter bounds. In other cases, this may be a result of unfortunate arrangement of model expressions, which can sometimes be solved by passing a suitable simplification function to the model import.

Steady state sensitivity computation failed due to unsuccessful factorization of RHS Jacobian

Let's run a simulation:

[17]:
```
petab_problem = benchmark_models_petab.get_problem("Blasi_CellSystems2016")
with suppress(KeyError):
    del os.environ["AMICI_EXPERIMENTAL_SBML_NONCONST_CLS"]
amici_model = import_petab_problem(
    petab_problem,
    verbose=False,
    force_compile=True,
    model_name="Blasi_CellSystems2016_1"
)

amici_solver = amici_model.getSolver()
amici_solver.setSensitivityMethod(amici.SensitivityMethod.forward)
```

(continues on next page)

(continued from previous page)

```
amici_solver.setSensitivityOrder(amici.SensitivityOrder.first)

np.random.seed(2020)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

# hard to reproduce on GHA
if os.getenv('GITHUB_ACTIONS') is None:
    assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_ERROR']
```

```
2023-01-25 16:03:49.913 - amici.swig_wrappers - ERROR - [control][OTHER] AMICI␣
↪simulation failed: Steady state sensitivity computation failed due to unsuccessful␣
↪factorization of RHS Jacobian
Error occurred in:
2       0x7ff63bac3da5 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6da5)␣
↪[0x7ff63bac3da5]
3       0x7ff63bbba73e amici::runAmiciSimulation(amici::Solver&, amici::ExpData const*,␣
↪amici::Model&, bool) + 2702
4       0x7ff63bbbaa67 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x1bda67)␣
↪[0x7ff63bbbaa67]
5       0x7ff63b9a88e6
```

```
Status: ['AMICI_ERROR']
```

What happened?

AMICI failed to compute steadystate sensitivities, because it was not able to factorize the Jacobian.

How to address?

This is most likely a result of a singular Jacobian. Let's check the condition number:

[18]:
```
rdata = res[RDATAS][0]
np.linalg.cond(rdata.J)
```

[18]:
```
3.01187419527941e+19
```

Indeed, the condition number shows that the Jacobian is numerically singular. If this happens consistently, it is usually due to conserved quantities in the model.

There are two ways we can address that:

1. Use numerical integration to compute sensitivities, for which a singular Jacobian is not an issue. This is, usually, slower, though.
2. Remove any conserved quantities.

Let's try both approaches:

[19]:
```
# use numerical integration
amici_model.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.
→integrationOnly)

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
```

```
Status: ['AMICI_SUCCESS']
```

[20]:
```
# Remove conserved quantities - this requires re-importing the model

# this is enabled by the `AMICI_EXPERIMENTAL_SBML_NONCONST_CLS` environment variable
os.environ["AMICI_EXPERIMENTAL_SBML_NONCONST_CLS"] = "1"
amici_model = import_petab_problem(
    petab_problem,
    verbose=False,
    # we need a different model name if we import the model again
    # we cannot load a model with the same name as an already loaded model
    model_name="Blasi_CellSystems2016_2",
    force_compile=True,
)
del os.environ["AMICI_EXPERIMENTAL_SBML_NONCONST_CLS"]

amici_solver = amici_model.getSolver()
amici_solver.setSensitivityMethod(amici.SensitivityMethod.forward)
amici_solver.setSensitivityOrder(amici.SensitivityOrder.first)

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)
print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
```

```
Status: ['AMICI_SUCCESS']
```

Steady state computation failed

Let's run a simulation:

[21]:
```
petab_problem = benchmark_models_petab.get_problem("Brannmark_JBC2010")
amici_model = import_petab_problem(
    petab_problem,
    verbose=False,
)

amici_solver = amici_model.getSolver()


np.random.seed(1851)
problem_parameters = dict(
    zip(
        petab_problem.x_free_ids,
        petab_problem.sample_parameter_startpoints(n_starts=1)[0],
    )
)
res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)

print("Status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

# hard to reproduce on GHA
if os.getenv('GITHUB_ACTIONS') is None:
    assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_ERROR', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS',
↪'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
```

```
2023-01-25 16:04:17.227 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][EQUILIBRATION_
↪FAILURE] AMICI equilibration failed: AMICI failed to integrate the forward problem
2023-01-25 16:04:17.228 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][OTHER] AMICI␣
↪simulation failed: Steady state computation failed. First run of Newton solver failed:␣
↪No convergence was achieved. Simulation to steady state failed. Second run of Newton␣
↪solver failed: No convergence was achieved.
Error occurred in:
2       0x7ff63bac3767 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6767)␣
↪[0x7ff63bac3767]
3       0x7ff63bc083ba /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x20b3ba)␣
↪[0x7ff63bc083ba]
4       0x7ff63bc083ff amici::SteadystateProblem::workSteadyStateProblem(amici::Solver␣
↪const&, amici::Model&, int) + 63
5       0x7ff63bbb
```

```
Status: ['AMICI_ERROR', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS
↪', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
```

(continues on next page)

(continued from previous page)

What happened?

All given experimental conditions require pre-equilibration, i.e., finding a steady state. AMICI first tries find a steady state using the Newton solver, if that fails, it tries simulating until steady state, if that also failes, it tries the Newton solver from the end of the simulation. In this case, all three failed. Neither Newton's method nor simulation yielded a steadystate satisfying the required tolerances.

This can also be seen in `ReturnDataView.preeq_status` (the three statuses corresponds to Newton #1, Simulation, Newton #2):

[22]:
```
rdata = res[RDATAS][0]
list(map(amici.SteadyStateStatus, rdata.preeq_status.flatten()))
```

[22]:
```
[<SteadyStateStatus.failed_convergence: -2>,
 <SteadyStateStatus.failed: -1>,
 <SteadyStateStatus.failed_convergence: -2>]
```

How to address?

There are several ways to address that:

1. Stricter integration tolerances (preferred if affordable - higher accuracy, but generally slower)
2. Looser steadystate tolerances (lower accuracy, generally faster)
3. Increase the number of allowed steps for Newton's method

Let's try all of them:

[23]:
```
# Reduce relative tolerance for integration
amici_solver = amici_model.getSolver()
amici_solver.setRelativeTolerance(1/100 * amici_solver.getRelativeTolerance())

res = simulate_petab(
    petab_problem=petab_problem,
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)
print("status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

rdata = res[RDATAS][0]
print(f"preeq_status={list(map(amici.SteadyStateStatus, rdata.preeq_status.flatten()))}")
print(f"{rdata.preeq_numsteps=}")

# hard to reproduce on GHA
if os.getenv('GITHUB_ACTIONS') is None:
    assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
```

```
status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
preeq_status=[<SteadyStateStatus.failed_convergence: -2>, <SteadyStateStatus.success: 1>,
↪ <SteadyStateStatus.not_run: 0>]
rdata.preeq_numsteps=array([[   0, 3150,    0]])
```

[24]:
```
# Increase relative steady state tolerance
for log10_relaxation_factor in range(1, 10):
    print(f"Relaxing tolerances by factor {10 ** log10_relaxation_factor}")
    amici_solver = amici_model.getSolver()
    amici_solver.setRelativeToleranceSteadyState(amici_solver.
↪getRelativeToleranceSteadyState() * 10 ** log10_relaxation_factor)

    res = simulate_petab(
        petab_problem=petab_problem,
        amici_model=amici_model,
        problem_parameters=problem_parameters,
        scaled_parameters=True,
        solver=amici_solver,
    )
    if all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS]):
        print(f"-> Succeeded with relative steady state tolerance {amici_solver.
↪getRelativeToleranceSteadyState()}\n")
        break
    else:
        print("-> Failed.\n")

print("status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

rdata = res[RDATAS][0]
print(f"preeq_status={list(map(amici.SteadyStateStatus, rdata.preeq_status.flatten()))}")
print(f"{rdata.preeq_numsteps=}")
assert all(rdata.status == amici.AMICI_SUCCESS for rdata in res[RDATAS])
```

```
Relaxing tolerances by factor 10
```

```
2023-01-25 16:04:17.763 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][EQUILIBRATION_
↪FAILURE] AMICI equilibration failed: AMICI failed to integrate the forward problem
2023-01-25 16:04:17.764 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][OTHER] AMICI␣
↪simulation failed: Steady state computation failed. First run of Newton solver failed:␣
↪No convergence was achieved. Simulation to steady state failed. Second run of Newton␣
↪solver failed: No convergence was achieved.
Error occurred in:
2       0x7ff63bac3767 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6767)␣
↪[0x7ff63bac3767]
3       0x7ff63bc083ba /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x20b3ba)␣
↪[0x7ff63bc083ba]
4       0x7ff63bc083ff amici::SteadystateProblem::workSteadyStateProblem(amici::Solver␣
↪const&, amici::Model&, int) + 63
5       0x7ff63bbb
```

```
-> Failed.

Relaxing tolerances by factor 100
```

```
2023-01-25 16:04:17.986 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][EQUILIBRATION_
↪FAILURE] AMICI equilibration failed: AMICI failed to integrate the forward problem
2023-01-25 16:04:17.987 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][OTHER] AMICI␣
↪simulation failed: Steady state computation failed. First run of Newton solver failed:␣
↪No convergence was achieved. Simulation to steady state failed. Second run of Newton␣
↪solver failed: No convergence was achieved.
```

(continues on next page)

(continued from previous page)

```
Error occurred in:
2       0x7ff63bac3767 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6767)␣
↪[0x7ff63bac3767]
3       0x7ff63bc083ba /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x20b3ba)␣
↪[0x7ff63bc083ba]
4       0x7ff63bc083ff amici::SteadystateProblem::workSteadyStateProblem(amici::Solver␣
↪const&, amici::Model&, int) + 63
5       0x7ff63bbb
```

```
-> Failed.

Relaxing tolerances by factor 1000
```

```
2023-01-25 16:04:18.210 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][EQUILIBRATION_
↪FAILURE] AMICI equilibration failed: AMICI failed to integrate the forward problem
2023-01-25 16:04:18.211 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][OTHER] AMICI␣
↪simulation failed: Steady state computation failed. First run of Newton solver failed:␣
↪No convergence was achieved. Simulation to steady state failed. Second run of Newton␣
↪solver failed: No convergence was achieved.
Error occurred in:
2       0x7ff63bac3767 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6767)␣
↪[0x7ff63bac3767]
3       0x7ff63bc083ba /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x20b3ba)␣
↪[0x7ff63bc083ba]
4       0x7ff63bc083ff amici::SteadystateProblem::workSteadyStateProblem(amici::Solver␣
↪const&, amici::Model&, int) + 63
5       0x7ff63bbb
```

```
-> Failed.

Relaxing tolerances by factor 10000
-> Succeeded with relative steady state tolerance 0.01

status: ['AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_
↪SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
preeq_status=[<SteadyStateStatus.failed_convergence: -2>, <SteadyStateStatus.success: 1>,
↪ <SteadyStateStatus.not_run: 0>]
rdata.preeq_numsteps=array([[  0, 577,   0]])
```

That fixed the error, and took only a quarter of the number steps as the previous run, but at the cost of much lower accuracy.

[25]:
```
# Let's try increasing the number of Newton steps
# (this is 0 by default, so the Newton solver wasn't used before,
#  as can be seen from the 0 in `rdata.preeq_numsteps[0]`)
amici_solver = amici_model.getSolver()
amici_solver.setNewtonMaxSteps(10**4)

res = simulate_petab(
    petab_problem=petab_problem,
```

(continues on next page)

(continued from previous page)

```
    amici_model=amici_model,
    problem_parameters=problem_parameters,
    scaled_parameters=True,
    solver=amici_solver,
)
print("status:", [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]])

rdata = res[RDATAS][0]
print(f"preeq_status={list(map(amici.SteadyStateStatus, rdata.preeq_status.flatten()))}")
print(f"{rdata.preeq_numsteps=}")
# hard to reproduce on GHA
if os.getenv('GITHUB_ACTIONS') is None:
    assert [amici.simulation_status_to_str(rdata.status) for rdata in res[RDATAS]] == [
↪'AMICI_ERROR', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS',
↪'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
```

```
2023-01-25 16:04:18.677 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][EQUILIBRATION_
↪FAILURE] AMICI equilibration failed: AMICI failed to integrate the forward problem
2023-01-25 16:04:18.677 - amici.swig_wrappers - ERROR - [Dose_0+Dose_0][OTHER] AMICI␣
↪simulation failed: Steady state computation failed. First run of Newton solver failed:␣
↪RHS could not be factorized. Simulation to steady state failed. Second run of Newton␣
↪solver failed: RHS could not be factorized.
Error occurred in:
2       0x7ff63bac3767 /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0xc6767)␣
↪[0x7ff63bac3767]
3       0x7ff63bc083ba /home/docs/checkouts/readthedocs.org/user_builds/amici/checkouts/
↪v0.16.0/python/sdist/amici/_amici.cpython-38-x86_64-linux-gnu.so(+0x20b3ba)␣
↪[0x7ff63bc083ba]
4       0x7ff63bc083ff amici::SteadystateProblem::workSteadyStateProblem(amici::Solver␣
↪const&, amici::Model&, int) + 63
5       0x7ff63bbb
```

```
status: ['AMICI_ERROR', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS
↪', 'AMICI_SUCCESS', 'AMICI_SUCCESS', 'AMICI_SUCCESS']
preeq_status=[<SteadyStateStatus.failed_factorization: -3>, <SteadyStateStatus.failed: -
↪1>, <SteadyStateStatus.failed_factorization: -3>]
rdata.preeq_numsteps=array([[   0, 1105,    0]])
```

Increasing the maximum number of Newton steps doesn't seem to help here. The Jacobian was numerically singular and its factorization failed. This can be a result of conserved quantities in the model. Section *Steady state sensitivity computation failed due to unsuccessful factorization of RHS Jacobian* shows how to address that.

10.2.3 Environment variables affecting model import

In addition to the environment variables listed *here*, the following environment variables control various behaviours during model import and compilation:

Table 1: Environment variables affecting model import

Variable	Purpose	Example
AMICI_EXTRACT_CSE	Extract common subexpressions. May significantly reduce file size and compile time for large models, but makes the generated code less readable. Disabled by default.	AMICI_EXTRACT_CSE=1
AMICI_IMPORT_NPROCS	Number of processes to be used for model import. Defaults to 1. Speeds up import of large models. Will slow down import of small models, benchmarking recommended.	AMICI_IMPORT_NPROCS=4
AMICI_EXPERIMENTAL_SBML_NONCONST_CLS	Compute conservation laws for non-constant species. SBML-import only. See *amici.sbml_import.SbmlImporter.sbml2amici()*.	

10.2.4 Miscellaneous

OpenMP support for parallelized simulation for multiple experimental conditions

AMICI can be built with OpenMP support, which allows to parallelize model simulations for multiple experimental conditions.

On Linux and OSX this is enabled by default. This can be verified using:

```
import amici
amici.compiledWithOpenMP()
```

If not already enabled by default, you can enable OpenMP support by setting the environment variables AMICI_CXXFLAGS and AMICI_LDFLAGS to the correct OpenMP flags of your compiler and linker, respectively. This has to be done for both AMICI package installation *and* model compilation. When using gcc on Linux, this would be:

```
# on your shell:
AMICI_CXXFLAGS=-fopenmp AMICI_LDFLAGS=-fopenmp pip3 install amici
```

```
# in python, before model compilation:
import os
os.environ['AMICI_CXXFLAGS'] = '-fopenmp'
os.environ['AMICI_LDFLAGS'] = '-fopenmp'
```

10.3 FAQ

Q: I am trying to install the AMICI Python package, but installation fails with something like

```
amici/src/cblas.cpp:16:13: fatal error: cblas.h: No such file or directory
#include <cblas.h>
         ^~~~~~~~~
compilation terminated.
error: command 'x86_64-linux-gnu-gcc' failed with exit status 1
```

A: You will have to install a CBLAS-compatible BLAS library and/or set `BLAS_CFLAGS` as described in the *installation guide*.

Q: Importing my model fails with something like `ImportError: _someModelName.cpython-37m-x86_64-linux-gnu.so: undefined symbol: omp_get_thread_num`.

A: You probably installed the AMICI package with OpenMP support, but did not have the relevant compiler/linker flags set when importing/building the model. See *here*.

10.4 AMICI Python API

Modules

amici	AMICI
amici.amici	Core C++ bindings This module encompasses the complete public C++ API of AMICI, which was exposed via swig. All functions listed here are directly accessible in the main amici package, i.e., amici.amici.ExpData is available as amici.ExpData. Usage of functions and classes from the base amici package is generally recommended as they often include convenience wrappers that avoid common pitfalls when accessing C++ types from python and implement some nonstandard type conversions.
amici.sbml_import	SBML Import This module provides all necessary functionality to import a model specified in the Systems Biology Markup Language (SBML).
amici.pysb_import	PySB Import This module provides all necessary functionality to import a model specified in the pysb.core.Model format.
amici.bngl_import	BNGL Import This module provides all necessary functionality to import a model specified in the BNGL format.
amici.petab_import	PEtab Import Import a model in the petab (https://github.com/PEtab-dev/PEtab) format into AMICI.
amici.petab_import_pysb	PySB-PEtab Import Import a model in the PySB-adapted petab (https://github.com/PEtab-dev/PEtab) format into AMICI.
amici.petab_objective	PEtab Objective Functionality related to running simulations or evaluating the objective function as defined by a PEtab problem
amici.petab_simulate	PEtab Simulate Functionality related to the use of AMICI for simulation with PEtab's Simulator class.
amici.import_utils	Miscellaneous functions related to model import, independent of any specific model format
amici.ode_export	C++ Export This module provides all necessary functionality specify an ODE model and generate executable C++ simulation code. The user generally won't have to directly call any function from this module as this will be done by amici.pysb_import.pysb2amici(), amici.sbml_import.SbmlImporter.sbml2amici() and amici.petab_import.import_model().
amici.ode_model	Objects for AMICI's internal ODE model representation
amici.plotting	Plotting Plotting related functions
amici.pandas	Pandas Wrappers This module contains convenience wrappers that allow for easy interconversion between C++ objects from amici.amici and pandas DataFrames
amici.logging	Logging This module provides custom logging functionality for other amici modules
amici.gradient_check	Finite Difference Check This module provides functions to automatically check correctness of amici computed sensitivities using finite difference approximations

10.4.1 amici

AMICI

The AMICI Python module provides functionality for importing SBML or PySB models and turning them into C++ Python extensions.

var amici_path absolute root path of the amici repository or Python package

var amiciSwigPath absolute path of the amici swig directory

var amiciSrcPath absolute path of the amici source directory

var amiciModulePath absolute root path of the amici module

var hdf5_enabled boolean indicating if amici was compiled with hdf5 support

var has_clibs boolean indicating if this is the full package with swig interface or the raw package without

Classes

ModelModule(*args, **kwargs)	Enable Python static type checking for AMICI-generated model modules
add_path(path)	Context manager for temporarily changing PYTHONPATH

amici.ModelModule

class amici.**ModelModule**(**args*, ***kwargs*)

Enable Python static type checking for AMICI-generated model modules

__init__(**args*, ***kwargs*)

Methods Summary

__init__(*args, **kwargs)	
getModel()	**rtype** *amici.amici.Model*
get_model()	**rtype** *amici.amici.Model*

Methods

__init__(**args*, ***kwargs*)

getModel()

Return type *amici.amici.Model*

get_model()

Return type *amici.amici.Model*

amici.add_path

class amici.**add_path**(*path*)

Context manager for temporarily changing PYTHONPATH

__init__(*path*)

Methods Summary

__init__(path)	

Methods

__init__(*path*)

Functions Summary

import_model_module(module_name[, module_path])	Import Python module of an AMICI model

Functions

amici.**import_model_module**(*module_name*, *module_path=None*)

Import Python module of an AMICI model

Parameters

- **module_name** (str) – Name of the python package of the model
- **module_path** (typing.Union[pathlib.Path, str, None]) – Absolute or relative path of the package directory

Return type *amici.ModelModule*

Returns The model module

10.4.2 amici.amici

Core C++ bindings

This module encompasses the complete public C++ API of AMICI, which was exposed via swig. All functions listed here are directly accessible in the main amici package, i.e., *amici.amici.ExpData* is available as `amici.ExpData`. Usage of functions and classes from the base `amici` package is generally recommended as they often include convenience wrappers that avoid common pitfalls when accessing C++ types from python and implement some nonstandard type conversions.

Classes

BoolVector(*args)	Swig-Generated class templating common python types including `Iterable` [`bool`] and `numpy.array` [`bool`] to facilitate interfacing with C++ bindings.
DoubleVector(*args)	Swig-Generated class templating common python types including `Iterable` [`float`] and `numpy.array` [`float`] to facilitate interfacing with C++ bindings.
ExpData(*args)	ExpData carries all information about experimental or condition-specific data
ExpDataPtr(*args)	Swig-Generated class that implements smart pointers to ExpData as objects.
ExpDataPtrVector(*args)	Swig-Generated class templating common python types including `Iterable` [*amici.amici.ExpData*] and `numpy.array` [*amici.amici.ExpData*] to facilitate interfacing with C++ bindings.
FixedParameterContext(value)	An enumeration.
IntVector(*args)	Swig-Generated class templating common python types including `Iterable` [`int`] and `numpy.array` [`int`] to facilitate interfacing with C++ bindings.
InternalSensitivityMethod(value)	An enumeration.
InterpolationType(value)	An enumeration.
LinearMultistepMethod(value)	An enumeration.
LinearSolver(value)	An enumeration.
LogItem(*args)	A log item.
LogItemVector(*args)	
Logger()	A logger, holding a list of error messages.
Model(*args, **kwargs)	The Model class represents an AMICI ODE/DAE model.
ModelDimensions(*args)	Container for model dimensions.
ModelPtr(*args)	Swig-Generated class that implements smart pointers to Model as objects.
NewtonDampingFactorMode(value)	An enumeration.
NonlinearSolverIteration(value)	An enumeration.
ObservableScaling(value)	An enumeration.
ParameterScaling(value)	An enumeration.
ParameterScalingVector(*args)	
RDataReporting(value)	An enumeration.

continues on next page

Table 2 – continued from previous page

ReturnData(*args)	Stores all data to be returned by *amici.amici.runAmiciSimulation()*.
ReturnDataPtr(*args)	Swig-Generated class that implements smart pointers to ReturnData as objects.
SecondOrderMode(value)	An enumeration.
SensitivityMethod(value)	An enumeration.
SensitivityOrder(value)	An enumeration.
SimulationParameters(*args)	Container for various simulation parameters.
SimulationState()	implements an exchange format to store and transfer the state of a simulation at a specific timepoint.
Solver(*args, **kwargs)	The Solver class provides a generic interface to CVODES and IDAS solvers, individual realizations are realized in the CVodeSolver and the IDASolver class.
SolverPtr(*args)	Swig-Generated class that implements smart pointers to Solver as objects.
SteadyStateSensitivityMode(value)	An enumeration.
SteadyStateStatus(value)	An enumeration.
SteadyStateStatusVector(*args)	
StringDoubleMap(*args)	Swig-Generated class templating `Dict` [`str`, `float`] to facilitate interfacing with C++ bindings.
StringVector(*args)	Swig-Generated class templating common python types including `Iterable` [`str`] and `numpy.array` [`str`] to facilitate interfacing with C++ bindings.

amici.amici.BoolVector

class amici.amici.**BoolVector**(**args*)

Swig-Generated class templating common python types including `Iterable` [`bool`] and `numpy.array` [`bool`] to facilitate interfacing with C++ bindings.

amici.amici.DoubleVector

class amici.amici.**DoubleVector**(**args*)

Swig-Generated class templating common python types including `Iterable` [`float`] and `numpy.array` [`float`] to facilitate interfacing with C++ bindings.

amici.amici.ExpData

class amici.amici.**ExpData**(**args*)

ExpData carries all information about experimental or condition-specific data

__init__(**args*)

Overload 1:

default constructor

Overload 2:

Copy constructor, needs to be declared to be generated in swig

Overload 3:

constructor that only initializes dimensions

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track

Overload 4:

constructor that initializes timepoints from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)

Overload 5:

constructor that initializes timepoints and fixed parameters from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)
- **fixedParameters** (DoubleVector) – Model constants (dimension: nk)

Overload 6:

constructor that initializes timepoints and data from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)
- **observedData** (DoubleVector) – observed data (dimension: nt x nytrue, row-major)
- **observedDataStdDev** (DoubleVector) – standard deviation of observed data (dimension: nt x nytrue, row-major)
- **observedEvents** (DoubleVector) – observed events (dimension: nmaxevents x nztrue, row-major)
- **observedEventsStdDev** (DoubleVector) – standard deviation of observed events/roots (dimension: nmaxevents x nztrue, row-major)

Overload 7:

constructor that initializes with Model

Parameters **model** (*Model*) – pointer to model specification object

Overload 8:

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** (*ReturnData*) – return data pointer with stored simulation results
- **sigma_y** (*float*) – scalar standard deviations for all observables
- **sigma_z** (*float*) – scalar standard deviations for all event observables

Overload 9:

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** (*ReturnData*) – return data pointer with stored simulation results

- **sigma_y** (DoubleVector) – vector of standard deviations for observables (dimension: nytrue or nt x nytrue, row-major)
- **sigma_z** (DoubleVector) – vector of standard deviations for event observables (dimension: nztrue or nmaxevent x nztrue, row-major)

Methods Summary

`__init__`(*args)	*Overload 1:*
`getObservedData`()	get function that copies data from ExpData::observedData to output
`getObservedDataPtr`(it)	get function that returns a pointer to observed data at index
`getObservedDataStdDev`()	get function that copies data from ExpData::observedDataStdDev to output
`getObservedDataStdDevPtr`(it)	get function that returns a pointer to standard deviation of observed data at index
`getObservedEvents`()	get function that copies data from ExpData::mz to output
`getObservedEventsPtr`(ie)	get function that returns a pointer to observed data at ieth occurrence
`getObservedEventsStdDev`()	get function that copies data from ExpData::observedEventsStdDev to output
`getObservedEventsStdDevPtr`(ie)	get function that returns a pointer to standard deviation of observed event data at ie-th occurrence
`getTimepoint`(it)	get function that returns timepoint at index
`getTimepoints`()	get function that copies data from ExpData::ts to output
`isSetObservedData`(it, iy)	get function that checks whether data at specified indices has been set
`isSetObservedDataStdDev`(it, iy)	get function that checks whether standard deviation of data at specified indices has been set
`isSetObservedEvents`(ie, iz)	get function that checks whether event data at specified indices has been set
`isSetObservedEventsStdDev`(ie, iz)	get function that checks whether standard deviation of even data at specified indices has been set
`nmaxevent`()	maximal number of events to track
`nt`()	number of timepoints
`nytrue`()	number of observables of the non-augmented model
`nztrue`()	number of event observables of the non-augmented model
`reinitializeAllFixedParameterDependentInitialStates`(...)	Set reinitialization of all states based on model constants for all simulation phases.
`reinitializeAllFixedParameterDependentInitialStatesForPresimulation`(...)	Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).
`reinitializeAllFixedParameterDependentInitialStatesForSimulation`(...)	Set reinitialization of all states based on model constants for the 'main' simulation (only meaningful if presimulation or preequilibration is performed).
`setObservedData`(*args)	*Overload 1:*
`setObservedDataStdDev`(*args)	*Overload 1:*
`setObservedEvents`(*args)	*Overload 1:*
`setObservedEventsStdDev`(*args)	*Overload 1:*
`setTimepoints`(ts)	Set function that copies data from input to ExpData::ts

Attributes

`fixedParameters`	Model constants
`fixedParametersPreequilibration`	Model constants for pre-equilibration
`fixedParametersPresimulation`	Model constants for pre-simulation
`id`	Arbitrary (not necessarily unique) identifier.
`parameters`	Model parameters
`plist`	Parameter indices w.r.t.
`pscale`	Parameter scales
`reinitialization_state_idxs_presim`	Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.
`reinitialization_state_idxs_sim`	Indices of states to be reinitialized based on provided constants / fixed parameters.
`reinitializeFixedParameterInitialStates`	Flag indicating whether reinitialization of states depending on fixed parameters is activated
`sx0`	Initial state sensitivities
`t_presim`	Duration of pre-simulation.
`ts_`	Timepoints for which model state/outputs/.
`tstart_`	starting time
`x0`	Initial state

Methods

`__init__`(**args*)

Overload 1:

default constructor

Overload 2:

Copy constructor, needs to be declared to be generated in swig

Overload 3:

constructor that only initializes dimensions

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track

Overload 4:

constructor that initializes timepoints from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)

Overload 5:

constructor that initializes timepoints and fixed parameters from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)
- **fixedParameters** (DoubleVector) – Model constants (dimension: nk)

Overload 6:

constructor that initializes timepoints and data from vectors

Parameters

- **nytrue** (*int*) – Number of observables
- **nztrue** (*int*) – Number of event outputs
- **nmaxevent** (*int*) – Maximal number of events to track
- **ts** (DoubleVector) – Timepoints (dimension: nt)
- **observedData** (DoubleVector) – observed data (dimension: nt x nytrue, row-major)
- **observedDataStdDev** (DoubleVector) – standard deviation of observed data (dimension: nt x nytrue, row-major)
- **observedEvents** (DoubleVector) – observed events (dimension: nmaxevents x nztrue, row-major)
- **observedEventsStdDev** (DoubleVector) – standard deviation of observed events/roots (dimension: nmaxevents x nztrue, row-major)

Overload 7:

constructor that initializes with Model

Parameters **model** (*Model*) – pointer to model specification object

Overload 8:

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** (*ReturnData*) – return data pointer with stored simulation results
- **sigma_y** (*float*) – scalar standard deviations for all observables
- **sigma_z** (*float*) – scalar standard deviations for all event observables

Overload 9:

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** (*ReturnData*) – return data pointer with stored simulation results
- **sigma_y** (DoubleVector) – vector of standard deviations for observables (dimension: nytrue or nt x nytrue, row-major)
- **sigma_z** (DoubleVector) – vector of standard deviations for event observables (dimension: nztrue or nmaxevent x nztrue, row-major)

getObservedData() → *amici.amici.DoubleVector*

get function that copies data from ExpData::observedData to output

Return type *DoubleVector*

Returns observed data (dimension: nt x nytrue, row-major)

getObservedDataPtr(*it: int*) → Iterable[float]

get function that returns a pointer to observed data at index

Parameters **it** (int) – timepoint index

Return type float

Returns pointer to observed data at index (dimension: nytrue)

getObservedDataStdDev() → *amici.amici.DoubleVector*

get function that copies data from ExpData::observedDataStdDev to output

Return type *DoubleVector*

Returns standard deviation of observed data

getObservedDataStdDevPtr(*it: int*) → Iterable[float]

get function that returns a pointer to standard deviation of observed data at index

Parameters **it** (int) – timepoint index

Return type float

Returns pointer to standard deviation of observed data at index

getObservedEvents() → *amici.amici.DoubleVector*

get function that copies data from ExpData::mz to output

Return type *DoubleVector*

Returns observed event data

getObservedEventsPtr(*ie: int*) → Iterable[float]

get function that returns a pointer to observed data at ieth occurrence

Parameters **ie** (int) – event occurrence

Return type float

Returns pointer to observed event data at ieth occurrence

getObservedEventsStdDev() → *amici.amici.DoubleVector*

get function that copies data from ExpData::observedEventsStdDev to output

Return type *DoubleVector*

Returns standard deviation of observed event data

getObservedEventsStdDevPtr(*ie: int*) → Iterable[float]

get function that returns a pointer to standard deviation of observed event data at ie-th occurrence

Parameters **ie** (int) – event occurrence

Return type float

Returns pointer to standard deviation of observed event data at ie-th occurrence

getTimepoint(*it: int*) → float

get function that returns timepoint at index

Parameters **it** (int) – timepoint index

Return type float

Returns timepoint timepoint at index

getTimepoints() → *amici.amici.DoubleVector*

get function that copies data from ExpData::ts to output

Return type *DoubleVector*

Returns ExpData::ts

isSetObservedData(*it: int*, *iy: int*) → bool

get function that checks whether data at specified indices has been set

Parameters

- **it** (int) – time index
- **iy** (int) – observable index

Return type boolean

Returns boolean specifying if data was set

isSetObservedDataStdDev(*it: int*, *iy: int*) → bool

get function that checks whether standard deviation of data at specified indices has been set

Parameters

- **it** (int) – time index
- **iy** (int) – observable index

Return type boolean

Returns boolean specifying if standard deviation of data was set

isSetObservedEvents(*ie: int*, *iz: int*) → bool

get function that checks whether event data at specified indices has been set

Parameters

- **ie** (int) – event index
- **iz** (int) – event observable index

Return type boolean

Returns boolean specifying if data was set

isSetObservedEventsStdDev(*ie: int*, *iz: int*) → bool

get function that checks whether standard deviation of even data at specified indices has been set

Parameters

- **ie** (int) – event index
- **iz** (int) – event observable index

Return type boolean

Returns boolean specifying if standard deviation of event data was set

nmaxevent() → int

maximal number of events to track

Return type int

Returns maximal number of events to track

nt() → int

number of timepoints

Return type int

Returns number of timepoints

nytrue() → int

number of observables of the non-augmented model

Return type int

Returns number of observables of the non-augmented model

nztrue() → int

number of event observables of the non-augmented model

Return type int

Returns number of event observables of the non-augmented model

reinitializeAllFixedParameterDependentInitialStates(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for all simulation phases.

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (`int`) – Number of states (Model::nx_rdata)

Return type `None`

reinitializeAllFixedParameterDependentInitialStatesForPresimulation(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (`int`) – Number of states (Model::nx_rdata)

Return type `None`

reinitializeAllFixedParameterDependentInitialStatesForSimulation(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for the 'main' simulation (only meaningful if presimulation or preequilibration is performed).

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (`int`) – Number of states (Model::nx_rdata)

Return type `None`

setObservedData(**args*) → None

Overload 1:

set function that copies data from input to ExpData::my

Parameters **observedData** (`DoubleVector`) – observed data (dimension: nt x nytrue, row-major)

Overload 2:

set function that copies observed data for specific observable

Parameters

- **observedData** (`DoubleVector`) – observed data (dimension: nt)
- **iy** (*int*) – observed data index

Return type `None`

setObservedDataStdDev(**args*) → None

Overload 1:

set function that copies data from input to ExpData::observedDataStdDev

Parameters **observedDataStdDev** (`DoubleVector`) – standard deviation of observed data (dimension: nt x nytrue, row-major)

Overload 2:

set function that sets all ExpData::observedDataStdDev to the input value

Parameters **stdDev** (*float*) – standard deviation (dimension: scalar)

Overload 3:

set function that copies standard deviation of observed data for specific observable

Parameters

- **observedDataStdDev** (DoubleVector) – standard deviation of observed data (dimension: nt)
- **iy** (*int*) – observed data index

Overload 4:

set function that sets all standard deviation of a specific observable to the input value

Parameters

- **stdDev** (*float*) – standard deviation (dimension: scalar)
- **iy** (*int*) – observed data index

Return type None

setObservedEvents(**args*) → None

Overload 1:

set function that copies observed event data from input to ExpData::observedEvents

Parameters **observedEvents** (DoubleVector) – observed data (dimension: nmaxevent x nztrue, row-major)

Overload 2:

set function that copies observed event data for specific event observable

Parameters

- **observedEvents** (DoubleVector) – observed data (dimension: nmaxevent)
- **iz** (*int*) – observed event data index

Return type None

setObservedEventsStdDev(**args*) → None

Overload 1:

set function that copies data from input to ExpData::observedEventsStdDev

Parameters **observedEventsStdDev** (DoubleVector) – standard deviation of observed event data

Overload 2:

set function that sets all ExpData::observedDataStdDev to the input value

Parameters **stdDev** (*float*) – standard deviation (dimension: scalar)

Overload 3:

set function that copies standard deviation of observed data for specific observable

Parameters

- **observedEventsStdDev** (DoubleVector) – standard deviation of observed data (dimension: nmaxevent)
- **iz** (*int*) – observed data index

Overload 4:

set function that sets all standard deviation of a specific observable to the input value

Parameters

- **stdDev** (*float*) – standard deviation (dimension: scalar)
- **iz** (*int*) – observed data index

Return type None

setTimepoints(*ts:* amici.amici.DoubleVector) → None

Set function that copies data from input to ExpData::ts

Parameters **ts** (*amici.amici.DoubleVector*) – timepoints

Return type None

amici.amici.ExpDataPtr

class amici.amici.**ExpDataPtr**(**args*)

Swig-Generated class that implements smart pointers to ExpData as objects.

Attributes

fixedParameters	Model constants
fixedParametersPreequilibration	Model constants for pre-equilibration
fixedParametersPresimulation	Model constants for pre-simulation
id	Arbitrary (not necessarily unique) identifier.
parameters	Model parameters
plist	Parameter indices w.r.t.
pscale	Parameter scales
reinitialization_state_idxs_presim	Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.
reinitialization_state_idxs_sim	Indices of states to be reinitialized based on provided constants / fixed parameters.
reinitializeFixedParameterInitialStates	Flag indicating whether reinitialization of states depending on fixed parameters is activated
sx0	Initial state sensitivities
t_presim	Duration of pre-simulation.
ts_	Timepoints for which model state/outputs/.
tstart_	starting time
x0	Initial state

amici.amici.ExpDataPtrVector

class amici.amici.**ExpDataPtrVector**(**args*)

Swig-Generated class templating common python types including Iterable [*amici.amici.ExpData*] and numpy.array [*amici.amici.ExpData*] to facilitate interfacing with C++ bindings.

amici.amici.FixedParameterContext

class amici.amici.**FixedParameterContext**(*value*)

An enumeration.

Attributes

simulation
preequilibration
presimulation

amici.amici.IntVector

class amici.amici.**IntVector**(**args*)

Swig-Generated class templating common python types including `Iterable` [`int`] and `numpy.array` [`int`] to facilitate interfacing with C++ bindings.

amici.amici.InternalSensitivityMethod

class amici.amici.**InternalSensitivityMethod**(*value*)

An enumeration.

Attributes

`simultaneous`
`staggered`
`staggered1`

amici.amici.InterpolationType

class amici.amici.**InterpolationType**(*value*)

An enumeration.

Attributes

`hermite`
`polynomial`

amici.amici.LinearMultistepMethod

class amici.amici.**LinearMultistepMethod**(*value*)

An enumeration.

Attributes

`adams`
`BDF`

amici.amici.LinearSolver

class amici.amici.**LinearSolver**(*value*)

An enumeration.

Attributes

dense
band
LAPACKDense
LAPACKBand
diag
SPGMR
SPBCG
SPTFQMR
KLU
SuperLUMT

amici.amici.LogItem

class amici.amici.**LogItem**(**args*)

A log item.

__init__(**args*)

Overload 1:

Default ctor.

Overload 2:

Construct a LogItem

Parameters

- **severity** (*int*) –
- **identifier** (*str*) –
- **message** (*str*) –

Methods Summary

`__init__`(*args)	*Overload 1:*

Attributes

`identifier`	Short identifier for the logged event
`message`	A more detailed and readable message
`severity`	Severity level

Methods

`__init__`(**args*)

Overload 1:

Default ctor.

Overload 2:

Construct a LogItem

Parameters

- **severity** (*int*) –
- **identifier** (*str*) –
- **message** (*str*) –

amici.amici.LogItemVector

`class amici.amici.`**`LogItemVector`**(**args*)

amici.amici.Logger

`class amici.amici.`**`Logger`**

A logger, holding a list of error messages.

`__init__`()

Methods Summary

__init__()	
log(*args)	*Overload 1:*

Attributes

`items`	The log items

Methods

__init__()

log(**args*) → None

Overload 1:

Add a log entry

Parameters

- **severity** (*int*) – Severity level
- **identifier** (*str*) – Short identifier for the logged event
- **message** (*str*) – A more detailed message

Overload 2:

Add a log entry with printf-like message formatting

Parameters

- **severity** (*int*) – Severity level
- **identifier** (*str*) – Short identifier for the logged event
- **format** (*string*) – printf format string

Return type None

amici.amici.Model

class amici.amici.**Model**(**args*, ***kwargs*)

The Model class represents an AMICI ODE/DAE model.

The model can compute various model related quantities based on symbolically generated code.

__init__(**args*, ***kwargs*)

Overload 1: Default ctor

Overload 2:

Constructor with model dimensions

Parameters

- **nx_rdata** (*int*) – Number of state variables
- **nxtrue_rdata** (*int*) – Number of state variables of the non-augmented model
- **nx_solver** (*int*) – Number of state variables with conservation laws applied
- **nxtrue_solver** (*int*) – Number of state variables of the non-augmented model with conservation laws applied
- **nx_solver_reinit** (*int*) – Number of state variables with conservation laws subject to reinitialization
- **np** (*int*) – Number of parameters
- **nk** (*int*) – Number of constants
- **ny** (*int*) – Number of observables
- **nytrue** (*int*) – Number of observables of the non-augmented model
- **nz** (*int*) – Number of event observables
- **nztrue** (*int*) – Number of event observables of the non-augmented model
- **ne** (*int*) – Number of events
- **nJ** (*int*) – Number of objective functions
- **nw** (*int*) – Number of repeating elements
- **ndwdx** (*int*) – Number of nonzero elements in the *x* derivative of the repeating elements
- **ndwdp** (*int*) – Number of nonzero elements in the *p* derivative of the repeating elements
- **ndwdw** (*int*) – Number of nonzero elements in the *w* derivative of the repeating elements
- **ndxdotdw** (*int*) – Number of nonzero elements in the w derivative of $xdot$
- **ndJydy** (IntVector) – Number of nonzero elements in the y derivative of dJy (shape *nytrue*)
- **ndxrdatadxsolver** (*int*) – Number of nonzero elements in the x derivative of $x_r data$
- **ndxrdatadtcl** (*int*) – Number of nonzero elements in the tcl derivative of $x_r data$
- **ndtotal_cldx_rdata** (*int*) – Number of nonzero elements in the $x_r data$ derivative of $total_c l$
- **nnz** (*int*) – Number of nonzero elements in Jacobian
- **ubw** (*int*) – Upper matrix bandwidth in the Jacobian
- **lbw** (*int*) – Lower matrix bandwidth in the Jacobian

Methods Summary

__init__(*args, **kwargs)	*Overload 1:*
clone()	Clone this instance.
fdsigmaydy(dsigmaydy, t, p, k, y)	Model-specific implementation of fsigmay
fdtotal_cldp(dtotal_cldp, x_rdata, p, k, ip)	Compute dtotal_cl / dp
fdtotal_cldx_rdata(dtotal_cldx_rdata, ...)	Compute dtotal_cl / dx_rdata
fdx_rdatadp(dx_rdatadp, x, tcl, p, k, ip)	Compute dx_rdata / dp
fdx_rdatadtcl(dx_rdatadtcl, x, tcl, p, k)	Compute dx_rdata / dtcl
fdx_rdatadx_solver(dx_rdatadx_solver, x, ...)	Compute dx_rdata / dx_solver
getAddSigmaResiduals()	Checks whether residuals should be added to account for parameter dependent sigma.
getAlwaysCheckFinite()	Get setting of whether the result of every call to *Model::f** should be checked for finiteness.
getAmiciCommit()	Returns the AMICI commit that was used to generate the model
getAmiciVersion()	Returns the AMICI version that was used to generate the model
getExpressionIds()	Get IDs of the expression.
getExpressionNames()	Get names of the expressions.
getFixedParameterById(par_id)	Get value of fixed parameter with the specified ID.
getFixedParameterByName(par_name)	Get value of fixed parameter with the specified name.
getFixedParameterIds()	Get IDs of the fixed model parameters.
getFixedParameterNames()	Get names of the fixed model parameters.
getFixedParameters()	Get values of fixed parameters.
getInitialStateSensitivities()	Get the initial states sensitivities.
getInitialStates()	Get the initial states.
getMinimumSigmaResiduals()	Gets the specified estimated lower boundary for sigma_y.
getName()	Get the model name.
getObservableIds()	Get IDs of the observables.
getObservableNames()	Get names of the observables.
getObservableScaling(iy)	Get scaling type for observable
getParameterById(par_id)	Get value of first model parameter with the specified ID.
getParameterByName(par_name)	Get value of first model parameter with the specified name.
getParameterIds()	Get IDs of the model parameters.
getParameterList()	Get the list of parameters for which sensitivities are computed.
getParameterNames()	Get names of the model parameters.
getParameterScale()	Get parameter scale for each parameter.
getParameters()	Get parameter vector.
getReinitializationStateIdxs()	Return indices of states to be reinitialized based on provided constants / fixed parameters
getReinitializeFixedParameterInitialStates()	Get whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation.
getSolver()	Retrieves the solver object
getStateIds()	Get IDs of the model states.
getStateIdsSolver()	Get IDs of the solver states.

continues on next page

Table 3 – continued from previous page

getStateIsNonNegative()	Get flags indicating whether states should be treated as non-negative.
getStateNames()	Get names of the model states.
getStateNamesSolver()	Get names of the solver states.
getSteadyStateSensitivityMode()	Gets the mode how sensitivities are computed in the steadystate simulation.
getTimepoint(it)	Get simulation timepoint for time index *it*.
getTimepoints()	Get the timepoint vector.
getUnscaledParameters()	Get parameters with transformation according to parameter scale applied.
hasCustomInitialStateSensitivities()	Return whether custom initial state sensitivities have been set.
hasCustomInitialStates()	Return whether custom initial states have been set.
hasExpressionIds()	Report whether the model has expression IDs set.
hasExpressionNames()	Report whether the model has expression names set.
hasFixedParameterIds()	Report whether the model has fixed parameter IDs set.
hasFixedParameterNames()	Report whether the model has fixed parameter names set.
hasObservableIds()	Report whether the model has observable IDs set.
hasObservableNames()	Report whether the model has observable names set.
hasParameterIds()	Report whether the model has parameter IDs set.
hasParameterNames()	Report whether the model has parameter names set.
hasQuadraticLLH()	Checks whether the defined noise model is gaussian, i.e., the nllh is quadratic
hasStateIds()	Report whether the model has state IDs set.
hasStateNames()	Report whether the model has state names set.
isFixedParameterStateReinitializationAllowed()	Function indicating whether reinitialization of states depending on fixed parameters is permissible
k()	Get fixed parameters.
nMaxEvent()	Get maximum number of events that may occur for each type.
ncl()	Get number of conservation laws.
nk()	Get number of constants
np()	Get total number of model parameters.
nplist()	Get number of parameters wrt to which sensitivities are computed.
nt()	Get number of timepoints.
nx_reinit()	Get number of solver states subject to reinitialization.
plist(pos)	Get entry in parameter list by index.
requireSensitivitiesForAllParameters()	Require computation of sensitivities for all parameters p [0..np[in natural order.
setAddSigmaResiduals(sigma_res)	Specifies whether residuals should be added to account for parameter dependent sigma.
setAllStatesNonNegative()	Set flags indicating that all states should be treated as non-negative.
setAlwaysCheckFinite(alwaysCheck)	Set whether the result of every call to *Model::f** should be checked for finiteness.
setFixedParameterById(par_id, value)	Set value of first fixed parameter with the specified ID.

continues on next page

Table 3 – continued from previous page

setFixedParameterByName(par_name, value)	Set value of first fixed parameter with the specified name.
setFixedParameters(k)	Set values for constants.
setFixedParametersByIdRegex(par_id_regex, value)	Set values of all fixed parameters with the ID matching the specified regex.
setFixedParametersByNameRegex(...)	Set value of all fixed parameters with name matching the specified regex.
setInitialStateSensitivities(sx0)	Set the initial state sensitivities.
setInitialStates(x0)	Set the initial states.
setMinimumSigmaResiduals(min_sigma)	Sets the estimated lower boundary for sigma_y.
setNMaxEvent(nmaxevent)	Set maximum number of events that may occur for each type.
setParameterById(*args)	*Overload 1:*
setParameterByName(*args)	*Overload 1:*
setParameterList(plist)	Set the list of parameters for which sensitivities are to be computed.
setParameterScale(*args)	**rtype** None
setParameters(p)	Set the parameter vector.
setParametersByIdRegex(par_id_regex, value)	Set all values of model parameters with IDs matching the specified regular expression.
setParametersByNameRegex(par_name_regex, value)	Set all values of all model parameters with names matching the specified regex.
setReinitializationStateIdxs(idxs)	Set indices of states to be reinitialized based on provided constants / fixed parameters
setReinitializeFixedParameterInitialStates(flag)	Set whether initial states depending on fixed parameters are to be reinitialized after preequilibration and presimulation.
setStateIsNonNegative(stateIsNonNegative)	Set flags indicating whether states should be treated as non-negative.
setSteadyStateSensitivityMode(mode)	Set the mode how sensitivities are computed in the steadystate simulation.
setT0(t0)	Set simulation start time.
setTimepoints(ts)	Set the timepoint vector.
setUnscaledInitialStateSensitivities(sx0)	Set the initial state sensitivities.
t0()	Get simulation start time.

Attributes

idlist	Flag array for DAE equations
lbw	Lower bandwidth of the Jacobian
logger	Logger
nJ	Dimension of the augmented objective function for 2nd order ASA
ndJydy	Number of nonzero elements in the y derivative of dJy (dimension *nytrue*)
ndtotal_cldx_rdata	Number of nonzero elements in the $x_r data$ derivative of $total_c l$
ndwdp	Number of nonzero elements in the *p* derivative of the repeating elements
ndwdw	Number of nonzero elements in the *w* derivative of the repeating elements
ndwdx	Number of nonzero elements in the *x* derivative of the repeating elements
ndxdotdw	Number of nonzero elements in the w derivative of $xdot$
ndxrdatadtcl	Number of nonzero elements in the tcl derivative of $x_r data$
ndxrdatadxsolver	Number of nonzero elements in the x derivative of $x_r data$
ne	Number of events
nnz	Number of nonzero entries in Jacobian
nw	Number of common expressions
nx_rdata	Number of states
nx_solver	Number of states with conservation laws applied
nx_solver_reinit	Number of solver states subject to reinitialization
nxtrue_rdata	Number of states in the unaugmented system
nxtrue_solver	Number of states in the unaugmented system with conservation laws applied
ny	Number of observables
nytrue	Number of observables in the unaugmented system
nz	Number of event outputs
nztrue	Number of event outputs in the unaugmented system
o2mode	Flag indicating whether for *amici::Solver::sensi_* == *amici::SensitivityOrder::second* directional or full second order derivative will be computed
pythonGenerated	Flag indicating Matlab- or Python-based model generation
ubw	Upper bandwidth of the Jacobian

Methods

`__init__`(**args*, ***kwargs*)

Overload 1: Default ctor

Overload 2:

Constructor with model dimensions

Parameters

- **nx_rdata** (*int*) – Number of state variables
- **nxtrue_rdata** (*int*) – Number of state variables of the non-augmented model
- **nx_solver** (*int*) – Number of state variables with conservation laws applied
- **nxtrue_solver** (*int*) – Number of state variables of the non-augmented model with conservation laws applied
- **nx_solver_reinit** (*int*) – Number of state variables with conservation laws subject to reinitialization
- **np** (*int*) – Number of parameters
- **nk** (*int*) – Number of constants
- **ny** (*int*) – Number of observables
- **nytrue** (*int*) – Number of observables of the non-augmented model
- **nz** (*int*) – Number of event observables
- **nztrue** (*int*) – Number of event observables of the non-augmented model
- **ne** (*int*) – Number of events
- **nJ** (*int*) – Number of objective functions
- **nw** (*int*) – Number of repeating elements
- **ndwdx** (*int*) – Number of nonzero elements in the *x* derivative of the repeating elements
- **ndwdp** (*int*) – Number of nonzero elements in the *p* derivative of the repeating elements
- **ndwdw** (*int*) – Number of nonzero elements in the *w* derivative of the repeating elements
- **ndxdotdw** (*int*) – Number of nonzero elements in the w derivative of $xdot$
- **ndJydy** (IntVector) – Number of nonzero elements in the y derivative of dJy (shape *nytrue*)
- **ndxrdatadxsolver** (*int*) – Number of nonzero elements in the x derivative of x_rdata
- **ndxrdatadtcl** (*int*) – Number of nonzero elements in the tcl derivative of x_rdata
- **ndtotal_cldx_rdata** (*int*) – Number of nonzero elements in the x_rdata derivative of $total_cl$
- **nnz** (*int*) – Number of nonzero elements in Jacobian
- **ubw** (*int*) – Upper matrix bandwidth in the Jacobian

- **lbw** (*int*) – Lower matrix bandwidth in the Jacobian

clone() → Iterable[*amici.amici.Model*]

Clone this instance.

Return type *Model*

Returns The clone

fdsigmaydy(*dsigmaydy: Iterable[float], t: float, p: Iterable[float], k: Iterable[float], y: Iterable[float]*) → None

Model-specific implementation of fsigmay

Parameters

- **dsigmaydy** (typing.Iterable[float]) – partial derivative of standard deviation of measurements w.r.t. model outputs
- **t** (float) – current time
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **y** (typing.Iterable[float]) – model output at timepoint t

Return type None

fdtotal_cldp(*dtotal_cldp: Iterable[float], x_rdata: Iterable[float], p: Iterable[float], k: Iterable[float], ip: int*) → None

Compute dtotal_cl / dp

Parameters

- **dtotal_cldp** (typing.Iterable[float]) – dtotal_cl / dp
- **x_rdata** (typing.Iterable[float]) – State variables with conservation laws applied
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **ip** (int) – Sensitivity index

Return type None

fdtotal_cldx_rdata(*dtotal_cldx_rdata: Iterable[float], x_rdata: Iterable[float], p: Iterable[float], k: Iterable[float], tcl: Iterable[float]*) → None

Compute dtotal_cl / dx_rdata

Parameters

- **dtotal_cldx_rdata** (typing.Iterable[float]) – dtotal_cl / dx_rdata
- **x_rdata** (typing.Iterable[float]) – State variables with conservation laws applied
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **tcl** (typing.Iterable[float]) – Total abundances for conservation laws

Return type None

fdx_rdatadp(*dx_rdatadp: Iterable[float], x: Iterable[float], tcl: Iterable[float], p: Iterable[float], k: Iterable[float], ip: int*) → None

Compute dx_rdata / dp

Parameters

- **dx_rdatadp** (typing.Iterable[float]) – dx_rdata / dp
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **x** (typing.Iterable[float]) – State variables with conservation laws applied
- **tcl** (typing.Iterable[float]) – Total abundances for conservation laws
- **ip** (int) – Sensitivity index

Return type None

fdx_rdatadtcl(*dx_rdatadtcl: Iterable[float], x: Iterable[float], tcl: Iterable[float], p: Iterable[float], k: Iterable[float]*) → None

Compute dx_rdata / dtcl

Parameters

- **dx_rdatadtcl** (typing.Iterable[float]) – dx_rdata / dtcl
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **x** (typing.Iterable[float]) – State variables with conservation laws applied
- **tcl** (typing.Iterable[float]) – Total abundances for conservation laws

Return type None

fdx_rdatadx_solver(*dx_rdatadx_solver: Iterable[float], x: Iterable[float], tcl: Iterable[float], p: Iterable[float], k: Iterable[float]*) → None

Compute dx_rdata / dx_solver

Parameters

- **dx_rdatadx_solver** (typing.Iterable[float]) – dx_rdata / dx_solver
- **p** (typing.Iterable[float]) – parameter vector
- **k** (typing.Iterable[float]) – constant vector
- **x** (typing.Iterable[float]) – State variables with conservation laws applied
- **tcl** (typing.Iterable[float]) – Total abundances for conservation laws

Return type None

getAddSigmaResiduals() → bool

Checks whether residuals should be added to account for parameter dependent sigma.

Return type boolean

Returns sigma_res

getAlwaysCheckFinite() → bool

Get setting of whether the result of every call to *Model::f** should be checked for finiteness.

Return type boolean

Returns that

getAmiciCommit() → str

Returns the AMICI commit that was used to generate the model

Return type str

Returns AMICI commit string

getAmiciVersion() → str

Returns the AMICI version that was used to generate the model

Return type str

Returns AMICI version string

getExpressionIds() → *amici.amici.StringVector*

Get IDs of the expression.

Return type *StringVector*

Returns Expression IDs

getExpressionNames() → *amici.amici.StringVector*

Get names of the expressions.

Return type *StringVector*

Returns Expression names

getFixedParameterById(*par_id: str*) → float

Get value of fixed parameter with the specified ID.

Parameters **par_id** (str) – Parameter ID

Return type float

Returns Parameter value

getFixedParameterByName(*par_name: str*) → float

Get value of fixed parameter with the specified name.

If multiple parameters have the same name, the first parameter with matching name is returned.

Parameters **par_name** (str) – Parameter name

Return type float

Returns Parameter value

getFixedParameterIds() → *amici.amici.StringVector*

Get IDs of the fixed model parameters.

Return type *StringVector*

Returns Fixed parameter IDs

getFixedParameterNames() → *amici.amici.StringVector*

Get names of the fixed model parameters.

Return type *StringVector*

Returns Fixed parameter names

getFixedParameters() → *amici.amici.DoubleVector*

Get values of fixed parameters.

Return type *DoubleVector*

Returns Vector of fixed parameters with same ordering as in Model::getFixedParameterIds

getInitialStateSensitivities() → *amici.amici.DoubleVector*

Get the initial states sensitivities.

Return type *DoubleVector*

Returns vector of initial state sensitivities

getInitialStates() → *amici.amici.DoubleVector*

Get the initial states.

Return type *DoubleVector*

Returns Initial state vector

getMinimumSigmaResiduals() → float

Gets the specified estimated lower boundary for sigma_y.

Return type float

Returns lower boundary

getName() → str

Get the model name.

Return type str

Returns Model name

getObservableIds() → *amici.amici.StringVector*

Get IDs of the observables.

Return type *StringVector*

Returns Observable IDs

getObservableNames() → *amici.amici.StringVector*

Get names of the observables.

Return type *StringVector*

Returns Observable names

getObservableScaling(*iy: int*) → *amici.amici.ObservableScaling*

Get scaling type for observable

Parameters **iy** (`int`) – observable index

Return type int

Returns scaling type

getParameterById(*par_id: str*) → float

Get value of first model parameter with the specified ID.

Parameters **par_id** (str) – Parameter ID

Return type float

Returns Parameter value

getParameterByName(*par_name: str*) → float

Get value of first model parameter with the specified name.

Parameters **par_name** (str) – Parameter name

Return type float

Returns Parameter value

getParameterIds() → *amici.amici.StringVector*

Get IDs of the model parameters.

Return type *StringVector*

Returns Parameter IDs

getParameterList() → *amici.amici.IntVector*

Get the list of parameters for which sensitivities are computed.

Return type *IntVector*

Returns List of parameter indices

getParameterNames() → *amici.amici.StringVector*

Get names of the model parameters.

Return type *StringVector*

Returns The parameter names

getParameterScale() → *amici.amici.ParameterScalingVector*

Get parameter scale for each parameter.

Return type *ParameterScalingVector*

Returns Vector of parameter scales

getParameters() → *amici.amici.DoubleVector*

Get parameter vector.

Return type *DoubleVector*

Returns The user-set parameters (see also *Model::getUnscaledParameters*)

getReinitializationStateIdxs() → *amici.amici.IntVector*

Return indices of states to be reinitialized based on provided constants / fixed parameters

Return type *IntVector*

Returns Those indices.

getReinitializeFixedParameterInitialStates() → bool

Get whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation.

Return type boolean

Returns flag *true* / *false*

`getSolver()` → *amici.amici.Solver*

Retrieves the solver object

Return type *Solver*

Returns The Solver instance

`getStateIds()` → *amici.amici.StringVector*

Get IDs of the model states.

Return type *StringVector*

Returns State IDs

`getStateIdsSolver()` → *amici.amici.StringVector*

Get IDs of the solver states.

Return type *StringVector*

Returns State IDs

`getStateIsNonNegative()` → *amici.amici.BoolVector*

Get flags indicating whether states should be treated as non-negative.

Return type *BoolVector*

Returns Vector of flags

`getStateNames()` → *amici.amici.StringVector*

Get names of the model states.

Return type *StringVector*

Returns State names

`getStateNamesSolver()` → *amici.amici.StringVector*

Get names of the solver states.

Return type *StringVector*

Returns State names

`getSteadyStateSensitivityMode()` → *amici.amici.SteadyStateSensitivityMode*

Gets the mode how sensitivities are computed in the steadystate simulation.

Return type *SteadyStateSensitivityMode*

Returns Mode

`getTimepoint(`*it: int*`)` → float

Get simulation timepoint for time index *it*.

Parameters **it** (`int`) – Time index

Return type float

Returns Timepoint

`getTimepoints()` → *amici.amici.DoubleVector*

Get the timepoint vector.

Return type *DoubleVector*

Returns Timepoint vector

getUnscaledParameters() → *amici.amici.DoubleVector*

Get parameters with transformation according to parameter scale applied.

Return type *DoubleVector*

Returns Unscaled parameters

hasCustomInitialStateSensitivities() → bool

Return whether custom initial state sensitivities have been set.

Return type boolean

Returns *true* if has custom initial state sensitivities, otherwise *false*.

hasCustomInitialStates() → bool

Return whether custom initial states have been set.

Return type boolean

Returns *true* if has custom initial states, otherwise *false*

hasExpressionIds() → bool

Report whether the model has expression IDs set.

Return type boolean

Returns Boolean indicating whether expression ids were set. Also returns *true* if the number of corresponding variables is just zero.

hasExpressionNames() → bool

Report whether the model has expression names set.

Return type boolean

Returns Boolean indicating whether expression names were set. Also returns *true* if the number of corresponding variables is just zero.

hasFixedParameterIds() → bool

Report whether the model has fixed parameter IDs set.

Return type boolean

Returns Boolean indicating whether fixed parameter IDs were set. Also returns *true* if the number of corresponding variables is just zero.

hasFixedParameterNames() → bool

Report whether the model has fixed parameter names set.

Return type boolean

Returns Boolean indicating whether fixed parameter names were set. Also returns *true* if the number of corresponding variables is just zero.

hasObservableIds() → bool

Report whether the model has observable IDs set.

Return type boolean

Returns Boolean indicating whether observable ids were set. Also returns *true* if the number of corresponding variables is just zero.

hasObservableNames() → bool

Report whether the model has observable names set.

Return type boolean

Returns Boolean indicating whether observable names were set. Also returns *true* if the number of corresponding variables is just zero.

hasParameterIds() → bool

Report whether the model has parameter IDs set.

Return type boolean

Returns Boolean indicating whether parameter IDs were set. Also returns *true* if the number of corresponding variables is just zero.

hasParameterNames() → bool

Report whether the model has parameter names set.

Return type boolean

Returns Boolean indicating whether parameter names were set. Also returns *true* if the number of corresponding variables is just zero.

hasQuadraticLLH() → bool

Checks whether the defined noise model is gaussian, i.e., the nllh is quadratic

Return type boolean

Returns boolean flag

hasStateIds() → bool

Report whether the model has state IDs set.

Return type boolean

Returns Boolean indicating whether state IDs were set. Also returns *true* if the number of corresponding variables is just zero.

hasStateNames() → bool

Report whether the model has state names set.

Return type boolean

Returns Boolean indicating whether state names were set. Also returns *true* if the number of corresponding variables is just zero.

isFixedParameterStateReinitializationAllowed() → bool

Function indicating whether reinitialization of states depending on fixed parameters is permissible

Return type boolean

Returns flag indicating whether reinitialization of states depending on fixed parameters is permissible

k() → Iterable[float]

Get fixed parameters.

Return type float

Returns Pointer to constants array

nMaxEvent() → int

Get maximum number of events that may occur for each type.

Return type int

Returns Maximum number of events that may occur for each type

ncl() → int

Get number of conservation laws.

Return type int

Returns Number of conservation laws (i.e., difference between *nx_rdata* and *nx_solver*).

nk() → int

Get number of constants

Return type int

Returns Length of constant vector

np() → int

Get total number of model parameters.

Return type int

Returns Length of parameter vector

nplist() → int

Get number of parameters wrt to which sensitivities are computed.

Return type int

Returns Length of sensitivity index vector

nt() → int

Get number of timepoints.

Return type int

Returns Number of timepoints

nx_reinit() → int

Get number of solver states subject to reinitialization.

Return type int

Returns Model member *nx_solver_reinit*

plist(*pos: int*) → int

Get entry in parameter list by index.

Parameters **pos** (int) – Index in sensitivity parameter list

Return type int

Returns Index in parameter list

requireSensitivitiesForAllParameters() → None

Require computation of sensitivities for all parameters p [0..np[in natural order.

NOTE: Resets initial state sensitivities.

Return type None

setAddSigmaResiduals(*sigma_res: bool*) → None

Specifies whether residuals should be added to account for parameter dependent sigma.

If set to true, additional residuals of the form $\sqrt{\log(\sigma) + C}$ will be added. This enables least-squares optimization for variables with Gaussian noise assumption and parameter dependent standard deviation sigma. The constant C can be set via `setMinimumSigmaResiduals()`.

Parameters **sigma_res** (bool) – if true, additional residuals are added

Return type None

setAllStatesNonNegative() → None

Set flags indicating that all states should be treated as non-negative.

Return type None

setAlwaysCheckFinite(*alwaysCheck: bool*) → None

Set whether the result of every call to *Model::f** should be checked for finiteness.

Parameters **alwaysCheck** (bool) –

Return type None

setFixedParameterById(*par_id: str*, *value: float*) → None

Set value of first fixed parameter with the specified ID.

Parameters

- **par_id** (str) – Fixed parameter id
- **value** (float) – Fixed parameter value

Return type None

setFixedParameterByName(*par_name: str*, *value: float*) → None

Set value of first fixed parameter with the specified name.

Parameters

- **par_name** (str) – Fixed parameter ID
- **value** (float) – Fixed parameter value

Return type None

setFixedParameters(*k:* amici.amici.DoubleVector) → None

Set values for constants.

Parameters **k** (*amici.amici.DoubleVector*) – Vector of fixed parameters

Return type None

setFixedParametersByIdRegex(*par_id_regex: str*, *value: float*) → int

Set values of all fixed parameters with the ID matching the specified regex.

Parameters

- **par_id_regex** (str) – Fixed parameter name regex
- **value** (float) – Fixed parameter value

Return type int

Returns Number of fixed parameter IDs that matched the regex

setFixedParametersByNameRegex(*par_name_regex: str*, *value: float*) → int

Set value of all fixed parameters with name matching the specified regex.

Parameters

- **par_name_regex** (str) – Fixed parameter name regex
- **value** (float) – Fixed parameter value

Return type int

Returns Number of fixed parameter names that matched the regex

setInitialStateSensitivities(*sx0: amici.amici.DoubleVector*) → None

Set the initial state sensitivities.

Parameters **sx0** (*amici.amici.DoubleVector*) – vector of initial state sensitivities with chainrule applied. This could be a slice of ReturnData::sx or ReturnData::sx0

Return type None

setInitialStates(*x0: amici.amici.DoubleVector*) → None

Set the initial states.

Parameters **x0** (*amici.amici.DoubleVector*) – Initial state vector

Return type None

setMinimumSigmaResiduals(*min_sigma: float*) → None

Sets the estimated lower boundary for sigma_y. When *setAddSigmaResiduals()* is activated, this lower boundary must ensure that log(sigma) + min_sigma > 0.

Parameters **min_sigma** (float) – lower boundary

Return type None

setNMaxEvent(*nmaxevent: int*) → None

Set maximum number of events that may occur for each type.

Parameters **nmaxevent** (int) – Maximum number of events that may occur for each type

Return type None

setParameterById(**args*) → None

Overload 1:

Set model parameters according to the parameter IDs and mapped values.

Parameters

- **p** (StringDoubleMap) – Map of parameters IDs and values
- **ignoreErrors** (*boolean*) – Ignore errors such as parameter IDs in p which are not model parameters

Overload 2:

Set value of first model parameter with the specified ID.

Parameters

- **par_id** (*str*) – Parameter ID
- **value** (*float*) – Parameter value

Return type None

setParameterByName(**args*) → None

Overload 1:

Set value of first model parameter with the specified name.

Parameters

- **par_name** (*str*) – Parameter name
- **value** (*float*) – Parameter value

Overload 2:

Set model parameters according to the parameter name and mapped values.

Parameters

- **p** (StringDoubleMap) – Map of parameters names and values
- **ignoreErrors** (*boolean*) – Ignore errors such as parameter names in p which are not model parameters

Overload 3:

Set model parameters according to the parameter name and mapped values.

Parameters

- **p** (StringDoubleMap) – Map of parameters names and values
- **ignoreErrors** – Ignore errors such as parameter names in p which are not model parameters

Return type None

setParameterList(*plist:* amici.amici.IntVector) → None

Set the list of parameters for which sensitivities are to be computed.

NOTE: Resets initial state sensitivities.

Parameters **plist** (*amici.amici.IntVector*) – List of parameter indices

Return type None

setParameterScale(**args*) → None

Return type None

setParameters(*p: amici.amici.DoubleVector*) → None

Set the parameter vector.

Parameters **p** (*amici.amici.DoubleVector*) – Vector of parameters

Return type None

setParametersByIdRegex(*par_id_regex: str*, *value: float*) → int

Set all values of model parameters with IDs matching the specified regular expression.

Parameters

- **par_id_regex** (str) – Parameter ID regex
- **value** (float) – Parameter value

Return type int

Returns Number of parameter IDs that matched the regex

setParametersByNameRegex(*par_name_regex: str*, *value: float*) → int

Set all values of all model parameters with names matching the specified regex.

Parameters

- **par_name_regex** (str) – Parameter name regex
- **value** (float) – Parameter value

Return type int

Returns Number of fixed parameter names that matched the regex

setReinitializationStateIdxs(*idxs: amici.amici.IntVector*) → None

Set indices of states to be reinitialized based on provided constants / fixed parameters

Parameters **idxs** (*amici.amici.IntVector*) – Array of state indices

Return type None

setReinitializeFixedParameterInitialStates(*flag: bool*) → None

Set whether initial states depending on fixed parameters are to be reinitialized after preequilibration and presimulation.

Parameters **flag** (bool) – Fixed parameters reinitialized?

Return type None

setStateIsNonNegative(*stateIsNonNegative: amici.amici.BoolVector*) → None

Set flags indicating whether states should be treated as non-negative.

Parameters **stateIsNonNegative** (*amici.amici.BoolVector*) – Vector of flags

Return type None

setSteadyStateSensitivityMode(*mode: amici.amici.SteadyStateSensitivityMode*) → None

Set the mode how sensitivities are computed in the steadystate simulation.

Parameters **mode** (*amici.amici.SteadyStateSensitivityMode*) – Steadystate sensitivity mode

Return type None

setT0(*t0: float*) → None

Set simulation start time.

Parameters **t0** (float) – Simulation start time

Return type None

setTimepoints(*ts:* amici.amici.DoubleVector) → None

Set the timepoint vector.

Parameters **ts** (*amici.amici.DoubleVector*) – New timepoint vector

Return type None

setUnscaledInitialStateSensitivities(*sx0:* amici.amici.DoubleVector) → None

Set the initial state sensitivities.

Parameters **sx0** (*amici.amici.DoubleVector*) – Vector of initial state sensitivities without chainrule applied. This could be the readin from a *model.sx0data* saved to HDF5.

Return type None

t0() → float

Get simulation start time.

Return type float

Returns Simulation start time

amici.amici.ModelDimensions

class amici.amici.**ModelDimensions**(**args*)

Container for model dimensions.

Holds number of states, observables, etc.

__init__(**args*)

Overload 1: Default ctor

Overload 2:

Constructor with model dimensions

Parameters

- **nx_rdata** (*int*) – Number of state variables
- **nxtrue_rdata** (*int*) – Number of state variables of the non-augmented model
- **nx_solver** (*int*) – Number of state variables with conservation laws applied
- **nxtrue_solver** (*int*) – Number of state variables of the non-augmented model with conservation laws applied
- **nx_solver_reinit** (*int*) – Number of state variables with conservation laws subject to reinitialization
- **np** (*int*) – Number of parameters

- **nk** (*int*) – Number of constants
- **ny** (*int*) – Number of observables
- **nytrue** (*int*) – Number of observables of the non-augmented model
- **nz** (*int*) – Number of event observables
- **nztrue** (*int*) – Number of event observables of the non-augmented model
- **ne** (*int*) – Number of events
- **nJ** (*int*) – Number of objective functions
- **nw** (*int*) – Number of repeating elements
- **ndwdx** (*int*) – Number of nonzero elements in the *x* derivative of the repeating elements
- **ndwdp** (*int*) – Number of nonzero elements in the *p* derivative of the repeating elements
- **ndwdw** (*int*) – Number of nonzero elements in the *w* derivative of the repeating elements
- **ndxdotdw** (*int*) – Number of nonzero elements in the w derivative of $xdot$
- **ndJydy** (IntVector) – Number of nonzero elements in the y derivative of dJy (shape *nytrue*)
- **ndxrdatadxsolver** (*int*) – Number of nonzero elements in the x derivative of x_rdata
- **ndxrdatadtcl** (*int*) – Number of nonzero elements in the tcl derivative of x_rdata
- **ndtotal_cldx_rdata** (*int*) – Number of nonzero elements in the x_rdata derivative of $total_cl$
- **nnz** (*int*) – Number of nonzero elements in Jacobian
- **ubw** (*int*) – Upper matrix bandwidth in the Jacobian
- **lbw** (*int*) – Lower matrix bandwidth in the Jacobian

Methods Summary

`__init__`(*args)	*Overload 1:*

Attributes

lbw	Lower bandwidth of the Jacobian
nJ	Dimension of the augmented objective function for 2nd order ASA
ndJydy	Number of nonzero elements in the y derivative of dJy (dimension *nytrue*)
ndtotal_cldx_rdata	Number of nonzero elements in the $x_r data$ derivative of $total_c l$
ndwdp	Number of nonzero elements in the p derivative of the repeating elements
ndwdw	Number of nonzero elements in the w derivative of the repeating elements
ndwdx	Number of nonzero elements in the *x* derivative of the repeating elements
ndxdotdw	Number of nonzero elements in the w derivative of $xdot$
ndxrdatadtcl	Number of nonzero elements in the tcl derivative of $x_r data$
ndxrdatadxsolver	Number of nonzero elements in the x derivative of $x_r data$
ne	Number of events
nk	Number of constants
nnz	Number of nonzero entries in Jacobian
np	Number of parameters
nw	Number of common expressions
nx_rdata	Number of states
nx_solver	Number of states with conservation laws applied
nx_solver_reinit	Number of solver states subject to reinitialization
nxtrue_rdata	Number of states in the unaugmented system
nxtrue_solver	Number of states in the unaugmented system with conservation laws applied
ny	Number of observables
nytrue	Number of observables in the unaugmented system
nz	Number of event outputs
nztrue	Number of event outputs in the unaugmented system
ubw	Upper bandwidth of the Jacobian

Methods

__init__(**args*)

Overload 1: Default ctor

Overload 2:

Constructor with model dimensions

Parameters

- **nx_rdata** (*int*) – Number of state variables
- **nxtrue_rdata** (*int*) – Number of state variables of the non-augmented model
- **nx_solver** (*int*) – Number of state variables with conservation laws applied
- **nxtrue_solver** (*int*) – Number of state variables of the non-augmented model with conservation laws applied
- **nx_solver_reinit** (*int*) – Number of state variables with conservation laws subject to reinitialization
- **np** (*int*) – Number of parameters
- **nk** (*int*) – Number of constants
- **ny** (*int*) – Number of observables
- **nytrue** (*int*) – Number of observables of the non-augmented model
- **nz** (*int*) – Number of event observables
- **nztrue** (*int*) – Number of event observables of the non-augmented model
- **ne** (*int*) – Number of events
- **nJ** (*int*) – Number of objective functions
- **nw** (*int*) – Number of repeating elements
- **ndwdx** (*int*) – Number of nonzero elements in the *x* derivative of the repeating elements
- **ndwdp** (*int*) – Number of nonzero elements in the *p* derivative of the repeating elements
- **ndwdw** (*int*) – Number of nonzero elements in the *w* derivative of the repeating elements
- **ndxdotdw** (*int*) – Number of nonzero elements in the w derivative of $xdot$
- **ndJydy** (IntVector) – Number of nonzero elements in the y derivative of dJy (shape *nytrue*)
- **ndxrdatadxsolver** (*int*) – Number of nonzero elements in the x derivative of $x_r data$
- **ndxrdatadtcl** (*int*) – Number of nonzero elements in the tcl derivative of $x_r data$
- **ndtotal_cldx_rdata** (*int*) – Number of nonzero elements in the $x_r data$ derivative of $total_c l$
- **nnz** (*int*) – Number of nonzero elements in Jacobian
- **ubw** (*int*) – Upper matrix bandwidth in the Jacobian
- **lbw** (*int*) – Lower matrix bandwidth in the Jacobian

amici.amici.ModelPtr

`class amici.amici.ModelPtr(*args)`

Swig-Generated class that implements smart pointers to Model as objects.

Attributes

idlist	Flag array for DAE equations
lbw	Lower bandwidth of the Jacobian
logger	Logger
nJ	Dimension of the augmented objective function for 2nd order ASA
ndJydy	Number of nonzero elements in the y derivative of dJy (dimension *nytrue*)
ndtotal_cldx_rdata	Number of nonzero elements in the $x_r data$ derivative of $total_c l$
ndwdp	Number of nonzero elements in the *p* derivative of the repeating elements
ndwdw	Number of nonzero elements in the *w* derivative of the repeating elements
ndwdx	Number of nonzero elements in the *x* derivative of the repeating elements
ndxdotdw	Number of nonzero elements in the w derivative of $xdot$
ndxrdatadtcl	Number of nonzero elements in the tcl derivative of $x_r data$
ndxrdatadxsolver	Number of nonzero elements in the x derivative of $x_r data$
ne	Number of events
nnz	Number of nonzero entries in Jacobian
nw	Number of common expressions
nx_rdata	Number of states
nx_solver	Number of states with conservation laws applied
nx_solver_reinit	Number of solver states subject to reinitialization
nxtrue_rdata	Number of states in the unaugmented system
nxtrue_solver	Number of states in the unaugmented system with conservation laws applied
ny	Number of observables
nytrue	Number of observables in the unaugmented system
nz	Number of event outputs
nztrue	Number of event outputs in the unaugmented system
o2mode	Flag indicating whether for *amici::Solver::sensi_* == *amici::SensitivityOrder::second* directional or full second order derivative will be computed
pythonGenerated	Flag indicating Matlab- or Python-based model generation
ubw	Upper bandwidth of the Jacobian

amici.amici.NewtonDampingFactorMode

class amici.amici.**NewtonDampingFactorMode**(*value*)

An enumeration.

Attributes

off
on

amici.amici.NonlinearSolverIteration

class amici.amici.**NonlinearSolverIteration**(*value*)

An enumeration.

Attributes

functional
fixedpoint
newton

amici.amici.ObservableScaling

class amici.amici.**ObservableScaling**(*value*)

An enumeration.

Attributes

lin
log
log10

amici.amici.ParameterScaling

class amici.amici.**ParameterScaling**(*value*)

An enumeration.

Attributes

none
ln
log10

amici.amici.ParameterScalingVector

class amici.amici.**ParameterScalingVector**(**args*)

amici.amici.RDataReporting

class amici.amici.**RDataReporting**(*value*)

An enumeration.

Attributes

full
residuals
likelihood

amici.amici.ReturnData

class amici.amici.**ReturnData**(**args*)

Stores all data to be returned by `amici.amici.runAmiciSimulation()`.

NOTE: multi-dimensional arrays are stored in row-major order (C-style)

__init__(**args*)

Overload 1:

Default constructor

Overload 2:

Constructor

Parameters

- **ts** (DoubleVector) – see amici::SimulationParameters::ts
- **model_dimensions** (*ModelDimensions*) – Model dimensions
- **nplist** (*int*) – see amici::ModelDimensions::nplist
- **nmaxevent** (*int*) – see amici::ModelDimensions::nmaxevent
- **nt** (*int*) – see amici::ModelDimensions::nt
- **newton_maxsteps** (*int*) – see amici::Solver::newton_maxsteps
- **pscale** (ParameterScalingVector) – see amici::SimulationParameters::pscale
- **o2mode** (*int*) – see amici::SimulationParameters::o2mode
- **sensi** (SensitivityOrder) – see amici::Solver::sensi
- **sensi_meth** (SensitivityMethod) – see amici::Solver::sensi_meth
- **rdrm** (RDataReporting) – see amici::Solver::rdata_reporting
- **quadratic_llh** (*boolean*) – whether model defines a quadratic nllh and computing res, sres and FIM makes sense
- **sigma_res** (*boolean*) – indicates whether additional residuals are to be added for each sigma
- **sigma_offset** (*float*) – offset to ensure real-valuedness of sigma residuals

Overload 3:

constructor that uses information from model and solver to appropriately initialize fields

Parameters

- **solver** (*Solver*) – solver instance
- **model** (*Model*) – model instance

Methods Summary

__init__(*args)	*Overload 1:*

Attributes

FIM	fisher information matrix (shape *nplist* x *nplist*, row-major)
J	Jacobian of differential equation right hand side (shape *nx* x *nx*, row-major)
chi2	χ^2 value
cpu_time	computation time of forward solve [ms]
cpu_timeB	computation time of backward solve [ms]
cpu_time_total	total CPU time from entering runAmiciSimulation until exiting [ms]
id	Arbitrary (not necessarily unique) identifier.
lbw	Lower bandwidth of the Jacobian
llh	log-likelihood value
messages	log messages
nJ	Dimension of the augmented objective function for 2nd order ASA
ndJydy	Number of nonzero elements in the y derivative of dJy (dimension *nytrue*)
ndtotal_cldx_rdata	Number of nonzero elements in the x_rdata derivative of $total_cl$
ndwdp	Number of nonzero elements in the p derivative of the repeating elements
ndwdw	Number of nonzero elements in the *w* derivative of the repeating elements
ndwdx	Number of nonzero elements in the *x* derivative of the repeating elements
ndxdotdw	Number of nonzero elements in the w derivative of $xdot$
ndxrdatadtcl	Number of nonzero elements in the tcl derivative of x_rdata
ndxrdatadxsolver	Number of nonzero elements in the x derivative of x_rdata
ne	Number of events
newton_maxsteps	maximal number of newton iterations for steady state calculation
nk	Number of constants
nmaxevent	maximal number of occurring events (for every event type)
nnz	Number of nonzero entries in Jacobian
np	Number of parameters
nplist	number of parameter for which sensitivities were requested
nt	number of considered timepoints
numerrtestfails	number of error test failures forward problem (shape *nt*)
numerrtestfailsB	number of error test failures backward problem (shape *nt*)
numnonlinsolvconvfails	number of linear solver convergence failures forward problem (shape *nt*)

continues on next page

Table 4 – continued from previous page

numnonlinsolvconvfailsB	number of linear solver convergence failures backward problem (shape *nt*)
numrhsevals	number of right hand side evaluations forward problem (shape *nt*)
numrhsevalsB	number of right hand side evaluations backward problem (shape *nt*)
numsteps	number of integration steps forward problem (shape *nt*)
numstepsB	number of integration steps backward problem (shape *nt*)
nw	Number of common expressions
nx	number of states (alias *nx_rdata*, kept for backward compatibility)
nx_rdata	Number of states
nx_solver	Number of states with conservation laws applied
nx_solver_reinit	Number of solver states subject to reinitialization
nxtrue	number of states in the unaugmented system (alias nxtrue_rdata, kept for backward compatibility)
nxtrue_rdata	Number of states in the unaugmented system
nxtrue_solver	Number of states in the unaugmented system with conservation laws applied
ny	Number of observables
nytrue	Number of observables in the unaugmented system
nz	Number of event outputs
nztrue	Number of event outputs in the unaugmented system
o2mode	flag indicating whether second-order sensitivities were requested
order	employed order forward problem (shape *nt*)
posteq_cpu_time	computation time of the steady state solver [ms] (postequilibration)
posteq_cpu_timeB	computation time of the steady state solver of the backward problem [ms] (postequilibration)
posteq_numsteps	number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (shape *3*) (postequilibration)
posteq_numstepsB	number of simulation steps for adjoint steady state problem (postequilibration) [== 0 if analytical solution worked, > 0 otherwise]
posteq_status	flags indicating success of steady state solver (postequilibration)
posteq_t	time when steadystate was reached via simulation (postequilibration)
posteq_wrms	weighted root-mean-square of the rhs when steadystate was reached (postequilibration)
preeq_cpu_time	computation time of the steady state solver [ms] (preequilibration)
preeq_cpu_timeB	computation time of the steady state solver of the backward problem [ms] (preequilibration)
preeq_numsteps	number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3)

continues on next page

Table 4 – continued from previous page

preeq_numstepsB	number of simulation steps for adjoint steady state problem (preequilibration) [== 0 if analytical solution worked, > 0 otherwise]
preeq_status	flags indicating success of steady state solver (preequilibration)
preeq_t	time when steadystate was reached via simulation (preequilibration)
preeq_wrms	weighted root-mean-square of the rhs when steadystate was reached (preequilibration)
pscale	scaling of parameterization
rdata_reporting	reporting mode
res	observable (shape *nt*ny*, row-major)
rz	event trigger output (shape *nmaxevent* x *nz*, row-major)
s2llh	second-order parameter derivative of log-likelihood (shape *nJ-1* x *nplist*, row-major)
s2rz	second-order parameter derivative of event trigger output (shape *nmaxevent* x *nztrue* x *nplist* x *nplist*, row-major)
sensi	sensitivity order
sensi_meth	sensitivity method
sigma_res	boolean indicating whether residuals for standard deviations have been added
sigmay	observable standard deviation (shape *nt* x *ny*, row-major)
sigmaz	event output sigma standard deviation (shape *nmaxevent* x *nz*, row-major)
sllh	parameter derivative of log-likelihood (shape *nplist*)
sres	parameter derivative of residual (shape *nt*ny* x *nplist*, row-major)
srz	parameter derivative of event trigger output (shape *nmaxevent* x *nplist* x *nz*, row-major)
ssigmay	parameter derivative of observable standard deviation (shape *nt* x *nplist* x *ny*, row-major)
ssigmaz	parameter derivative of event output standard deviation (shape *nmaxevent* x *nplist* x *nz*, row-major)
status	Simulation status code.
sx	parameter derivative of state (shape *nt* x *nplist* x *nx*, row-major)
sx0	initial sensitivities (shape *nplist* x *nx*, row-major)
sx_ss	preequilibration sensitivities found by Newton solver (shape *nplist* x *nx*, row-major)
sy	parameter derivative of observable (shape *nt* x *nplist* x *ny*, row-major)
sz	parameter derivative of event output (shape *nmaxevent* x *nplist* x *nz*, row-major)
ts	timepoints (shape *nt*)
ubw	Upper bandwidth of the Jacobian
w	w data from the model (recurring terms in xdot, for imported SBML models from python, this contains the flux vector) (shape *nt* x *nw*, row major)

continues on next page

Table 4 – continued from previous page

x	state (shape *nt* x *nx*, row-major)
x0	initial state (shape *nx*)
x_ss	preequilibration steady state found by Newton solver (shape *nx*)
xdot	time derivative (shape *nx*)
y	observable (shape *nt* x *ny*, row-major)
z	event output (shape *nmaxevent* x *nz*, row-major)

Methods

__init__(**args*)

Overload 1:

Default constructor

Overload 2:

Constructor

Parameters

- **ts** (DoubleVector) – see amici::SimulationParameters::ts
- **model_dimensions** (*ModelDimensions*) – Model dimensions
- **nplist** (*int*) – see amici::ModelDimensions::nplist
- **nmaxevent** (*int*) – see amici::ModelDimensions::nmaxevent
- **nt** (*int*) – see amici::ModelDimensions::nt
- **newton_maxsteps** (*int*) – see amici::Solver::newton_maxsteps
- **pscale** (ParameterScalingVector) – see amici::SimulationParameters::pscale
- **o2mode** (*int*) – see amici::SimulationParameters::o2mode
- **sensi** (SensitivityOrder) – see amici::Solver::sensi
- **sensi_meth** (SensitivityMethod) – see amici::Solver::sensi_meth
- **rdrm** (RDataReporting) – see amici::Solver::rdata_reporting
- **quadratic_llh** (*boolean*) – whether model defines a quadratic nllh and computing res, sres and FIM makes sense
- **sigma_res** (*boolean*) – indicates whether additional residuals are to be added for each sigma
- **sigma_offset** (*float*) – offset to ensure real-valuedness of sigma residuals

Overload 3:

constructor that uses information from model and solver to appropriately initialize fields

Parameters

- **solver** (*Solver*) – solver instance
- **model** (*Model*) – model instance

amici.amici.ReturnDataPtr

`class amici.amici.ReturnDataPtr(*args)`

Swig-Generated class that implements smart pointers to ReturnData as objects.

Attributes

`FIM`	fisher information matrix (shape *nplist* x *nplist*, row-major)
`J`	Jacobian of differential equation right hand side (shape *nx* x *nx*, row-major)
`chi2`	χ^2 value
`cpu_time`	computation time of forward solve [ms]
`cpu_timeB`	computation time of backward solve [ms]
`cpu_time_total`	total CPU time from entering runAmiciSimulation until exiting [ms]
`id`	Arbitrary (not necessarily unique) identifier.
`lbw`	Lower bandwidth of the Jacobian
`llh`	log-likelihood value
`messages`	log messages
`nJ`	Dimension of the augmented objective function for 2nd order ASA
`ndJydy`	Number of nonzero elements in the y derivative of dJy (dimension *nytrue*)
`ndtotal_cldx_rdata`	Number of nonzero elements in the x_rdata derivative of $total_cl$
`ndwdp`	Number of nonzero elements in the *p* derivative of the repeating elements
`ndwdw`	Number of nonzero elements in the *w* derivative of the repeating elements
`ndwdx`	Number of nonzero elements in the *x* derivative of the repeating elements
`ndxdotdw`	Number of nonzero elements in the w derivative of $xdot$
`ndxrdatadtcl`	Number of nonzero elements in the tcl derivative of x_rdata
`ndxrdatadxsolver`	Number of nonzero elements in the x derivative of x_rdata
`ne`	Number of events
`newton_maxsteps`	maximal number of newton iterations for steady state calculation
`nk`	Number of constants

continues on next page

Table 5 – continued from previous page

nmaxevent	maximal number of occurring events (for every event type)
nnz	Number of nonzero entries in Jacobian
np	Number of parameters
nplist	number of parameter for which sensitivities were requested
nt	number of considered timepoints
numerrtestfails	number of error test failures forward problem (shape *nt*)
numerrtestfailsB	number of error test failures backward problem (shape *nt*)
numnonlinsolvconvfails	number of linear solver convergence failures forward problem (shape *nt*)
numnonlinsolvconvfailsB	number of linear solver convergence failures backward problem (shape *nt*)
numrhsevals	number of right hand side evaluations forward problem (shape *nt*)
numrhsevalsB	number of right hand side evaluations backward problem (shape *nt*)
numsteps	number of integration steps forward problem (shape *nt*)
numstepsB	number of integration steps backward problem (shape *nt*)
nw	Number of common expressions
nx	number of states (alias *nx_rdata*, kept for backward compatibility)
nx_rdata	Number of states
nx_solver	Number of states with conservation laws applied
nx_solver_reinit	Number of solver states subject to reinitialization
nxtrue	number of states in the unaugmented system (alias nxtrue_rdata, kept for backward compatibility)
nxtrue_rdata	Number of states in the unaugmented system
nxtrue_solver	Number of states in the unaugmented system with conservation laws applied
ny	Number of observables
nytrue	Number of observables in the unaugmented system
nz	Number of event outputs
nztrue	Number of event outputs in the unaugmented system
o2mode	flag indicating whether second-order sensitivities were requested
order	employed order forward problem (shape *nt*)
posteq_cpu_time	computation time of the steady state solver [ms] (postequilibration)
posteq_cpu_timeB	computation time of the steady state solver of the backward problem [ms] (postequilibration)
posteq_numsteps	number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (shape *3*) (postequilibration)
posteq_numstepsB	number of simulation steps for adjoint steady state problem (postequilibration) [== 0 if analytical solution worked, > 0 otherwise]

continues on next page

Table 5 – continued from previous page

posteq_status	flags indicating success of steady state solver (postequilibration)
posteq_t	time when steadystate was reached via simulation (postequilibration)
posteq_wrms	weighted root-mean-square of the rhs when steadystate was reached (postequilibration)
preeq_cpu_time	computation time of the steady state solver [ms] (preequilibration)
preeq_cpu_timeB	computation time of the steady state solver of the backward problem [ms] (preequilibration)
preeq_numsteps	number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3)
preeq_numstepsB	number of simulation steps for adjoint steady state problem (preequilibration) [== 0 if analytical solution worked, > 0 otherwise]
preeq_status	flags indicating success of steady state solver (preequilibration)
preeq_t	time when steadystate was reached via simulation (preequilibration)
preeq_wrms	weighted root-mean-square of the rhs when steadystate was reached (preequilibration)
pscale	scaling of parameterization
rdata_reporting	reporting mode
res	observable (shape *nt*ny*, row-major)
rz	event trigger output (shape *nmaxevent* x *nz*, row-major)
s2llh	second-order parameter derivative of log-likelihood (shape *nJ-1* x *nplist*, row-major)
s2rz	second-order parameter derivative of event trigger output (shape *nmaxevent* x *nztrue* x *nplist* x *nplist*, row-major)
sensi	sensitivity order
sensi_meth	sensitivity method
sigma_res	boolean indicating whether residuals for standard deviations have been added
sigmay	observable standard deviation (shape *nt* x *ny*, row-major)
sigmaz	event output sigma standard deviation (shape *nmaxevent* x *nz*, row-major)
sllh	parameter derivative of log-likelihood (shape *nplist*)
sres	parameter derivative of residual (shape *nt*ny* x *nplist*, row-major)
srz	parameter derivative of event trigger output (shape *nmaxevent* x *nplist* x *nz*, row-major)
ssigmay	parameter derivative of observable standard deviation (shape *nt* x *nplist* x *ny*, row-major)
ssigmaz	parameter derivative of event output standard deviation (shape *nmaxevent* x *nplist* x *nz*, row-major)
status	Simulation status code.

continues on next page

Table 5 – continued from previous page

sx	parameter derivative of state (shape *nt* x *nplist* x *nx*, row-major)
sx0	initial sensitivities (shape *nplist* x *nx*, row-major)
sx_ss	preequilibration sensitivities found by Newton solver (shape *nplist* x *nx*, row-major)
sy	parameter derivative of observable (shape *nt* x *nplist* x *ny*, row-major)
sz	parameter derivative of event output (shape *nmaxevent* x *nplist* x *nz*, row-major)
ts	timepoints (shape *nt*)
ubw	Upper bandwidth of the Jacobian
w	w data from the model (recurring terms in xdot, for imported SBML models from python, this contains the flux vector) (shape *nt* x *nw*, row major)
x	state (shape *nt* x *nx*, row-major)
x0	initial state (shape *nx*)
x_ss	preequilibration steady state found by Newton solver (shape *nx*)
xdot	time derivative (shape *nx*)
y	observable (shape *nt* x *ny*, row-major)
z	event output (shape *nmaxevent* x *nz*, row-major)

amici.amici.SecondOrderMode

class amici.amici.**SecondOrderMode**(*value*)

An enumeration.

Attributes

none
full
directional

amici.amici.SensitivityMethod

class amici.amici.**SensitivityMethod**(*value*)

An enumeration.

Attributes

none
forward
adjoint

amici.amici.SensitivityOrder

class amici.amici.**SensitivityOrder**(*value*)

An enumeration.

Attributes

none
first
second

amici.amici.SimulationParameters

class amici.amici.**SimulationParameters**(**args*)

Container for various simulation parameters.

__init__(**args*)

Overload 1:

Constructor

Parameters **timepoints** (DoubleVector) – Timepoints for which simulation results are requested

Overload 2:

Constructor

Parameters

- **fixedParameters** (DoubleVector) – Model constants
- **parameters** (DoubleVector) – Model parameters

Overload 3:

Constructor

Parameters

- **fixedParameters** (DoubleVector) – Model constants
- **parameters** (DoubleVector) – Model parameters
- **plist** (IntVector) – Model parameter indices w.r.t. which sensitivities are to be computed

Overload 4:

Constructor

Parameters

- **timepoints** (DoubleVector) – Timepoints for which simulation results are requested
- **fixedParameters** (DoubleVector) – Model constants
- **parameters** (DoubleVector) – Model parameters

Methods Summary

__init__(*args)	*Overload 1:*
reinitializeAllFixedParameterDependentInitialStates(...)	Set reinitialization of all states based on model constants for all simulation phases.
reinitializeAllFixedParameterDependentInitialStatesForPresimulation(...)	Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).
reinitializeAllFixedParameterDependentInitialStatesForSimulation(...)	Set reinitialization of all states based on model constants for the 'main' simulation (only meaningful if presimulation or preequilibration is performed).

Attributes

`fixedParameters`	Model constants
`fixedParametersPreequilibration`	Model constants for pre-equilibration
`fixedParametersPresimulation`	Model constants for pre-simulation
`parameters`	Model parameters
`plist`	Parameter indices w.r.t.
`pscale`	Parameter scales
`reinitialization_state_idxs_presim`	Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.
`reinitialization_state_idxs_sim`	Indices of states to be reinitialized based on provided constants / fixed parameters.
`reinitializeFixedParameterInitialStates`	Flag indicating whether reinitialization of states depending on fixed parameters is activated
`sx0`	Initial state sensitivities
`t_presim`	Duration of pre-simulation.
`ts_`	Timepoints for which model state/outputs/.
`tstart_`	starting time
`x0`	Initial state

Methods

`__init__`(**args*)

Overload 1:

Constructor

Parameters **`timepoints`** (`DoubleVector`) – Timepoints for which simulation results are requested

Overload 2:

Constructor

Parameters

- **`fixedParameters`** (`DoubleVector`) – Model constants
- **`parameters`** (`DoubleVector`) – Model parameters

Overload 3:

Constructor

Parameters

- **`fixedParameters`** (`DoubleVector`) – Model constants

- **parameters** (DoubleVector) – Model parameters
- **plist** (IntVector) – Model parameter indices w.r.t. which sensitivities are to be computed

Overload 4:

Constructor

Parameters

- **timepoints** (DoubleVector) – Timepoints for which simulation results are requested
- **fixedParameters** (DoubleVector) – Model constants
- **parameters** (DoubleVector) – Model parameters

reinitializeAllFixedParameterDependentInitialStates(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for all simulation phases.

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (int) – Number of states (Model::nx_rdata)

Return type None

reinitializeAllFixedParameterDependentInitialStatesForPresimulation(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (int) – Number of states (Model::nx_rdata)

Return type None

reinitializeAllFixedParameterDependentInitialStatesForSimulation(*nx_rdata: int*) → None

Set reinitialization of all states based on model constants for the 'main' simulation (only meaningful if presimulation or preequilibration is performed).

Convenience function to populate *reinitialization_state_idxs_presim* and *reinitialization_state_idxs_sim*

Parameters **nx_rdata** (int) – Number of states (Model::nx_rdata)

Return type None

amici.amici.SimulationState

class amici.amici.**SimulationState**

implements an exchange format to store and transfer the state of a simulation at a specific timepoint.

__init__()

Methods Summary

__init__()	

Attributes

dx	state variables
state	state of the model that was used for simulation
sx	state variable sensitivity
t	timepoint
x	state variables

Methods

__init__()

amici.amici.Solver

class amici.amici.**Solver**(**args*, ***kwargs*)

The Solver class provides a generic interface to CVODES and IDAS solvers, individual realizations are realized in the CVodeSolver and the IDASolver class. All transient private/protected members (CVODES/IDAS memory, interface variables and status flags) are specified as mutable and not included in serialization or equality checks. No solver setting parameter should be marked mutable.

NOTE: Any changes in data members here must be propagated to copy ctor, equality operator, serialization functions in serialization.h, and amici::hdf5::(read/write)SolverSettings(From/To)HDF5 in hdf5.cpp.

__init__(**args*, ***kwargs*)

Methods Summary

__init__(*args, **kwargs)	
clone()	Clone this instance
computingASA()	check if ASA is being computed
computingFSA()	check if FSA is being computed
getAbsoluteTolerance()	Get the absolute tolerances for the forward problem
getAbsoluteToleranceB()	Returns the absolute tolerances for the backward problem for adjoint sensitivity analysis
getAbsoluteToleranceFSA()	Returns the absolute tolerances for the forward sensitivity problem
getAbsoluteToleranceQuadratures()	returns the absolute tolerance for the quadrature problem
getAbsoluteToleranceSteadyState()	returns the absolute tolerance for the steady state problem

continues on next page

Table 6 – continued from previous page

getAbsoluteToleranceSteadyStateSensi()	returns the absolute tolerance for the sensitivities of the steady state problem
getCpuTime()	Reads out the CPU time needed for forward solve
getCpuTimeB()	Reads out the CPU time needed for backward solve
getInternalSensitivityMethod()	returns the internal sensitivity method
getInterpolationType()	**rtype** InterpolationType
getLastOrder()	Accessor order
getLinearMultistepMethod()	returns the linear system multistep method
getLinearSolver()	**rtype** LinearSolver
getMaxSteps()	returns the maximum number of solver steps for the forward problem
getMaxStepsBackwardProblem()	returns the maximum number of solver steps for the backward problem
getMaxTime()	Returns the maximum time allowed for integration
getNewtonDampingFactorLowerBound()	Get a lower bound of the damping factor used in the Newton solver
getNewtonDampingFactorMode()	Get a state of the damping factor used in the Newton solver
getNewtonMaxSteps()	Get maximum number of allowed Newton steps for steady state computation
getNewtonStepSteadyStateCheck()	Returns how convergence checks for steadystate computation are performed.
getNonlinearSolverIteration()	returns the nonlinear system solution method
getNumErrTestFails()	Accessor netf
getNumErrTestFailsB()	Accessor netfB
getNumNonlinSolvConvFails()	Accessor nnlscf
getNumNonlinSolvConvFailsB()	Accessor nnlscfB
getNumRhsEvals()	Accessor nrhs
getNumRhsEvalsB()	Accessor nrhsB
getNumSteps()	Accessor ns
getNumStepsB()	Accessor nsB
getRelativeTolerance()	Get the relative tolerances for the forward problem
getRelativeToleranceB()	Returns the relative tolerances for the adjoint sensitivity problem
getRelativeToleranceFSA()	Returns the relative tolerances for the forward sensitivity problem
getRelativeToleranceQuadratures()	Returns the relative tolerance for the quadrature problem
getRelativeToleranceSteadyState()	returns the relative tolerance for the steady state problem
getRelativeToleranceSteadyStateSensi()	returns the relative tolerance for the sensitivities of the steady state problem
getReturnDataReportingMode()	returns the ReturnData reporting mode
getSensiSteadyStateCheck()	Returns how convergence checks for steadystate computation are performed.
getSensitivityMethod()	Return current sensitivity method

continues on next page

Table 6 – continued from previous page

getSensitivityMethodPreequilibration()	Return current sensitivity method during preequilibration
getSensitivityOrder()	Get sensitivity order
getStabilityLimitFlag()	returns stability limit detection mode
getStateOrdering()	Gets KLU / SuperLUMT state ordering mode
getSteadyStateSensiToleranceFactor()	returns the steady state sensitivity simulation tolerance factor.
getSteadyStateToleranceFactor()	returns the steady state simulation tolerance factor.
gett()	current solver timepoint
nplist()	number of parameters with which the solver was initialized
nquad()	number of quadratures with which the solver was initialized
nx()	number of states with which the solver was initialized
setAbsoluteTolerance(atol)	Sets the absolute tolerances for the forward problem
setAbsoluteToleranceB(atol)	Sets the absolute tolerances for the backward problem for adjoint sensitivity analysis
setAbsoluteToleranceFSA(atol)	Sets the absolute tolerances for the forward sensitivity problem
setAbsoluteToleranceQuadratures(atol)	sets the absolute tolerance for the quadrature problem
setAbsoluteToleranceSteadyState(atol)	sets the absolute tolerance for the steady state problem
setAbsoluteToleranceSteadyStateSensi(atol)	sets the absolute tolerance for the sensitivities of the steady state problem
setInternalSensitivityMethod(ism)	sets the internal sensitivity method
setInterpolationType(interpType)	sets the interpolation of the forward solution that is used for the backwards problem
setLinearMultistepMethod(lmm)	sets the linear system multistep method
setLinearSolver(linsol)	**type linsol** LinearSolver
setMaxSteps(maxsteps)	sets the maximum number of solver steps for the forward problem
setMaxStepsBackwardProblem(maxsteps)	sets the maximum number of solver steps for the backward problem
setMaxTime(maxtime)	Set the maximum time allowed for integration
setNewtonDampingFactorLowerBound(...)	Set a lower bound of the damping factor in the Newton solver
setNewtonDampingFactorMode(dampingFactorMode)	Turn on/off a damping factor in the Newton method
setNewtonMaxSteps(newton_maxsteps)	Set maximum number of allowed Newton steps for steady state computation
setNewtonStepSteadyStateCheck(flag)	Sets how convergence checks for steadystate computation are performed.
setNonlinearSolverIteration(iter)	sets the nonlinear system solution method
setRelativeTolerance(rtol)	Sets the relative tolerances for the forward problem
setRelativeToleranceB(rtol)	Sets the relative tolerances for the adjoint sensitivity problem
setRelativeToleranceFSA(rtol)	Sets the relative tolerances for the forward sensitivity problem
setRelativeToleranceQuadratures(rtol)	sets the relative tolerance for the quadrature problem
setRelativeToleranceSteadyState(rtol)	sets the relative tolerance for the steady state problem

continues on next page

Table 6 – continued from previous page

setRelativeToleranceSteadyStateSensi(rtol)	sets the relative tolerance for the sensitivities of the steady state problem
setReturnDataReportingMode(rdrm)	sets the ReturnData reporting mode
setSensiSteadyStateCheck(flag)	Sets for which variables convergence checks for steadystate computation are performed.
setSensitivityMethod(sensi_meth)	Set sensitivity method
setSensitivityMethodPreequilibration(...)	Set sensitivity method for preequilibration
setSensitivityOrder(sensi)	Set the sensitivity order
setStabilityLimitFlag(stldet)	set stability limit detection mode
setStateOrdering(ordering)	Sets KLU / SuperLUMT state ordering mode
setSteadyStateSensiToleranceFactor(factor)	set the steady state sensitivity simulation tolerance factor.
setSteadyStateToleranceFactor(factor)	set the steady state simulation tolerance factor.
startTimer()	Start timer for tracking integration time
switchForwardSensisOff()	Disable forward sensitivity integration (used in steady state sim)
timeExceeded()	Check whether maximum integration time was exceeded

Attributes

logger

Methods

__init__(**args*, ***kwargs*)

clone() → Iterable[*amici.amici.Solver*]

Clone this instance

Return type *Solver*

Returns The clone

computingASA() → bool

check if ASA is being computed

Return type boolean

Returns flag

computingFSA() → bool

check if FSA is being computed

Return type boolean

Returns flag

getAbsoluteTolerance() → float

Get the absolute tolerances for the forward problem

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Return type float

Returns absolute tolerances

`getAbsoluteToleranceB()` → float

Returns the absolute tolerances for the backward problem for adjoint sensitivity analysis

Return type float

Returns absolute tolerances

`getAbsoluteToleranceFSA()` → float

Returns the absolute tolerances for the forward sensitivity problem

Return type float

Returns absolute tolerances

`getAbsoluteToleranceQuadratures()` → float

returns the absolute tolerance for the quadrature problem

Return type float

Returns absolute tolerance

`getAbsoluteToleranceSteadyState()` → float

returns the absolute tolerance for the steady state problem

Return type float

Returns absolute tolerance

`getAbsoluteToleranceSteadyStateSensi()` → float

returns the absolute tolerance for the sensitivities of the steady state problem

Return type float

Returns absolute tolerance

`getCpuTime()` → float

Reads out the CPU time needed for forward solve

Return type float

Returns cpu_time

`getCpuTimeB()` → float

Reads out the CPU time needed for backward solve

Return type float

Returns cpu_timeB

`getInternalSensitivityMethod()` → *amici.amici.InternalSensitivityMethod*

returns the internal sensitivity method

Return type *InternalSensitivityMethod*

Returns internal sensitivity method

`getInterpolationType()` → *amici.amici.InterpolationType*

Return type *InterpolationType*

Returns

getLastOrder() → *amici.amici.IntVector*

Accessor order

Return type *IntVector*

Returns order

getLinearMultistepMethod() → *amici.amici.LinearMultistepMethod*

returns the linear system multistep method

Return type *LinearMultistepMethod*

Returns linear system multistep method

getLinearSolver() → *amici.amici.LinearSolver*

Return type *LinearSolver*

Returns

getMaxSteps() → int

returns the maximum number of solver steps for the forward problem

Return type int

Returns maximum number of solver steps

getMaxStepsBackwardProblem() → int

returns the maximum number of solver steps for the backward problem

Return type int

Returns maximum number of solver steps

getMaxTime() → float

Returns the maximum time allowed for integration

Return type float

Returns Time in seconds

getNewtonDampingFactorLowerBound() → float

Get a lower bound of the damping factor used in the Newton solver

Return type float

Returns

getNewtonDampingFactorMode() → *amici.amici.NewtonDampingFactorMode*

Get a state of the damping factor used in the Newton solver

Return type *NewtonDampingFactorMode*

Returns

getNewtonMaxSteps() → int

Get maximum number of allowed Newton steps for steady state computation

Return type int

Returns

`getNewtonStepSteadyStateCheck()` → bool

Returns how convergence checks for steadystate computation are performed. If activated, convergence checks are limited to every 25 steps in the simulation solver to limit performance impact.

Return type boolean

Returns boolean flag indicating newton step (true) or the right hand side (false)

`getNonlinearSolverIteration()` → *amici.amici.NonlinearSolverIteration*

returns the nonlinear system solution method

Return type *NonlinearSolverIteration*

Returns

`getNumErrTestFails()` → *amici.amici.IntVector*

Accessor netf

Return type *IntVector*

Returns netf

`getNumErrTestFailsB()` → *amici.amici.IntVector*

Accessor netfB

Return type *IntVector*

Returns netfB

`getNumNonlinSolvConvFails()` → *amici.amici.IntVector*

Accessor nnlscf

Return type *IntVector*

Returns nnlscf

`getNumNonlinSolvConvFailsB()` → *amici.amici.IntVector*

Accessor nnlscfB

Return type *IntVector*

Returns nnlscfB

`getNumRhsEvals()` → *amici.amici.IntVector*

Accessor nrhs

Return type *IntVector*

Returns nrhs

`getNumRhsEvalsB()` → *amici.amici.IntVector*

Accessor nrhsB

Return type *IntVector*

Returns nrhsB

`getNumSteps()` → *amici.amici.IntVector*

Accessor ns

Return type *IntVector*

Returns ns

`getNumStepsB()` → *amici.amici.IntVector*

Accessor nsB

Return type *IntVector*

Returns nsB

`getRelativeTolerance()` → float

Get the relative tolerances for the forward problem

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Return type float

Returns relative tolerances

`getRelativeToleranceB()` → float

Returns the relative tolerances for the adjoint sensitivity problem

Return type float

Returns relative tolerances

`getRelativeToleranceFSA()` → float

Returns the relative tolerances for the forward sensitivity problem

Return type float

Returns relative tolerances

`getRelativeToleranceQuadratures()` → float

Returns the relative tolerance for the quadrature problem

Return type float

Returns relative tolerance

`getRelativeToleranceSteadyState()` → float

returns the relative tolerance for the steady state problem

Return type float

Returns relative tolerance

`getRelativeToleranceSteadyStateSensi()` → float

returns the relative tolerance for the sensitivities of the steady state problem

Return type float

Returns relative tolerance

`getReturnDataReportingMode()` → *amici.amici.RDataReporting*

returns the ReturnData reporting mode

Return type *RDataReporting*

Returns ReturnData reporting mode

`getSensiSteadyStateCheck()` → bool

Returns how convergence checks for steadystate computation are performed.

Return type boolean

Returns boolean flag indicating state and sensitivity equations (true) or only state variables (false).

getSensitivityMethod() → *amici.amici.SensitivityMethod*

Return current sensitivity method

Return type *SensitivityMethod*

Returns method enum

getSensitivityMethodPreequilibration() → *amici.amici.SensitivityMethod*

Return current sensitivity method during preequilibration

Return type *SensitivityMethod*

Returns method enum

getSensitivityOrder() → *amici.amici.SensitivityOrder*

Get sensitivity order

Return type *SensitivityOrder*

Returns sensitivity order

getStabilityLimitFlag() → bool

returns stability limit detection mode

Return type boolean

Returns stldet can be false (deactivated) or true (activated)

getStateOrdering() → int

Gets KLU / SuperLUMT state ordering mode

Return type int

Returns State-ordering as integer according to SUNLinSolKLU::StateOrdering or SUNLinSol-SuperLUMT::StateOrdering (which differ).

getSteadyStateSensiToleranceFactor() → float

returns the steady state sensitivity simulation tolerance factor.

Steady state sensitivity simulation tolerances are the product of the sensitivity simulation tolerances and this factor, unless manually set with *set(Absolute/Relative)ToleranceSteadyStateSensi()*.

Return type float

Returns steady state simulation tolerance factor

getSteadyStateToleranceFactor() → float

returns the steady state simulation tolerance factor.

Steady state simulation tolerances are the product of the simulation tolerances and this factor, unless manually set with *set(Absolute/Relative)ToleranceSteadyState()*.

Return type float

Returns steady state simulation tolerance factor

gett() → float

current solver timepoint

Return type float

Returns t

nplist() → int

number of parameters with which the solver was initialized

Return type int

Returns sx.getLength()

nquad() → int

number of quadratures with which the solver was initialized

Return type int

Returns xQB.getLength()

nx() → int

number of states with which the solver was initialized

Return type int

Returns x.getLength()

setAbsoluteTolerance(*atol: float*) → None

Sets the absolute tolerances for the forward problem

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setAbsoluteToleranceB(*atol: float*) → None

Sets the absolute tolerances for the backward problem for adjoint sensitivity analysis

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setAbsoluteToleranceFSA(*atol: float*) → None

Sets the absolute tolerances for the forward sensitivity problem

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setAbsoluteToleranceQuadratures(*atol: float*) → None

sets the absolute tolerance for the quadrature problem

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setAbsoluteToleranceSteadyState(*atol: float*) → None

sets the absolute tolerance for the steady state problem

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setAbsoluteToleranceSteadyStateSensi(*atol: float*) → None

sets the absolute tolerance for the sensitivities of the steady state problem

Parameters **atol** (float) – absolute tolerance (non-negative number)

Return type None

setInternalSensitivityMethod(*ism: amici.amici.InternalSensitivityMethod*) → None
sets the internal sensitivity method

Parameters **ism** (*amici.amici.InternalSensitivityMethod*) – internal sensitivity method

Return type None

setInterpolationType(*interpType: amici.amici.InterpolationType*) → None
sets the interpolation of the forward solution that is used for the backwards problem

Parameters **interpType** (*amici.amici.InterpolationType*) – interpolation type

Return type None

setLinearMultistepMethod(*lmm: amici.amici.LinearMultistepMethod*) → None
sets the linear system multistep method

Parameters **lmm** (*amici.amici.LinearMultistepMethod*) – linear system multistep method

Return type None

setLinearSolver(*linsol: amici.amici.LinearSolver*) → None

Parameters **linsol** (*amici.amici.LinearSolver*) –

Return type None

setMaxSteps(*maxsteps: int*) → None
sets the maximum number of solver steps for the forward problem

Parameters **maxsteps** (int) – maximum number of solver steps (positive number)

Return type None

setMaxStepsBackwardProblem(*maxsteps: int*) → None
sets the maximum number of solver steps for the backward problem

Parameters **maxsteps** (int) – maximum number of solver steps (non-negative number)

Notes: default behaviour (100 times the value for the forward problem) can be restored by passing maxsteps=0

Return type None

setMaxTime(*maxtime: float*) → None
Set the maximum time allowed for integration

Parameters **maxtime** (float) – Time in seconds

Return type None

setNewtonDampingFactorLowerBound(*dampingFactorLowerBound: float*) → None
Set a lower bound of the damping factor in the Newton solver

Parameters **dampingFactorLowerBound** (float) –

Return type None

setNewtonDampingFactorMode(*dampingFactorMode: amici.amici.NewtonDampingFactorMode*) → None
Turn on/off a damping factor in the Newton method

Parameters **dampingFactorMode** (*amici.amici.NewtonDampingFactorMode*) –

Return type None

setNewtonMaxSteps(*newton_maxsteps: int*) → None

Set maximum number of allowed Newton steps for steady state computation

Parameters **newton_maxsteps** (int) –

Return type None

setNewtonStepSteadyStateCheck(*flag: bool*) → None

Sets how convergence checks for steadystate computation are performed.

Parameters **flag** (bool) – boolean flag to pick newton step (true) or the right hand side (false, default)

Return type None

setNonlinearSolverIteration(*iter:* amici.amici.NonlinearSolverIteration) → None

sets the nonlinear system solution method

Parameters **iter** (*amici.amici.NonlinearSolverIteration*) – nonlinear system solution method

Return type None

setRelativeTolerance(*rtol: float*) → None

Sets the relative tolerances for the forward problem

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setRelativeToleranceB(*rtol: float*) → None

Sets the relative tolerances for the adjoint sensitivity problem

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setRelativeToleranceFSA(*rtol: float*) → None

Sets the relative tolerances for the forward sensitivity problem

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setRelativeToleranceQuadratures(*rtol: float*) → None

sets the relative tolerance for the quadrature problem

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setRelativeToleranceSteadyState(*rtol: float*) → None

sets the relative tolerance for the steady state problem

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setRelativeToleranceSteadyStateSensi(*rtol: float*) → None

sets the relative tolerance for the sensitivities of the steady state problem

Parameters **rtol** (float) – relative tolerance (non-negative number)

Return type None

setReturnDataReportingMode(*rdrm: amici.amici.RDataReporting*) → None

sets the ReturnData reporting mode

Parameters **rdrm** (*amici.amici.RDataReporting*) – ReturnData reporting mode

Return type None

setSensiSteadyStateCheck(*flag: bool*) → None

Sets for which variables convergence checks for steadystate computation are performed.

Parameters **flag** (bool) – boolean flag to pick state and sensitivity equations (true, default) or only state variables (false).

Return type None

setSensitivityMethod(*sensi_meth: amici.amici.SensitivityMethod*) → None

Set sensitivity method

Parameters **sensi_meth** (*amici.amici.SensitivityMethod*) –

Return type None

setSensitivityMethodPreequilibration(*sensi_meth_preeq: amici.amici.SensitivityMethod*) → None

Set sensitivity method for preequilibration

Parameters **sensi_meth_preeq** (*amici.amici.SensitivityMethod*) –

Return type None

setSensitivityOrder(*sensi: amici.amici.SensitivityOrder*) → None

Set the sensitivity order

Parameters **sensi** (*amici.amici.SensitivityOrder*) – sensitivity order

Return type None

setStabilityLimitFlag(*stldet: bool*) → None

set stability limit detection mode

Parameters **stldet** (bool) – can be false (deactivated) or true (activated)

Return type None

setStateOrdering(*ordering: int*) → None

Sets KLU / SuperLUMT state ordering mode

This only applies when linsol is set to LinearSolver::KLU or LinearSolver::SuperLUMT. Mind the difference between SUNLinSolKLU::StateOrdering and SUNLinSolSuperLUMT::StateOrdering.

Parameters **ordering** (int) – state ordering

Return type None

setSteadyStateSensiToleranceFactor(*factor: float*) → None

set the steady state sensitivity simulation tolerance factor.

Steady state sensitivity simulation tolerances are the product of the sensitivity simulation tolerances and this factor, unless manually set with *set(Absolute/Relative)ToleranceSteadyStateSensi()*.

Parameters **factor** (float) – tolerance factor (non-negative number)

Return type None

setSteadyStateToleranceFactor(*factor: float*) → None

set the steady state simulation tolerance factor.

Steady state simulation tolerances are the product of the simulation tolerances and this factor, unless manually set with *set(Absolute/Relative)ToleranceSteadyState()*.

Parameters **factor** (float) – tolerance factor (non-negative number)

Return type None

startTimer() → None

Start timer for tracking integration time

Return type None

switchForwardSensisOff() → None

Disable forward sensitivity integration (used in steady state sim)

Return type None

timeExceeded() → bool

Check whether maximum integration time was exceeded

Return type boolean

Returns True if the maximum integration time was exceeded, false otherwise.

amici.amici.SolverPtr

class amici.amici.**SolverPtr**(**args*)

Swig-Generated class that implements smart pointers to Solver as objects.

Attributes

logger

amici.amici.SteadyStateSensitivityMode

class amici.amici.**SteadyStateSensitivityMode**(*value*)

An enumeration.

Attributes

newtonOnly
integrationOnly
integrateIfNewtonFails

amici.amici.SteadyStateStatus

class amici.amici.**SteadyStateStatus**(*value*)

An enumeration.

Attributes

failed_too_long_simulation
failed_damping
failed_factorization
failed_convergence
failed
not_run
success

amici.amici.SteadyStateStatusVector

class amici.amici.**SteadyStateStatusVector**(**args*)

amici.amici.StringDoubleMap

class amici.amici.**StringDoubleMap**(**args*)

Swig-Generated class templating `Dict` [`str`, `float`] to facilitate interfacing with C++ bindings.

amici.amici.StringVector

class amici.amici.**StringVector**(**args*)

Swig-Generated class templating common python types including `Iterable` [`str`] and `numpy.array` [`str`] to facilitate interfacing with C++ bindings.

Functions Summary

`backtraceString`(maxFrames[, first_frame])	Returns the current backtrace as std::string
`compiledWithOpenMP`()	AMICI extension was compiled with OpenMP?
`enum`(prefix)	
`getScaledParameter`(unscaledParameter, scaling)	Apply parameter scaling according to *scaling*
`getUnscaledParameter`(scaledParameter, scaling)	Remove parameter scaling according to *scaling*
`parameterScalingFromIntVector`(intVec)	Swig-Generated class, which, in contrast to other Vector classes, does not allow for simple interoperability with common Python types, but must be created using `amici.amici.parameterScalingFromIntVector()`
`runAmiciSimulation`(solver, edata, model[, ...])	Core integration routine.
`runAmiciSimulations`(solver, edatas, model, ...)	Same as runAmiciSimulation, but for multiple ExpData instances.
`scaleParameters`(bufferUnscaled, pscale, ...)	Apply parameter scaling according to *scaling*
`simulation_status_to_str`(status)	Get the string representation of the given simulation status code (see ReturnData::status).
`unscaleParameters`(bufferScaled, pscale, ...)	Remove parameter scaling according to the parameter scaling in pscale

Functions

`amici.amici.`**`backtraceString`**(*maxFrames: int*, *first_frame: int = 0*) → str

Returns the current backtrace as std::string

Parameters

- **`maxFrames`** (`int`) – Number of frames to include
- **`first_frame`** (`int`) – Index of first frame to include

Return type str

Returns Backtrace

`amici.amici.`**`compiledWithOpenMP`**() → bool

AMICI extension was compiled with OpenMP?

Return type `bool`

`amici.amici.`**`enum`**(*prefix*)

`amici.amici.`**`getScaledParameter`**(*unscaledParameter: float*, *scaling:* amici.amici.ParameterScaling) → float

Apply parameter scaling according to *scaling*

Parameters

- **`unscaledParameter`** (`float`) –
- **`scaling`** (`amici.amici.ParameterScaling`) – parameter scaling

Return type float

Returns Scaled parameter

amici.amici.**getUnscaledParameter**(*scaledParameter: float*, *scaling:* amici.amici.ParameterScaling) → float

Remove parameter scaling according to *scaling*

Parameters

- **scaledParameter** (float) – scaled parameter
- **scaling** (*amici.amici.ParameterScaling*) – parameter scaling

Return type float

Returns Unscaled parameter

amici.amici.**parameterScalingFromIntVector**(*intVec:* amici.amici.IntVector) → *amici.amici.ParameterScalingVector*

Swig-Generated class, which, in contrast to other Vector classes, does not allow for simple interoperability with common Python types, but must be created using *amici.amici.parameterScalingFromIntVector()*

Return type *amici.amici.ParameterScalingVector*

amici.amici.**runAmiciSimulation**(*solver:* amici.amici.Solver, *edata:* amici.amici.ExpData, *model:* amici.amici.Model, *rethrow: bool = False*) → *amici.amici.ReturnData*

Core integration routine. Initializes the solver and runs the forward and backward problem.

Parameters

- **solver** (*amici.amici.Solver*) – Solver instance
- **edata** (*amici.amici.ExpData*) – pointer to experimental data object
- **model** (*amici.amici.Model*) – model specification object
- **rethrow** (bool) – rethrow integration exceptions?

Return type *ReturnData*

Returns rdata pointer to return data object

amici.amici.**runAmiciSimulations**(*solver:* amici.amici.Solver, *edatas:* amici.amici.ExpDataPtrVector, *model:* amici.amici.Model, *failfast: bool*, *num_threads: int*) → Iterable[*amici.amici.ReturnData*]

Same as runAmiciSimulation, but for multiple ExpData instances. When compiled with OpenMP support, this function runs multi-threaded.

Parameters

- **solver** (*amici.amici.Solver*) – Solver instance
- **edatas** (*amici.amici.ExpDataPtrVector*) – experimental data objects
- **model** (*amici.amici.Model*) – model specification object
- **failfast** (bool) – flag to allow early termination
- **num_threads** (int) – number of threads for parallel execution

Return type Iterable[*ReturnData*]

Returns vector of pointers to return data objects

amici.amici.**scaleParameters**(*bufferUnscaled: Iterable[float]*, *pscale: Iterable*[amici.amici.ParameterScaling]*, bufferScaled: Iterable[float]*) → None

Apply parameter scaling according to *scaling*

Parameters

- **bufferUnscaled** (typing.Iterable[float]) –
- **pscale** (typing.Iterable[*amici.amici.ParameterScaling*]) – parameter scaling
- **bufferScaled** (typing.Iterable[float]) – destination

Return type None

amici.amici.**simulation_status_to_str**(*status: int*) → str

Get the string representation of the given simulation status code (see ReturnData::status).

Parameters **status** (int) – Status code

Return type str

Returns Name of the variable representing this status code.

amici.amici.**unscaleParameters**(*bufferScaled: Iterable[float]*, *pscale: Iterable[*amici.amici.ParameterScaling*]*, *bufferUnscaled: Iterable[float]*) → None

Remove parameter scaling according to the parameter scaling in pscale

All vectors must be of same length.

Parameters

- **bufferScaled** (typing.Iterable[float]) – scaled parameters
- **pscale** (typing.Iterable[*amici.amici.ParameterScaling*]) – parameter scaling
- **bufferUnscaled** (typing.Iterable[float]) – unscaled parameters are written to the array

Return type None

10.4.3 amici.sbml_import

SBML Import

This module provides all necessary functionality to import a model specified in the Systems Biology Markup Language (SBML).

Classes

SbmlImporter(sbml_source[, ...])	Class to generate AMICI C++ files for a model provided in the Systems Biology Markup Language (SBML).

amici.sbml_import.SbmlImporter

class amici.sbml_import.**SbmlImporter**(*sbml_source*, *show_sbml_warnings=False*, *from_file=True*)

Class to generate AMICI C++ files for a model provided in the Systems Biology Markup Language (SBML).

Variables

- **show_sbml_warnings** – indicates whether libSBML warnings should be displayed
- **symbols** – dict carrying symbolic definitions

- **sbml_reader** – The libSBML sbml reader

> **Warning:** Not storing this may result in a segfault.

- **sbml_doc** – document carrying the sbml definition

> **Warning:** Not storing this may result in a segfault.

- **sbml** – SBML model to import
- **compartments** – dict of compartment ids and compartment volumes
- **stoichiometric_matrix** – stoichiometric matrix of the model
- **flux_vector** – reaction kinetic laws
- **flux_ids** – identifiers for elements of flux_vector
- **_local_symbols** – model symbols for sympy to consider during sympification see *locals`argument in `sympy.sympify*
- **species_assignment_rules** – Assignment rules for species. Key is symbolic identifier and value is assignment value
- **compartment_assignment_rules** – Assignment rules for compartments. Key is symbolic identifier and value is assignment value
- **parameter_assignment_rules** – assignment rules for parameters, these parameters are not permissible for sensitivity analysis
- **initial_assignments** – initial assignments for parameters, these parameters are not permissible for sensitivity analysis
- **sbml_parser_settings** – sets behaviour of SBML Formula parsing

__init__(*sbml_source*, *show_sbml_warnings=False*, *from_file=True*)

Create a new Model instance.

Parameters

- **sbml_source** (typing.Union[str, pathlib.Path, libsbml.Model]) – Either a path to SBML file where the model is specified, or a model string as created by sbml.sbmlWriter( ).writeSBMLToString() or an instance of *libsbml.Model*.
- **show_sbml_warnings** (bool) – Indicates whether libSBML warnings should be displayed.
- **from_file** (bool) – Whether *sbml_source* is a file name (True, default), or an SBML string

Methods Summary

__init__(sbml_source[, show_sbml_warnings, ...])	Create a new Model instance.
add_d_dt(d_dt, variable, variable0, name)	Creates or modifies species, to implement rate rules for compartments and species, respectively.
add_local_symbol(key, value)	Add local symbols with some sanity checking for duplication which would indicate redefinition of internals, which SBML permits, but we don't.
check_event_support()	Check possible events in the model, as AMICI does currently not support
check_support()	Check whether all required SBML features are supported.
is_assignment_rule_target(element)	Checks if an element has a valid assignment rule in the specified model.
is_rate_rule_target(element)	Checks if an element has a valid assignment rule in the specified model.
process_conservation_laws(ode_model)	Find conservation laws in reactions and species.
sbml2amici(model_name[, output_dir, ...])	Generate and compile AMICI C++ files for the model provided to the constructor.

Methods

__init__(*sbml_source*, *show_sbml_warnings=False*, *from_file=True*)

Create a new Model instance.

Parameters

- **sbml_source** (`typing.Union`[`str`, `pathlib.Path`, `libsbml.Model`]) – Either a path to SBML file where the model is specified, or a model string as created by sbml.sbmlWriter( ).writeSBMLToString() or an instance of *libsbml.Model*.
- **show_sbml_warnings** (`bool`) – Indicates whether libSBML warnings should be displayed.
- **from_file** (`bool`) – Whether *sbml_source* is a file name (True, default), or an SBML string

add_d_dt(*d_dt*, *variable*, *variable0*, *name*)

Creates or modifies species, to implement rate rules for compartments and species, respectively.

Parameters

- **d_dt** (`sympy.core.expr.Expr`) – The rate rule (or, right-hand side of an ODE).
- **variable** (`sympy.core.symbol.Symbol`) – The subject of the rate rule.
- **variable0** (`typing.Union`[`float`, `sympy.core.expr.Expr`]) – The initial value of the variable.
- **name** (`str`) – Species name, only applicable if this function generates a new species

Return type `None`

add_local_symbol(*key*, *value*)

Add local symbols with some sanity checking for duplication which would indicate redefinition of internals, which SBML permits, but we don't.

Parameters

- **key** (str) – local symbol key
- **value** (sympy.core.expr.Expr) – local symbol value

check_event_support()

Check possible events in the model, as AMICI does currently not support

- delays in events
- priorities of events
- events fired at initial time

Furthermore, event triggers are optional (e.g., if an event is fired at initial time, no trigger function is necessary). In this case, warn that this event will have no effect.

Return type None

check_support()

Check whether all required SBML features are supported. Also ensures that the SBML contains at least one reaction, or rate rule, or assignment rule, to produce change in the system over time.

Return type None

is_assignment_rule_target(*element*)

Checks if an element has a valid assignment rule in the specified model.

Parameters **element** (libsbml.SBase) – SBML variable

Return type bool

Returns boolean indicating truth of function name

is_rate_rule_target(*element*)

Checks if an element has a valid assignment rule in the specified model.

Parameters **element** (libsbml.SBase) – SBML variable

Return type bool

Returns boolean indicating truth of function name

process_conservation_laws(*ode_model*)

Find conservation laws in reactions and species.

Parameters **ode_model** – ODEModel object with basic definitions

Return type None

sbml2amici(*model_name*, *output_dir=None*, *observables=None*, *event_observables=None*, *constant_parameters=None*, *sigmas=None*, *event_sigmas=None*, *noise_distributions=None*, *event_noise_distributions=None*, *verbose=40*, *assume_pow_positivity=False*, *compiler=None*, *allow_reinit_fixpar_initcond=True*, *compile=True*, *compute_conservation_laws=True*, *simplify=<function _default_simplify>*, *cache_simplify=False*, *log_as_log10=True*, *generate_sensitivity_code=True*)

Generate and compile AMICI C++ files for the model provided to the constructor.

The resulting model can be imported as a regular Python module (if *compile=True*), or used from Matlab or C++ as described in the documentation of the respective AMICI interface.

Note that this generates model ODEs for changes in concentrations, not amounts unless the *hasOnlySubstanceUnits* attribute has been defined for a particular species.

Sensitivity analysis for local parameters is enabled by creating global parameters _{reactionId}_{localParameterName}.

Parameters

- **model_name** (str) – name of the model/model directory
- **output_dir** (typing.Union[str, pathlib.Path, None]) – see *amici.ode_export.ODEExporter.set_paths()*
- **observables** (typing.Optional[typing.Dict[str, typing.Dict[str, str]]]) – dictionary(observableId:{'name':observableName (optional), 'formula':formulaString)}) to be added to the model
- **event_observables** (typing.Optional[typing.Dict[str, typing.Dict[str, str]]]) – dictionary(eventObservableId:{'name':eventObservableName (optional), 'event':eventId, 'formula':formulaString)}) to be added to the model
- **constant_parameters** (typing.Optional[typing.Iterable[str]]) – list of SBML Ids identifying constant parameters
- **sigmas** (typing.Optional[typing.Dict[str, typing.Union[str, float]]]) – dictionary(observableId: sigma value or (existing) parameter name)
- **event_sigmas** (typing.Optional[typing.Dict[str, typing.Union[str, float]]]) – dictionary(eventObservableId: sigma value or (existing) parameter name)
- **noise_distributions** (typing.Optional[typing.Dict[str, typing.Union[str, typing.Callable]]]) – dictionary(observableId: noise type). If nothing is passed for some observable id, a normal model is assumed as default. Either pass a noise type identifier, or a callable generating a custom noise string.
- **event_noise_distributions** (typing.Optional[typing.Dict[str, typing.Union[str, typing.Callable]]]) – dictionary(eventObservableId: noise type). If nothing is passed for some observable id, a normal model is assumed as default. Either pass a noise type identifier, or a callable generating a custom noise string.
- **verbose** (typing.Union[int, bool]) – verbosity level for logging, `True`/`False` default to `logging.Error`/`logging.DEBUG`
- **assume_pow_positivity** (bool) – if set to `True`, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors
- **compiler** (typing.Optional[str]) – distutils/setuptools compiler selection to build the python extension
- **allow_reinit_fixpar_initcond** (bool) – see *amici.ode_export.ODEExporter*
- **compile** (bool) – If `True`, compile the generated Python package, if `False`, just generate code.
- **compute_conservation_laws** (bool) – if set to `True`, conservation laws are automatically computed and applied such that the state-jacobian of the ODE right-hand-side has full rank. This option should be set to `True` when using the Newton algorithm to compute steadystate sensitivities. Conservation laws for constant species are enabled by default. Support for conservation laws for non-constant species is experimental and may be enabled by setting an environment variable `AMICI_EXPERIMENTAL_SBML_NONCONST_CLS` to either `demartino` to use the algorithm proposed by De Martino et al. (2014) https://doi.org/10.1371/journal.pone.0100750, or to any other value to use the deterministic algorithm implemented in `conserved_moieties2.py`. In some cases, the `demartino` may run for a very long time. This has been observed for example in the case of stoichiometric coefficients with many significant digits.
- **simplify** (typing.Optional[typing.Callable]) – see `ODEModel._simplify`

- **cache_simplify** (bool) – see amici.ODEModel.__init__()
- **log_as_log10** (bool) – If True, log in the SBML model will be parsed as log10 (default), if False, log will be parsed as natural logarithm ln
- **generate_sensitivity_code** (bool) – If False, the code required for sensitivity computation will not be generated

Return type None

Functions Summary

assignmentRules2observables(sbml_model[, ...])	Turn assignment rules into observables.
get_species_initial(species)	Extract the initial concentration from a given species
replace_logx(math_str)	Replace logX(.) by log(., X) since sympy cannot parse the former

Functions

amici.sbml_import.**assignmentRules2observables**(*sbml_model*, *filter_function=<function <lambda>>*)

Turn assignment rules into observables.

Parameters

- **sbml_model** (libsbml.Model) – Model to operate on
- **filter_function** (typing.Callable) – Callback function taking assignment variable as input and returning True/False to indicate if the respective rule should be turned into an observable.

Returns A dictionary(observableId:{ 'name': observableName, 'formula': formulaString })

amici.sbml_import.**get_species_initial**(*species*)

Extract the initial concentration from a given species

Parameters **species** (libsbml.Species) – species index

Return type sympy.core.expr.Expr

Returns initial species concentration

amici.sbml_import.**replace_logx**(*math_str*)

Replace logX(.) by log(., X) since sympy cannot parse the former

Parameters **math_str** (typing.Union[str, float, None]) – string for sympification

Return type typing.Union[str, float, None]

Returns sympifiable string

10.4.4 amici.pysb_import

PySB Import

This module provides all necessary functionality to import a model specified in the `pysb.core.Model` format.

Functions Summary

extract_monomers(complex_patterns)	Constructs a list of monomer names contained in complex patterns.
has_fixed_parameter_ic(specie, pysb_model, ...)	Wrapper to interface `ode_export.ODEModel.state_has_fixed_parameter_initial_condition()` from a pysb specie/model arguments
ode_model_from_pysb_importer(model[, ...])	Creates an `amici.ODEModel` instance from a `pysb.Model` instance.
pysb2amici(model[, output_dir, observables, ...])	Generate AMICI C++ files for the provided model.
pysb_model_from_path(pysb_model_file)	Load a pysb model module and return the `pysb.Model` instance

Functions

`amici.pysb_import.`**`extract_monomers`**(*complex_patterns*)

Constructs a list of monomer names contained in complex patterns. Multiplicity of names corresponds to the stoichiometry in the complex.

Parameters **complex_patterns** (`typing.Union`[`pysb.core.ComplexPattern`, `typing.List`[`pysb.core.ComplexPattern`]]) – (list of) complex pattern(s)

Return type `typing.List`[`str`]

Returns list of monomer names

`amici.pysb_import.`**`has_fixed_parameter_ic`**(*specie*, *pysb_model*, *ode_model*)

Wrapper to interface `ode_export.ODEModel.state_has_fixed_parameter_initial_condition()` from a pysb specie/model arguments

Parameters

- **specie** (`pysb.core.ComplexPattern`) – pysb species
- **pysb_model** (`pysb.core.Model`) – pysb model
- **ode_model** (*`amici.ode_export.ODEModel`*) – ODE model

Return type `bool`

Returns `False` if the species does not have an initial condition at all. Otherwise the return value of `ode_export.ODEModel.state_has_fixed_parameter_initial_condition()`

`amici.pysb_import.`**`ode_model_from_pysb_importer`**(*model*, *constant_parameters=None*, *observables=None*, *sigmas=None*, *noise_distributions=None*, *compute_conservation_laws=True*, *simplify=<function powsimp>*, *cache_simplify=False*, *verbose=False*)

Creates an `amici.ODEModel` instance from a `pysb.Model` instance.

Parameters

- **model** (pysb.core.Model) – see *amici.pysb_import.pysb2amici()*
- **constant_parameters** (typing.Optional[typing.List[str]]) – see *amici.pysb_import.pysb2amici()*
- **observables** (typing.Optional[typing.List[str]]) – see *amici.pysb_import.pysb2amici()*
- **sigmas** (typing.Optional[typing.Dict[str, str]]) – dict with names of observable Expressions as keys and names of sigma Expressions as value sigma
- **noise_distributions** (typing.Optional[typing.Dict[str, typing.Union[str, typing.Callable]]]) – see *amici.pysb_import.pysb2amici()*
- **compute_conservation_laws** (bool) – see *amici.pysb_import.pysb2amici()*
- **simplify** (typing.Callable) – see amici.ODEModel._simplify
- **cache_simplify** (bool) – see amici.ODEModel.__init__() Note that there are possible issues with PySB models: https://github.com/AMICI-dev/AMICI/pull/1672
- **verbose** (typing.Union[int, bool]) – verbosity level for logging, True/False default to logging.DEBUG/logging.ERROR

Return type *amici.ode_export.ODEModel*

Returns New ODEModel instance according to pysbModel

amici.pysb_import.**pysb2amici**(*model*, *output_dir=None*, *observables=None*, *constant_parameters=None*, *sigmas=None*, *noise_distributions=None*, *verbose=False*, *assume_pow_positivity=False*, *compiler=None*, *compute_conservation_laws=True*, *compile=True*, *simplify=<function <lambda>>*, *cache_simplify=False*, *generate_sensitivity_code=True*, *model_name=None*)

Generate AMICI C++ files for the provided model.

> **Warning: PySB models with Compartments**
>
> When importing a PySB model with pysb.Compartments, BioNetGen scales reaction fluxes with the compartment size. Instead of using the respective symbols, the compartment size Parameter or Expression is evaluated when generating equations. This may lead to unexpected results if the compartment size parameter is changed for AMICI simulations.

Parameters

- **model** (pysb.core.Model) – pysb model, pysb.Model.name will determine the name of the generated module
- **output_dir** (typing.Union[pathlib.Path, str, None]) – see *amici.ode_export.ODEExporter.set_paths()*
- **observables** (typing.Optional[typing.List[str]]) – list of pysb.core.Expression or pysb.core.Observable names in the provided model that should be mapped to observables
- **sigmas** (typing.Optional[typing.Dict[str, str]]) – dict of pysb.core.Expression names that should be mapped to sigmas

- **noise_distributions** (typing.Optional[typing.Dict[str, typing.Union[str, typing.Callable]]]) – dict with names of observable Expressions as keys and a noise type identifier, or a callable generating a custom noise formula string (see *amici.import_utils.noise_distribution_to_cost_function()*). If nothing is passed for some observable id, a normal model is assumed as default.
- **constant_parameters** (typing.Optional[typing.List[str]]) – list of pysb.core.Parameter names that should be mapped as fixed parameters
- **verbose** (typing.Union[int, bool]) – verbosity level for logging, True/False default to logging.DEBUG/logging.ERROR
- **assume_pow_positivity** (bool) – if set to True, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors
- **compiler** (typing.Optional[str]) – distutils/setuptools compiler selection to build the python extension
- **compute_conservation_laws** (bool) – if set to True, conservation laws are automatically computed and applied such that the state-jacobian of the ODE right-hand-side has full rank. This option should be set to True when using the Newton algorithm to compute steadystates
- **compile** (bool) – If True, build the python module for the generated model. If false, just generate the source code.
- **simplify** (typing.Callable) – see amici.ODEModel._simplify
- **cache_simplify** (bool) – see amici.ODEModel.__init__() Note that there are possible issues with PySB models: https://github.com/AMICI-dev/AMICI/pull/1672
- **generate_sensitivity_code** (bool) – if set to False, code for sensitivity computation will not be generated
- **model_name** (typing.Optional[str]) – Name for the generated model module. If None, pysb.Model.name will be used.

amici.pysb_import.**pysb_model_from_path**(*pysb_model_file*)

Load a pysb model module and return the pysb.Model instance

Parameters **pysb_model_file** (typing.Union[str, pathlib.Path]) – Full or relative path to the PySB model module

Return type pysb.core.Model

Returns The pysb Model instance

10.4.5 amici.bngl_import

BNGL Import

This module provides all necessary functionality to import a model specified in the *BNGL* format.

Functions Summary

bngl2amici(bngl_model, *args, **kwargs)	Generate AMICI C++ files for the provided model.

Functions

amici.bngl_import.**bngl2amici**(*bngl_model*, **args*, ***kwargs*)

Generate AMICI C++ files for the provided model.

Parameters

- **bngl_model** (str) – bngl model file, model name will determine the name of the generated module
- **args** – see *amici.pysb_import.pysb2amici()* for additional arguments
- **kwargs** – see *amici.pysb_import.pysb2amici()* for additional arguments

Return type None

10.4.6 amici.petab_import

PEtab Import

Import a model in the petab (https://github.com/PEtab-dev/PEtab) format into AMICI.

Functions Summary

check_model(amici_model, petab_problem)	Check that the model is consistent with the PEtab problem.
element_is_state(sbml_model, sbml_id)	Does the element with ID *sbml_id* correspond to a state variable?
get_fixed_parameters(petab_problem[, ...])	Determine, set and return fixed model parameters.
get_observation_model(observable_df)	Get observables, sigmas, and noise distributions from PEtab observation table in a format suitable for *amici.sbml_import.SbmlImporter.sbml2amici()*.
import_model([sbml_model, condition_table, ...])	Create AMICI model from PEtab problem
import_model_sbml([sbml_model, ...])	Create AMICI model from PEtab problem
import_petab_problem(petab_problem[, ...])	Import model from petab problem.
main()	Command line interface to import a model in the PEtab (https://github.com/PEtab-dev/PEtab/) format into AMICI.
petab_noise_distributions_to_amici(observable_df)	Map from the petab to the amici format of noise distribution identifiers.
petab_scale_to_amici_scale(scale_str)	Convert PEtab parameter scaling string to AMICI scaling integer
show_model_info(sbml_model)	Log some model quantities
species_to_parameters(species_ids, sbml_model)	Turn a SBML species into parameters and replace species references inside the model instance.

Functions

amici.petab_import.**check_model**(*amici_model*, *petab_problem*)

Check that the model is consistent with the PEtab problem.

Return type None

amici.petab_import.**element_is_state**(*sbml_model*, *sbml_id*)

Does the element with ID *sbml_id* correspond to a state variable?

Return type bool

amici.petab_import.**get_fixed_parameters**(*petab_problem*, *non_estimated_parameters_as_constants=True*)

Determine, set and return fixed model parameters.

Non-estimated parameters and parameters specified in the condition table are turned into constants (unless they are overridden). Only global SBML parameters are considered. Local parameters are ignored.

Parameters

- **petab_problem** (petab.problem.Problem) – The PEtab problem instance
- **non_estimated_parameters_as_constants** – Whether parameters marked as non-estimated in PEtab should be considered constant in AMICI. Setting this to `True` will reduce model size and simulation times. If sensitivities with respect to those parameters are required, this should be set to `False`.

Return type typing.List[str]

Returns List of IDs of parameters which are to be considered constant.

amici.petab_import.**get_observation_model**(*observable_df*)

Get observables, sigmas, and noise distributions from PEtab observation table in a format suitable for *amici.sbml_import.SbmlImporter.sbml2amici()*.

Parameters **observable_df** (pandas.core.frame.DataFrame) – PEtab observables table

Return type typing.Tuple[typing.Dict[str, typing.Dict[str, str]], typing.Dict[str, str], typing.Dict[str, typing.Union[str, float]]]

Returns Tuple of dicts with observables, noise distributions, and sigmas.

amici.petab_import.**import_model**(*sbml_model=None*, *condition_table=None*, *observable_table=None*, *measurement_table=None*, *petab_problem=None*, *model_name=None*, *model_output_dir=None*, *verbose=True*, *allow_reinit_fixpar_initcond=True*, *validate=True*, *non_estimated_parameters_as_constants=True*, ***kwargs*)

Create AMICI model from PEtab problem

Parameters

- **sbml_model** (typing.Union[str, pathlib.Path, libsbml.Model, None]) – PEtab SBML model or SBML file name. Deprecated, pass `petab_problem` instead.
- **condition_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab condition table. If provided, parameters from there will be turned into AMICI constant parameters (i.e. parameters w.r.t. which no sensitivities will be computed). Deprecated, pass `petab_problem` instead.
- **observable_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab observable table. Deprecated, pass `petab_problem` instead.

- **measurement_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab measurement table. Deprecated, pass petab_problem instead.
- **petab_problem** (typing.Optional[petab.problem.Problem]) – PEtab problem.
- **model_name** (typing.Optional[str]) – Name of the generated model. If model file name was provided, this defaults to the file name without extension, otherwise the SBML model ID will be used.
- **model_output_dir** (typing.Union[pathlib.Path, str, None]) – Directory to write the model code to. Will be created if doesn't exist. Defaults to current directory.
- **verbose** (typing.Union[bool, int, None]) – Print/log extra information.
- **allow_reinit_fixpar_initcond** (bool) – See *amici.ode_export.ODEExporter*. Must be enabled if initial states are to be reset after preequilibration.
- **validate** (bool) – Whether to validate the PEtab problem
- **non_estimated_parameters_as_constants** – Whether parameters marked as non-estimated in PEtab should be considered constant in AMICI. Setting this to True will reduce model size and simulation times. If sensitivities with respect to those parameters are required, this should be set to False.
- **kwargs** – Additional keyword arguments to be passed to *amici.sbml_import.SbmlImporter.sbml2amici()*.

Return type *amici.sbml_import.SbmlImporter*

Returns The created *amici.sbml_import.SbmlImporter* instance.

amici.petab_import.**import_model_sbml**(*sbml_model=None*, *condition_table=None*, *observable_table=None*, *measurement_table=None*, *petab_problem=None*, *model_name=None*, *model_output_dir=None*, *verbose=True*, *allow_reinit_fixpar_initcond=True*, *validate=True*, *non_estimated_parameters_as_constants=True*, ***kwargs*)

Create AMICI model from PEtab problem

Parameters

- **sbml_model** (typing.Union[str, pathlib.Path, libsbml.Model, None]) – PEtab SBML model or SBML file name. Deprecated, pass petab_problem instead.
- **condition_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab condition table. If provided, parameters from there will be turned into AMICI constant parameters (i.e. parameters w.r.t. which no sensitivities will be computed). Deprecated, pass petab_problem instead.
- **observable_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab observable table. Deprecated, pass petab_problem instead.
- **measurement_table** (typing.Union[str, pathlib.Path, pandas.core.frame.DataFrame, None]) – PEtab measurement table. Deprecated, pass petab_problem instead.
- **petab_problem** (typing.Optional[petab.problem.Problem]) – PEtab problem.
- **model_name** (typing.Optional[str]) – Name of the generated model. If model file name was provided, this defaults to the file name without extension, otherwise the SBML model ID will be used.

- **model_output_dir** (typing.Union[pathlib.Path, str, None]) – Directory to write the model code to. Will be created if doesn't exist. Defaults to current directory.
- **verbose** (typing.Union[bool, int, None]) – Print/log extra information.
- **allow_reinit_fixpar_initcond** (bool) – See *amici.ode_export.ODEExporter*. Must be enabled if initial states are to be reset after preequilibration.
- **validate** (bool) – Whether to validate the PEtab problem
- **non_estimated_parameters_as_constants** – Whether parameters marked as non-estimated in PEtab should be considered constant in AMICI. Setting this to True will reduce model size and simulation times. If sensitivities with respect to those parameters are required, this should be set to False.
- **kwargs** – Additional keyword arguments to be passed to *amici.sbml_import.SbmlImporter.sbml2amici()*.

Return type *amici.sbml_import.SbmlImporter*

Returns The created *amici.sbml_import.SbmlImporter* instance.

amici.petab_import.**import_petab_problem**(*petab_problem*, *model_output_dir=None*, *model_name=None*, *force_compile=False*, *non_estimated_parameters_as_constants=True*, ***kwargs*)

Import model from petab problem.

Parameters

- **petab_problem** (petab.problem.Problem) – A petab problem containing all relevant information on the model.
- **model_output_dir** (typing.Union[pathlib.Path, str, None]) – Directory to write the model code to. Will be created if doesn't exist. Defaults to current directory.
- **model_name** (typing.Optional[str]) – Name of the generated model. If model file name was provided, this defaults to the file name without extension, otherwise the model ID will be used.
- **force_compile** (bool) – Whether to compile the model even if the target folder is not empty, or the model exists already.
- **non_estimated_parameters_as_constants** – Whether parameters marked as non-estimated in PEtab should be considered constant in AMICI. Setting this to True will reduce model size and simulation times. If sensitivities with respect to those parameters are required, this should be set to False.
- **kwargs** – Additional keyword arguments to be passed to *amici.sbml_import.SbmlImporter.sbml2amici()*.

Return type *amici.amici.Model*

Returns The imported model.

amici.petab_import.**main**()

Command line interface to import a model in the PEtab (https://github.com/PEtab-dev/PEtab/) format into AMICI.

amici.petab_import.**petab_noise_distributions_to_amici**(*observable_df*)

Map from the petab to the amici format of noise distribution identifiers.

Parameters **observable_df** (pandas.core.frame.DataFrame) – PEtab observable table

Return type typing.Dict[str, str]

Returns Dictionary of observable_id => AMICI noise-distributions

`amici.petab_import.`**`petab_scale_to_amici_scale`**`(`*scale_str*`)`

Convert PEtab parameter scaling string to AMICI scaling integer

Return type `int`

`amici.petab_import.`**`show_model_info`**`(`*sbml_model*`)`

Log some model quantities

`amici.petab_import.`**`species_to_parameters`**`(`*species_ids*, *sbml_model*`)`

Turn a SBML species into parameters and replace species references inside the model instance.

Parameters

- **`species_ids`** (`typing.List[str]`) – List of SBML species ID to convert to parameters with the same ID as the replaced species.
- **`sbml_model`** (`libsbml.Model`) – SBML model to modify

Return type `typing.List[str]`

Returns List of IDs of species which have been converted to parameters

10.4.7 amici.petab_import_pysb

PySB-PEtab Import

Import a model in the PySB-adapted `petab` (https://github.com/PEtab-dev/PEtab) format into AMICI.

Classes

`PysbPetabProblem`([pysb_model])	Representation of a PySB-model-based PEtab problem

amici.petab_import_pysb.PysbPetabProblem

`class` `amici.petab_import_pysb.`**`PysbPetabProblem`**`(`*pysb_model=None*, **args*, ***kwargs*`)`

Representation of a PySB-model-based PEtab problem

This class extends `petab.Problem` with a PySB model. The model is augmented with the observation model based on the PEtab observable table. For now, a dummy SBML model is created which allows used the existing SBML-PEtab API.

Variables **`pysb_model`** – PySB model instance from of this PEtab problem.

`__init__``(`*pysb_model=None*, **args*, ***kwargs*`)`

Constructor

Parameters

- **`pysb_model`** (`typing.Optional[pysb.core.Model]`) – PySB model instance for this PEtab problem
- **`args`** – See `petab.Problem.__init__()`
- **`kwargs`** – See `petab.Problem.__init__()`

Methods Summary

`__init__`([pysb_model])	Constructor
`create_parameter_df`(*args, **kwargs)	Create a new PEtab parameter table
`from_combine`(filename)	Read PEtab COMBINE archive (http://co.mbine.org/documents/archive).
`from_files`([condition_file, ...])	Factory method to load model and tables from files.
`from_yaml`(yaml_config[, flatten])	Factory method to load model and tables as specified by YAML file.
`get_lb`([free, fixed, scaled])	Generic function to get lower parameter bounds.
`get_model_parameters`()	See `petab.sbml.get_model_parameters()`
`get_observable_ids`()	Returns dictionary of observable ids.
`get_optimization_parameter_scales`()	Return list of optimization parameter scaling strings.
`get_optimization_parameters`()	Return list of optimization parameter IDs.
`get_optimization_to_simulation_parameter_mapping`(...)	See `petab.parameter_mapping.get_optimization_to_simulation_parameter_mapping()`, to which all keyword arguments are forwarded.
`get_simulation_conditions_from_measurement_df`()	See petab.get_simulation_conditions
`get_ub`([free, fixed, scaled])	Generic function to get upper parameter bounds.
`get_x_ids`([free, fixed])	Generic function to get parameter ids.
`get_x_nominal`([free, fixed, scaled])	Generic function to get parameter nominal values.
`sample_parameter_startpoints`([n_starts])	Create starting points for optimization
`scale_parameters`(x_dict)	Scale parameter values.
`to_files`([sbml_file, condition_file, ...])	Write PEtab tables to files for this problem
`to_files_generic`(prefix_path)	Save a PEtab problem to generic file names.
`unscale_parameters`(x_dict)	Unscale parameter values.

Attributes

`lb`	Parameter table lower bounds.
`lb_scaled`	Parameter table lower bounds with applied parameter scaling
`ub`	Parameter table upper bounds
`ub_scaled`	Parameter table upper bounds with applied parameter scaling
`x_fixed_ids`	Parameter table parameter IDs, for fixed parameters.
`x_fixed_indices`	Parameter table non-estimated parameter indices.
`x_free_ids`	Parameter table parameter IDs, for free parameters.
`x_free_indices`	Parameter table estimated parameter indices.
`x_ids`	Parameter table parameter IDs
`x_nominal`	Parameter table nominal values
`x_nominal_fixed`	Parameter table nominal values, for fixed parameters.
`x_nominal_fixed_scaled`	Parameter table nominal values with applied parameter scaling, for fixed parameters.
`x_nominal_free`	Parameter table nominal values, for free parameters.
`x_nominal_free_scaled`	Parameter table nominal values with applied parameter scaling, for free parameters.
`x_nominal_scaled`	Parameter table nominal values with applied parameter scaling

Methods

__init__(*pysb_model=None*, **args*, ***kwargs*)

Constructor

Parameters

- **pysb_model** (typing.Optional[pysb.core.Model]) – PySB model instance for this PEtab problem
- **args** – See `petab.Problem.__init__()`
- **kwargs** – See `petab.Problem.__init__()`

create_parameter_df(**args*, ***kwargs*)

Create a new PEtab parameter table

See *create_parameter_df()*.

static from_combine(*filename*)

Read PEtab COMBINE archive (http://co.mbine.org/documents/archive).

See also `petab.create_combine_archive()`.

Parameters **filename** (typing.Union[pathlib.Path, str]) – Path to the PEtab-COMBINE archive

Return type Problem

Returns A `petab.Problem` instance.

static from_files(*condition_file=None*, *measurement_file=None*, *parameter_file=None*, *visualization_files=None*, *observable_files=None*, *pysb_model_file=None*, *flatten=False*)

Factory method to load model and tables from files.

Parameters

- **condition_file** (typing.Union[str, pathlib.Path, typing.Iterable[typing.Union[str, pathlib.Path]], None]) – PEtab condition table
- **measurement_file** (typing.Union[str, pathlib.Path, typing.Iterable[typing.Union[str, pathlib.Path]], None]) – PEtab measurement table
- **parameter_file** (typing.Union[str, pathlib.Path, typing.Iterable[typing.Union[str, pathlib.Path]], None]) – PEtab parameter table
- **visualization_files** (typing.Union[str, pathlib.Path, typing.Iterable[typing.Union[str, pathlib.Path]], None]) – PEtab visualization tables
- **observable_files** (typing.Union[str, pathlib.Path, typing.Iterable[typing.Union[str, pathlib.Path]], None]) – PEtab observables tables
- **pysb_model_file** (typing.Union[str, pathlib.Path, None]) – PySB model file
- **flatten** (bool) – Flatten the petab problem

Return type *amici.petab_import_pysb.PysbPetabProblem*

Returns Petab Problem

static **from_yaml**(*yaml_config*, *flatten=False*)

Factory method to load model and tables as specified by YAML file.

NOTE: The PySB model is currently expected in the YAML file under `sbml_files`.

Parameters

- **yaml_config** (`typing.Union`[`typing.Dict`, `pathlib.Path`, `str`]) – PEtab configuration as dictionary or YAML file name
- **flatten** (`bool`) – Flatten the petab problem

Return type *amici.petab_import_pysb.PysbPetabProblem*

Returns Petab Problem

get_lb(*free=True*, *fixed=True*, *scaled=False*)

Generic function to get lower parameter bounds.

Parameters

- **free** (`bool`) – Whether to return free parameters, i.e. parameters to estimate.
- **fixed** (`bool`) – Whether to return fixed parameters, i.e. parameters not to estimate.
- **scaled** (`bool`) – Whether to scale the values according to the parameter scale, or return them on linear scale.

Return type The lower parameter bounds.

get_model_parameters()

See `petab.sbml.get_model_parameters()`

get_observable_ids()

Returns dictionary of observable ids.

get_optimization_parameter_scales()

Return list of optimization parameter scaling strings.

See `petab.parameters.get_optimization_parameters()`.

get_optimization_parameters()

Return list of optimization parameter IDs.

See `petab.parameters.get_optimization_parameters()`.

get_optimization_to_simulation_parameter_mapping(***kwargs*)

See `petab.parameter_mapping.get_optimization_to_simulation_parameter_mapping()`, to which all keyword arguments are forwarded.

get_simulation_conditions_from_measurement_df()

See petab.get_simulation_conditions

get_ub(*free=True*, *fixed=True*, *scaled=False*)

Generic function to get upper parameter bounds.

Parameters

- **free** (`bool`) – Whether to return free parameters, i.e. parameters to estimate.
- **fixed** (`bool`) – Whether to return fixed parameters, i.e. parameters not to estimate.
- **scaled** (`bool`) – Whether to scale the values according to the parameter scale, or return them on linear scale.

Return type The upper parameter bounds.

get_x_ids(*free=True*, *fixed=True*)

Generic function to get parameter ids.

Parameters

- **free** (bool) – Whether to return free parameters, i.e. parameters to estimate.
- **fixed** (bool) – Whether to return fixed parameters, i.e. parameters not to estimate.

Return type The parameter IDs.

get_x_nominal(*free=True*, *fixed=True*, *scaled=False*)

Generic function to get parameter nominal values.

Parameters

- **free** (bool) – Whether to return free parameters, i.e. parameters to estimate.
- **fixed** (bool) – Whether to return fixed parameters, i.e. parameters not to estimate.
- **scaled** (bool) – Whether to scale the values according to the parameter scale, or return them on linear scale.

Return type The parameter nominal values.

sample_parameter_startpoints(*n_starts=100*)

Create starting points for optimization

See `petab.sample_parameter_startpoints()`.

scale_parameters(*x_dict*)

Scale parameter values.

Parameters **x_dict** (typing.Dict[str, float]) – Keys are parameter IDs in the PEtab problem, values are unscaled parameter values.

Return type The scaled parameter values.

to_files(*sbml_file=None*, *condition_file=None*, *measurement_file=None*, *parameter_file=None*, *visualization_file=None*, *observable_file=None*, *yaml_file=None*, *prefix_path=None*, *relative_paths=True*, *model_file=None*)

Write PEtab tables to files for this problem

Writes PEtab files for those entities for which a destination was passed.

NOTE: If this instance was created from multiple measurement or visualization tables, they will be merged and written to a single file.

Parameters

- **sbml_file** (typing.Union[pathlib.Path, str, None]) – SBML model destination
- **model_file** (typing.Union[pathlib.Path, str, None]) – Model destination
- **condition_file** (typing.Union[pathlib.Path, str, None]) – Condition table destination
- **measurement_file** (typing.Union[pathlib.Path, str, None]) – Measurement table destination
- **parameter_file** (typing.Union[pathlib.Path, str, None]) – Parameter table destination

- **visualization_file** (typing.Union[pathlib.Path, str, None]) – Visualization table destination
- **observable_file** (typing.Union[pathlib.Path, str, None]) – Observables table destination
- **yaml_file** (typing.Union[pathlib.Path, str, None]) – YAML file destination
- **prefix_path** (typing.Union[pathlib.Path, str, None]) – Specify a prefix to all paths, to avoid specifying the prefix for all paths individually. NB: the prefix is added to paths before relative_paths is handled.
- **relative_paths** (bool) – whether all paths in the YAML file should be relative to the location of the YAML file. If False, then paths are left unchanged.

Raises ValueError – If a destination was provided for a non-existing entity.

Return type None

to_files_generic(*prefix_path*)

Save a PEtab problem to generic file names.

The PEtab problem YAML file is always created. PEtab data files are only created if the PEtab problem contains corresponding data (e.g. a PEtab visualization TSV file is only created if the PEtab problem has one).

Parameters **prefix_path** (typing.Union[str, pathlib.Path]) – Specify a prefix to all paths, to avoid specifying the prefix for all paths individually. NB: the prefix is added to paths before relative_paths is handled downstream in petab.yaml.create_problem_yaml().

Return type str

Returns The path to the PEtab problem YAML file.

unscale_parameters(*x_dict*)

Unscale parameter values.

Parameters **x_dict** (typing.Dict[str, float]) – Keys are parameter IDs in the PEtab problem, values are scaled parameter values.

Return type The unscaled parameter values.

Functions Summary

create_dummy_sbml(pysb_model[, observable_ids])	Create SBML dummy model for to use PySB models with PEtab.
import_model_pysb(petab_problem[, ...])	Create AMICI model from PySB-PEtab problem

Functions

amici.petab_import_pysb.**create_dummy_sbml**(*pysb_model*, *observable_ids=None*)

Create SBML dummy model for to use PySB models with PEtab.

Model must at least contain PEtab problem parameter and noise parameters for observables.

Parameters

- **pysb_model** (pysb.core.Model) – PySB model
- **observable_ids** (typing.Optional[typing.Iterable[str]]) – Observable IDs

Return type typing.Tuple[libsbml.Model, libsbml.SBMLDocument]

Returns A dummy SBML model and document.

amici.petab_import_pysb.**import_model_pysb**(*petab_problem*, *model_output_dir=None*, *verbose=True*, *model_name=None*, ***kwargs*)

Create AMICI model from PySB-PEtab problem

Parameters

- **petab_problem** (*amici.petab_import_pysb.PysbPetabProblem*) – PySB PEtab problem
- **model_output_dir** (typing.Union[pathlib.Path, str, None]) – Directory to write the model code to. Will be created if doesn't exist. Defaults to current directory.
- **verbose** (typing.Union[bool, int, None]) – Print/log extra information.
- **model_name** (typing.Optional[str]) – Name of the generated model module
- **kwargs** – Additional keyword arguments to be passed to *amici.pysb_import.pysb2amici()*.

Return type None

10.4.8 amici.petab_objective

PEtab Objective

Functionality related to running simulations or evaluating the objective function as defined by a PEtab problem

Functions Summary

create_edata_for_condition(condition, ...)	Get *amici.amici.ExpData* for the given PEtab condition.
create_edatas(amici_model, petab_problem[, ...])	Create list of *amici.amici.ExpData* objects for PEtab problem.
create_parameter_mapping(petab_problem, ...)	Generate AMICI specific parameter mapping.
create_parameter_mapping_for_condition(...)	Generate AMICI specific parameter mapping for condition.
create_parameterized_edatas(amici_model, ...)	Create list of :class:amici.ExpData objects with parameters filled in.
rdatas_to_measurement_df(rdatas, model, ...)	Create a measurement dataframe in the PEtab format from the passed `rdatas` and own information.
rdatas_to_simulation_df(rdatas, model, ...)	Create a PEtab simulation dataframe from *amici.amici.ReturnData* s.
simulate_petab(petab_problem, amici_model[, ...])	Simulate PEtab model.
subset_dict(full, *args)	Get subset of dictionary based on provided keys

Functions

amici.petab_objective.**create_edata_for_condition**(*condition*, *measurement_df*, *amici_model*, *petab_problem*, *observable_ids*)

Get *amici.amici.ExpData* for the given PEtab condition.

Sets timepoints, observed data and sigmas.

Parameters

- **condition** (`typing.Union`[`typing.Dict`, `pandas.core.series.Series`]) – `pandas.DataFrame` row with `preequilibrationConditionId` and `simulationConditionId`.
- **measurement_df** (`pandas.core.frame.DataFrame`) – `pandas.DataFrame` with measurements for the given condition.
- **amici_model** (`typing.Union`[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model
- **petab_problem** (`petab.problem.Problem`) – Underlying PEtab problem
- **observable_ids** (`typing.List`[`str`]) – List of observable IDs

Return type `amici.swig_wrappers.ExpData`

Returns ExpData instance.

amici.petab_objective.**create_edatas**(*amici_model*, *petab_problem*, *simulation_conditions=None*)

Create list of *amici.amici.ExpData* objects for PEtab problem.

Parameters

- **amici_model** (`typing.Union`[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model.
- **petab_problem** (`petab.problem.Problem`) – Underlying PEtab problem.
- **simulation_conditions** (`typing.Union`[`pandas.core.frame.DataFrame`, `typing.Dict`, `None`]) – Result of `petab.get_simulation_conditions()`. Can be provided to save time if this has be obtained before.

Return type typing.List[amici.swig_wrappers.ExpData]

Returns List with one *amici.amici.ExpData* per simulation condition, with filled in timepoints and data.

amici.petab_objective.**create_parameter_mapping**(*petab_problem, simulation_conditions, scaled_parameters, amici_model, **parameter_mapping_kwargs*)

Generate AMICI specific parameter mapping.

Parameters

- **petab_problem** (petab.problem.Problem) – PEtab problem
- **simulation_conditions** (typing.Union[pandas.core.frame.DataFrame, typing.List[typing.Dict]]) – Result of petab.get_simulation_conditions(). Can be provided to save time if this has been obtained before.
- **scaled_parameters** (bool) – If True, problem_parameters are assumed to be on the scale provided in the PEtab parameter table and will be unscaled. If False, they are assumed to be in linear scale.
- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model.
- **parameter_mapping_kwargs** – Optional keyword arguments passed to petab.get_optimization_to_simulation_parameter_mapping(). To allow changing fixed PEtab problem parameters (estimate=0), use fill_fixed_parameters=False.

Return type *amici.parameter_mapping.ParameterMapping*

Returns List of the parameter mappings.

amici.petab_objective.**create_parameter_mapping_for_condition**(*parameter_mapping_for_condition, condition, petab_problem, amici_model*)

Generate AMICI specific parameter mapping for condition.

Parameters

- **parameter_mapping_for_condition** (typing.Tuple[typing.Dict[str, typing.Union[str, numbers.Number]], typing.Dict[str, typing.Union[str, numbers.Number]], typing.Dict[str, str], typing.Dict[str, str]]) – Preliminary parameter mapping for condition.
- **condition** (typing.Union[pandas.core.series.Series, typing.Dict]) – pandas.DataFrame row with preequilibrationConditionId and simulationConditionId.
- **petab_problem** (petab.problem.Problem) – Underlying PEtab problem.
- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model.

Return type *amici.parameter_mapping.ParameterMappingForCondition*

Returns The parameter and parameter scale mappings, for fixed preequilibration, fixed simulation, and variable simulation parameters, and then the respective scalings.

amici.petab_objective.**create_parameterized_edatas**(*amici_model, petab_problem, problem_parameters, scaled_parameters=False, parameter_mapping=None, simulation_conditions=None*)

Create list of :class:amici.ExpData objects with parameters filled in.

Parameters

- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI Model assumed to be compatible with petab_problem.
- **petab_problem** (petab.problem.Problem) – PEtab problem to work on.
- **problem_parameters** (typing.Dict[str, numbers.Number]) – Run simulation with these parameters. If None, PEtab *nominalValues* will be used). To be provided as dict, mapping PEtab problem parameters to SBML IDs.
- **scaled_parameters** (bool) – If True, problem_parameters are assumed to be on the scale provided in the PEtab parameter table and will be unscaled. If False, they are assumed to be in linear scale.
- **parameter_mapping** (typing.Optional[*amici.parameter_mapping.ParameterMapping*]) – Optional precomputed PEtab parameter mapping for efficiency, as generated by *create_parameter_mapping()*.
- **simulation_conditions** (typing.Union[pandas.core.frame.DataFrame, typing.Dict, None]) – Result of petab.get_simulation_conditions(). Can be provided to save time if this has been obtained before.

Return type typing.List[amici.swig_wrappers.ExpData]

Returns List with one *amici.amici.ExpData* per simulation condition, with filled in timepoints, data and parameters.

amici.petab_objective.**rdatas_to_measurement_df**(*rdatas*, *model*, *measurement_df*)

Create a measurement dataframe in the PEtab format from the passed rdatas and own information.

Parameters

- **rdatas** (typing.Sequence[*amici.amici.ReturnData*]) – A sequence of rdatas with the ordering of petab.get_simulation_conditions().
- **model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model used to generate rdatas.
- **measurement_df** (pandas.core.frame.DataFrame) – PEtab measurement table used to generate rdatas.

Return type pandas.core.frame.DataFrame

Returns A dataframe built from the rdatas in the format of measurement_df.

amici.petab_objective.**rdatas_to_simulation_df**(*rdatas*, *model*, *measurement_df*)

Create a PEtab simulation dataframe from *amici.amici.ReturnData* s.

See *rdatas_to_measurement_df()* for details, only that model outputs will appear in column simulation instead of measurement.

Return type pandas.core.frame.DataFrame

amici.petab_objective.**simulate_petab**(*petab_problem*, *amici_model*, *solver=None*, *problem_parameters=None*, *simulation_conditions=None*, *edatas=None*, *parameter_mapping=None*, *scaled_parameters=False*, *log_level=30*, *num_threads=1*, *failfast=True*)

Simulate PEtab model.

Parameters

- **petab_problem** (petab.problem.Problem) – PEtab problem to work on.

- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI Model assumed to be compatible with petab_problem.
- **solver** (typing.Optional[*amici.amici.Solver*]) – An AMICI solver. Will use default options if None.
- **problem_parameters** (typing.Optional[typing.Dict[str, float]]) – Run simulation with these parameters. If None, PEtab *nominalValues* will be used). To be provided as dict, mapping PEtab problem parameters to SBML IDs.
- **simulation_conditions** (typing.Union[pandas.core.frame.DataFrame, typing.Dict, None]) – Result of petab.get_simulation_conditions(). Can be provided to save time if this has be obtained before. Not required if edatas and parameter_mapping are provided.
- **edatas** (typing.Optional[typing.List[typing.Union[amici.swig_wrappers.ExpData, *amici.amici.ExpDataPtr*]]]) – Experimental data. Parameters are inserted in-place for simulation.
- **parameter_mapping** (typing.Optional[*amici.parameter_mapping.ParameterMapping*]) – Optional precomputed PEtab parameter mapping for efficiency, as generated by *create_parameter_mapping()*.
- **scaled_parameters** (typing.Optional[bool]) – If True, problem_parameters are assumed to be on the scale provided in the PEtab parameter table and will be unscaled. If False, they are assumed to be in linear scale.
- **log_level** (int) – Log level, see *amici.logging* module.
- **num_threads** (int) – Number of threads to use for simulating multiple conditions (only used if compiled with OpenMP).
- **failfast** (bool) – Returns as soon as an integration failure is encountered, skipping any remaining simulations.

Return type typing.Dict[str, typing.Any]

Returns

Dictionary of

- cost function value (LLH),
- list of *amici.amici.ReturnData* (RDATAS),
- list of *amici.amici.ExpData* (EDATAS),

corresponding to the different simulation conditions. For ordering of simulation conditions, see petab.Problem.get_simulation_conditions_from_measurement_df().

amici.petab_objective.**subset_dict**(*full*, **args*)

Get subset of dictionary based on provided keys

Parameters

- **full** (typing.Dict[typing.Any, typing.Any]) – Dictionary to subset
- **args** (typing.Collection[typing.Any]) – Collections of keys to be contained in the different subsets

Return type typing.Iterator[typing.Dict[typing.Any, typing.Any]]

Returns subsetted dictionary

10.4.9 amici.petab_simulate

PEtab Simulate

Functionality related to the use of AMICI for simulation with PEtab's Simulator class.

Use cases:

- generate data for use with PEtab's plotting methods
- generate synthetic data

Classes

PetabSimulator(*args[, amici_model])	Implementation of the PEtab *Simulator* class that uses AMICI.

amici.petab_simulate.PetabSimulator

class amici.petab_simulate.**PetabSimulator**(**args*, *amici_model=None*, ***kwargs*)

Implementation of the PEtab *Simulator* class that uses AMICI.

__init__(**args*, *amici_model=None*, ***kwargs*)

Initialize the simulator.

Initialize the simulator with sufficient information to perform a simulation. If no working directory is specified, a temporary one is created.

Parameters

- **petab_problem** – A PEtab problem.
- **working_dir** – All simulator-specific output files will be saved here. This directory and its contents may be modified and deleted, and should be considered ephemeral.

Methods Summary

__init__(*args[, amici_model])	Initialize the simulator.
add_noise(simulation_df[, noise_scaling_factor])	Add noise to simulated data.
remove_working_dir([force])	Remove the simulator working directory, and all files within.
simulate([noise, noise_scaling_factor])	Simulate a PEtab problem, optionally with noise.
simulate_without_noise(**kwargs)	See `petab.simulate.Simulator.simulate()` docstring.

Methods

`__init__`(**args*, *amici_model=None*, ***kwargs*)

Initialize the simulator.

Initialize the simulator with sufficient information to perform a simulation. If no working directory is specified, a temporary one is created.

Parameters

- **`petab_problem`** – A PEtab problem.
- **`working_dir`** – All simulator-specific output files will be saved here. This directory and its contents may be modified and deleted, and should be considered ephemeral.

`add_noise`(*simulation_df*, *noise_scaling_factor=1*, ***kwargs*)

Add noise to simulated data.

Parameters

- **`simulation_df`** (`pandas.core.frame.DataFrame`) – A PEtab measurements table that contains simulated data.
- **`noise_scaling_factor`** (`float`) – A multiplier of the scale of the noise distribution.
- **`**kwargs`** – Additional keyword arguments are passed to `sample_noise()`.

Return type `pandas.core.frame.DataFrame`

Returns Simulated data with noise, as a PEtab measurements table.

`remove_working_dir`(*force=False*, ***kwargs*)

Remove the simulator working directory, and all files within.

See the `petab.simulate.Simulator.__init__()` method arguments.

Parameters

- **`force`** (`bool`) – If `True`, the working directory is removed regardless of whether it is a temporary directory.
- **`**kwargs`** – Additional keyword arguments are passed to `shutil.rmtree()`.

Return type `None`

`simulate`(*noise=False*, *noise_scaling_factor=1*, ***kwargs*)

Simulate a PEtab problem, optionally with noise.

Parameters

- **`noise`** (`bool`) – If True, noise is added to simulated data.
- **`noise_scaling_factor`** (`float`) – A multiplier of the scale of the noise distribution.
- **`**kwargs`** – Additional keyword arguments are passed to `petab.simulate.Simulator.simulate_without_noise()`.

Return type `pandas.core.frame.DataFrame`

Returns Simulated data, as a PEtab measurements table.

simulate_without_noise(***kwargs*)

See `petab.simulate.Simulator.simulate()` docstring.

Additional keyword arguments can be supplied to specify arguments for the AMICI PEtab import, simulate, and export methods. See the docstrings for the respective methods for argument options: - *amici.petab_import.import_petab_problem()*, and - *amici.petab_objective.simulate_petab()*.

Note that some arguments are expected to have already been specified in the Simulator constructor (including the PEtab problem).

Return type `pandas.core.frame.DataFrame`

Functions Summary

subset_call(method, kwargs)	Helper function to call a method with the intersection of arguments in the method signature and the supplied arguments.

Functions

amici.petab_simulate.**subset_call**(*method*, *kwargs*)

Helper function to call a method with the intersection of arguments in the method signature and the supplied arguments.

Parameters

- **method** (`typing.Callable`) – The method to be called.
- **kwargs** (`dict`) – The argument superset as a dictionary, similar to ***kwargs* in method signatures.

Returns The output of *method*, called with the applicable arguments in *kwargs*.

10.4.10 amici.import_utils

Miscellaneous functions related to model import, independent of any specific model format

Classes

ObservableTransformation(value)	Different modes of observable transformation.

amici.import_utils.ObservableTransformation

class amici.import_utils.**ObservableTransformation**(*value*)

Different modes of observable transformation.

Attributes

LOG10
LOG
LIN

Functions Summary

cast_to_sym(value, input_name)	Typecasts the value to sympy.Float if possible, and ensures the value is a symbolic expression.
generate_flux_symbol(reaction_index[, name])	Generate identifier symbol for a reaction flux.
generate_measurement_symbol(observable_id)	Generates the appropriate measurement symbol for the provided observable
generate_regularization_symbol(observable_id)	Generates the appropriate regularization symbol for the provided observable
grouper(iterable, n[, fillvalue])	Collect data into fixed-length chunks or blocks
noise_distribution_to_cost_function(...)	Parse noise distribution string to a cost function definition amici can work with.
noise_distribution_to_observable_transformation(...)	Parse noise distribution string and extract observable transformation
smart_subs(element, old, new)	Optimized substitution that checks whether anything needs to be done first
smart_subs_dict(sym, subs[, field, reverse])	Substitutes expressions completely flattening them out.
strip_pysb(symbol)	Strips pysb info from a pysb.Component object
symbol_with_assumptions(name)	Central function to create symbols with consistent, canonical assumptions
toposort_symbols(symbols[, field])	Topologically sort symbol definitions according to their interdependency

Functions

amici.import_utils.**cast_to_sym**(*value*, *input_name*)

Typecasts the value to sympy.Float if possible, and ensures the value is a symbolic expression.

Parameters

- **value** (typing.Union[typing.SupportsFloat, sympy.core.expr.Expr, sympy.logic.boolalg.BooleanAtom]) – value to be cast
- **input_name** (str) – name of input variable

Return type sympy.core.expr.Expr

Returns typecast value

amici.import_utils.**generate_flux_symbol**(*reaction_index*, *name=None*)

Generate identifier symbol for a reaction flux. This function will always return the same unique python object for a given entity.

Parameters

- **reaction_index** (int) – index of the reaction to which the flux corresponds
- **name** (typing.Optional[str]) – an optional identifier of the reaction to which the flux corresponds

Return type sympy.core.symbol.Symbol

Returns identifier symbol

amici.import_utils.**generate_measurement_symbol**(*observable_id*)

Generates the appropriate measurement symbol for the provided observable

Parameters **observable_id** (typing.Union[str, sympy.core.symbol.Symbol]) – symbol (or string representation) of the observable

Returns symbol for the corresponding measurement

amici.import_utils.**generate_regularization_symbol**(*observable_id*)

Generates the appropriate regularization symbol for the provided observable

Parameters **observable_id** (typing.Union[str, sympy.core.symbol.Symbol]) – symbol (or string representation) of the observable

Returns symbol for the corresponding regularization

amici.import_utils.**grouper**(*iterable*, *n*, *fillvalue=None*)

Collect data into fixed-length chunks or blocks

grouper('ABCDEFG', 3, 'x') –> ABC DEF Gxx"

Parameters

- **iterable** (typing.Iterable) – any iterable
- **n** (int) – chunk length
- **fillvalue** (typing.Optional[typing.Any]) – padding for last chunk if length < n

Return type typing.Iterable[typing.Tuple[typing.Any]]

Returns itertools.zip_longest of requested chunks

amici.import_utils.**noise_distribution_to_cost_function**(*noise_distribution*)

Parse noise distribution string to a cost function definition amici can work with.

The noise distributions listed in the following are supported. m denotes the measurement, y the simulation, and σ a distribution scale parameter (currently, AMICI only supports a single distribution parameter).

- *'normal'*, *'lin-normal'*: A normal distribution:

$$\pi(m|y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\, exp\left(-\frac{(m-y)^2}{2\sigma^2}\right)$$

- *'log-normal'*: A log-normal distribution (i.e. log(m) is normally distributed):

$$\pi(m|y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma m}\, exp\left(-\frac{(\log m - \log y)^2}{2\sigma^2}\right)$$

- *'log10-normal'*: A log10-normal distribution (i.e. log10(m) is normally distributed):

$$\pi(m|y,\sigma) = \frac{1}{\sqrt{2\pi}\sigma m \log(10)}\, exp\left(-\frac{(\log_{10} m - \log_{10} y)^2}{2\sigma^2}\right)$$

- *'laplace'*, *'lin-laplace'*: A laplace distribution:

$$\pi(m|y,\sigma) = \frac{1}{2\sigma}\exp\left(-\frac{|m-y|}{\sigma}\right)$$

- *'log-laplace'*: A log-Laplace distribution (i.e. log(m) is Laplace distributed):

$$\pi(m|y,\sigma) = \frac{1}{2\sigma m}\exp\left(-\frac{|\log m - \log y|}{\sigma}\right)$$

- *'log10-laplace'*: A log10-Laplace distribution (i.e. log10(m) is Laplace distributed):

$$\pi(m|y,\sigma) = \frac{1}{2\sigma m\log(10)}\exp\left(-\frac{|\log_{10} m - \log_{10} y|}{\sigma}\right)$$

- *'binomial'*, *'lin-binomial'*: A (continuation of a discrete) binomial distribution, parameterized via the success probability $p = \sigma$:

$$\pi(m|y,\sigma) = \operatorname{Heaviside}(y-m)\cdot\frac{\Gamma(y+1)}{\Gamma(m+1)\Gamma(y-m+1)}\sigma^m(1-\sigma)^{(y-m)}$$

- *'negative-binomial'*, *'lin-negative-binomial'*: A (continuation of a discrete) negative binomial distribution, with with *mean* = *y*, parameterized via success probability *p*:

$$\pi(m|y,\sigma) = \frac{\Gamma(m+r)}{\Gamma(m+1)\Gamma(r)}(1-\sigma)^m\sigma^r$$

  where

$$r = \frac{1-\sigma}{\sigma}y$$

The distributions above are for a single data point. For a collection $D = \{m_i\}_i$ of data points and corresponding simulations $Y = \{y_i\}_i$ and noise parameters $\Sigma = \{\sigma_i\}_i$, AMICI assumes independence, i.e. the full distributions is

$$\pi(D|Y,\Sigma) = \prod_i \pi(m_i|y_i,\sigma_i)$$

AMICI uses the logarithm $\log(\pi(m|y,\sigma)$.

In addition to the above mentioned distributions, it is also possible to pass a function taking a symbol string and returning a log-distribution string with variables '{str_symbol}', 'm{str_symbol}', 'sigma{str_symbol}' for y, m, sigma, respectively.

Parameters **noise_distribution** (`typing.Union`[`str`, `typing.Callable`]) – An identifier specifying a noise model. Possible values are

{*'normal'*, *'lin-normal'*, *'log-normal'*, *'log10-normal'*, *'laplace'*, *'lin-laplace'*, *'log-laplace'*, *'log10-laplace'*, *'binomial'*, *'lin-binomial'*, *'negative-binomial'*, *'lin-negative-binomial'*, *<Callable>*}

For the meaning of the values see above.

Return type `typing.Callable`[[`str`], `str`]

Returns A function that takes a strSymbol and then creates a cost function string (negative log-likelihood) from it, which can be sympified.

amici.import_utils.**noise_distribution_to_observable_transformation**(*noise_distribution*)

Parse noise distribution string and extract observable transformation

Parameters **noise_distribution** (typing.Union[str, typing.Callable]) – see *noise_distribution_to_cost_function()*

Return type *amici.import_utils.ObservableTransformation*

Returns observable transformation

amici.import_utils.**smart_subs**(*element*, *old*, *new*)

Optimized substitution that checks whether anything needs to be done first

Parameters

- **element** (sympy.core.expr.Expr) – substitution target
- **old** (sympy.core.symbol.Symbol) – to be substituted
- **new** (sympy.core.expr.Expr) – subsitution value

Return type sympy.core.expr.Expr

Returns substituted expression

amici.import_utils.**smart_subs_dict**(*sym*, *subs*, *field=None*, *reverse=True*)

Substitutes expressions completely flattening them out. Requires sorting of expressions with toposort.

Parameters

- **sym** (sympy.core.expr.Expr) – Symbolic expression in which expressions will be substituted
- **subs** (typing.Dict[sympy.core.symbol.Symbol, typing.Union[typing.Dict[str, sympy.core.expr.Expr], sympy.core.expr.Expr]]) – Substitutions
- **field** (typing.Optional[str]) – Field of substitution expressions in subs.values(), if applicable
- **reverse** (bool) – Whether ordering in subs should be reversed. Note that substitution requires the reverse order of what is required for evaluation.

Return type sympy.core.expr.Expr

Returns Substituted symbolic expression

amici.import_utils.**strip_pysb**(*symbol*)

Strips pysb info from a pysb.Component object

Parameters **symbol** (sympy.core.basic.Basic) – symbolic expression

Return type sympy.core.basic.Basic

Returns stripped expression

amici.import_utils.**symbol_with_assumptions**(*name*)

Central function to create symbols with consistent, canonical assumptions

Parameters **name** (str) – name of the symbol

Returns symbol with canonical assumptions

amici.import_utils.**toposort_symbols**(*symbols*, *field=None*)

Topologically sort symbol definitions according to their interdependency

Parameters

- **symbols** (typing.Dict[sympy.core.symbol.Symbol, typing.Union[typing.Dict[str, sympy.core.expr.Expr], sympy.core.expr.Expr]]) – symbol definitions
- **field** (typing.Optional[str]) – field of definition.values() that is used to compute inter-dependency

Return type typing.Dict[sympy.core.symbol.Symbol, typing.Union[typing.Dict[str, sympy.core.expr.Expr], sympy.core.expr.Expr]]

Returns ordered symbol definitions

10.4.11 amici.ode_export

C++ Export

This module provides all necessary functionality specify an ODE model and generate executable C++ simulation code. The user generally won't have to directly call any function from this module as this will be done by *amici.pysb_import.pysb2amici()*, *amici.sbml_import.SbmlImporter.sbml2amici()* and *amici.petab_import.import_model()*.

Classes

ODEExporter(ode_model[, outdir, verbose, ...])	The ODEExporter class generates AMICI C++ files for ODE model as defined in symbolic expressions.
ODEModel([verbose, simplify, cache_simplify])	Defines an Ordinary Differential Equation as set of ModelQuantities.
TemplateAmici(template)	Template format used in AMICI (see string.Template for more details).

amici.ode_export.ODEExporter

class amici.ode_export.**ODEExporter**(*ode_model*, *outdir=None*, *verbose=False*, *assume_pow_positivity=False*, *compiler=None*, *allow_reinit_fixpar_initcond=True*, *generate_sensitivity_code=True*, *model_name='model'*)

The ODEExporter class generates AMICI C++ files for ODE model as defined in symbolic expressions.

Variables

- **model** – ODE definition
- **verbose** – more verbose output if True
- **assume_pow_positivity** – if set to true, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors
- **compiler** – distutils/setuptools compiler selection to build the Python extension
- **functions** – carries C++ function signatures and other specifications
- **model_name** – name of the model that will be used for compilation
- **model_path** – path to the generated model specific files
- **model_swig_path** – path to the generated swig files

- **allow_reinit_fixpar_initcond** – indicates whether reinitialization of initial states depending on fixedParameters is allowed for this model
- **_build_hints** – If the given model uses special functions, this set contains hints for model building.
- **generate_sensitivity_code** – Specifies whether code for sensitivity computation is to be generated

---

Note: When importing large models (several hundreds of species or parameters), import time can potentially be reduced by using multiple CPU cores. This is controlled by setting the `AMICI_IMPORT_NPROCS` environment variable to the number of parallel processes that are to be used (default: 1). Note that for small models this may (slightly) increase import times.

__init__(*ode_model*, *outdir=None*, *verbose=False*, *assume_pow_positivity=False*, *compiler=None*, *allow_reinit_fixpar_initcond=True*, *generate_sensitivity_code=True*, *model_name='model'*)

Generate AMICI C++ files for the ODE provided to the constructor.

Parameters

- **ode_model** (*amici.ode_export.ODEModel*) – ODE definition
- **outdir** (typing.Union[pathlib.Path, str, None]) – see *amici.ode_export.ODEExporter.set_paths()*
- **verbose** (typing.Union[bool, int, None]) – verbosity level for logging, `True`/`False` default to `logging.Error`/`logging.DEBUG`
- **assume_pow_positivity** (typing.Optional[bool]) – if set to true, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors
- **compiler** (typing.Optional[str]) – distutils/setuptools compiler selection to build the python extension
- **allow_reinit_fixpar_initcond** (typing.Optional[bool]) – see *amici.ode_export.ODEExporter*
- **generate_sensitivity_code** (typing.Optional[bool]) – specifies whether code required for sensitivity computation will be generated
- **model_name** (typing.Optional[str]) – name of the model to be used during code generation

Methods Summary

__init__(ode_model[, outdir, verbose, ...])	Generate AMICI C++ files for the ODE provided to the constructor.
compile_model()	Compiles the generated code it into a simulatable module
generate_model_code()	Generates the native C++ code for the loaded model and a Matlab script that can be run to compile a mex file from the C++ code
set_name(model_name)	Sets the model name
set_paths([output_dir])	Set output paths for the model and create if necessary

Methods

__init__(*ode_model*, *outdir=None*, *verbose=False*, *assume_pow_positivity=False*, *compiler=None*, *allow_reinit_fixpar_initcond=True*, *generate_sensitivity_code=True*, *model_name='model'*)

Generate AMICI C++ files for the ODE provided to the constructor.

Parameters

- **ode_model** (*amici.ode_export.ODEModel*) – ODE definition
- **outdir** (typing.Union[pathlib.Path, str, None]) – see *amici.ode_export.ODEExporter.set_paths()*
- **verbose** (typing.Union[bool, int, None]) – verbosity level for logging, True/False default to logging.Error/logging.DEBUG
- **assume_pow_positivity** (typing.Optional[bool]) – if set to true, a special pow function is used to avoid problems with state variables that may become negative due to numerical errors
- **compiler** (typing.Optional[str]) – distutils/setuptools compiler selection to build the python extension
- **allow_reinit_fixpar_initcond** (typing.Optional[bool]) – see *amici.ode_export.ODEExporter*
- **generate_sensitivity_code** (typing.Optional[bool]) – specifies whether code required for sensitivity computation will be generated
- **model_name** (typing.Optional[str]) – name of the model to be used during code generation

compile_model()

Compiles the generated code it into a simulatable module

Return type None

generate_model_code()

Generates the native C++ code for the loaded model and a Matlab script that can be run to compile a mex file from the C++ code

Return type None

set_name(*model_name*)

Sets the model name

Parameters **model_name** (str) – name of the model (may only contain upper and lower case letters, digits and underscores, and must not start with a digit)

Return type None

set_paths(*output_dir=None*)

Set output paths for the model and create if necessary

Parameters **output_dir** (typing.Union[pathlib.Path, str, None]) – relative or absolute path where the generated model code is to be placed. If None, this will default to amici-{self.model_name} in the current working directory. will be created if it does not exist.

Return type None

amici.ode_export.ODEModel

class amici.ode_export.**ODEModel**(*verbose=False*, *simplify=<function _default_simplify>*, *cache_simplify=False*)

Defines an Ordinary Differential Equation as set of ModelQuantities. This class provides general purpose interfaces to compute arbitrary symbolic derivatives that are necessary for model simulation or sensitivity computation.

Variables

- **_states** – list of state variables
- **_observables** – list of observables
- **_event_observables** – list of event observables
- **_sigmays** – list of sigmas for observables
- **_sigmazs** – list of sigmas for event observables
- **_parameters** – list of parameters
- **_loglikelihoodys** – list of loglikelihoods for observables
- **_loglikelihoodzs** – list of loglikelihoods for event observables
- **_loglikelihoodrzs** – list of loglikelihoods for event observable regularizations
- **_expressions** – list of expressions instances
- **_conservationlaws** – list of conservation laws
- **_symboldim_funs** – define functions that compute model dimensions, these are functions as the underlying symbolic expressions have not been populated at compile time
- **_eqs** – carries symbolic formulas of the symbolic variables of the model
- **_sparseeqs** – carries linear list of all symbolic formulas for sparsified variables
- **_vals** – carries numeric values of symbolic identifiers of the symbolic variables of the model
- **_names** – carries names of symbolic identifiers of the symbolic variables of the model
- **_syms** – carries symbolic identifiers of the symbolic variables of the model
- **_sparsesyms** – carries linear list of all symbolic identifiers for sparsified variables
- **_colptrs** – carries column pointers for sparsified variables. See SUNMatrixContent_Sparse definition in `sunmatrix/sunmatrix_sparse.h`
- **_rowvals** – carries row values for sparsified variables. See SUNMatrixContent_Sparse definition in `sunmatrix/sunmatrix_sparse.h`
- **_equation_prototype** – defines the attribute from which an equation should be generated via list comprehension (see `ODEModel._generate_equation()`)
- **_variable_prototype** – defines the attribute from which a variable should be generated via list comprehension (see `ODEModel._generate_symbol()`)
- **_value_prototype** – defines the attribute from which a value should be generated via list comprehension (see `ODEModel._generate_value()`)
- **_total_derivative_prototypes** – defines how a total derivative equation is computed for an equation, key defines the name and values should be arguments for ODEModel.totalDerivative()

- **_lock_total_derivative** – add chainvariables to this set when computing total derivative from a partial derivative call to enforce a partial derivative in the next recursion. prevents infinite recursion
- **_simplify** – If not None, this function will be used to simplify symbolic derivative expressions. Receives sympy expressions as only argument. To apply multiple simplifications, wrap them in a lambda expression.
- **_x0_fixedParameters_idx** – Index list of subset of states for which x0_fixedParameters was computed
- **_w_recursion_depth** – recursion depth in w, quantified as nilpotency of dwdw
- **_has_quadratic_nllh** – whether all observables have a gaussian noise model, i.e. whether res and FIM make sense.
- **_code_printer** – Code printer to generate C++ code
- **_z2event** – list of event indices for each event observable

__init__(*verbose=False*, *simplify=<function _default_simplify>*, *cache_simplify=False*)

Create a new ODEModel instance.

Parameters

- **verbose** (typing.Union[bool, int, None]) – verbosity level for logging, True/False default to `logging.DEBUG`/`logging.ERROR`
- **simplify** (typing.Optional[typing.Callable]) – see `ODEModel._simplify()`
- **cache_simplify** (bool) – Whether to cache calls to the simplify method. Can e.g. decrease import times for models with events.

Methods Summary

__init__([verbose, simplify, cache_simplify])	Create a new ODEModel instance.
add_component(component[, insert_first])	Adds a new ModelQuantity to the model.
add_conservation_law(state, total_abundance, ...)	Adds a new conservation law to the model.
colptrs(name)	Returns (and constructs if necessary) the column pointers for a sparsified symbolic variable.
conservation_law_has_multispecies(tcl)	Checks whether a conservation law has multiple species or it just defines one constant species
eq(name)	Returns (and constructs if necessary) the formulas for a symbolic entity.
free_symbols()	Returns list of free symbols that appear in ODE RHS and initial conditions.
generate_basic_variables()	Generates the symbolic identifiers for all variables in `ODEModel._variable_prototype`
get_appearance_counts(idxs)	Counts how often a state appears in the time derivative of another state and expressions for a subset of states
get_conservation_laws()	Returns a list of states with conservation law set
get_observable_transformations()	List of observable transformations
get_solver_indices()	Returns a mapping that maps rdata species indices to solver indices

continues on next page

Table 7 – continued from previous page

import_from_sbml_importer(si[, compute_cls])	Imports a model specification from a `amici.sbml_import.SbmlImporter` instance.
name(name)	Returns (and constructs if necessary) the names of a symbolic variable
num_cons_law()	Number of conservation laws.
num_const()	Number of Constants.
num_eventobs()	Number of Event Observables.
num_events()	Number of Events.
num_expr()	Number of Expressions.
num_obs()	Number of Observables.
num_par()	Number of Parameters.
num_state_reinits()	Number of solver states which would be reinitialized after preequilibration
num_states_rdata()	Number of states.
num_states_solver()	Number of states after applying conservation laws.
parse_events()	This function checks the right-hand side for roots of Heaviside functions or events, collects the roots, removes redundant roots, and replaces the formulae of the found roots by identifiers of AMICI's Heaviside function implementation in the right-hand side
rowvals(name)	Returns (and constructs if necessary) the row values for a sparsified symbolic variable.
sparseeq(name)	Returns (and constructs if necessary) the sparsified formulas for a sparsified symbolic variable.
sparsesym(name[, force_generate])	Returns (and constructs if necessary) the sparsified identifiers for a sparsified symbolic variable.
state_has_conservation_law(ix)	Checks whether the state at specified index has a conservation law set
state_has_fixed_parameter_initial_condition(ix)	Checks whether the state at specified index has a fixed parameter initial condition
state_is_constant(ix)	Checks whether the temporal derivative of the state is zero
sym(name)	Returns (and constructs if necessary) the identifiers for a symbolic entity.
sym_names()	Returns a list of names of generated symbolic variables
sym_or_eq(name, varname)	Returns symbols or equations depending on whether a given variable appears in the function signature or not.
val(name)	Returns (and constructs if necessary) the numeric values of a symbolic entity

Methods

__init__(*verbose=False*, *simplify=<function _default_simplify>*, *cache_simplify=False*)

Create a new ODEModel instance.

Parameters

- **verbose** (typing.Union[bool, int, None]) – verbosity level for logging, True/False default to logging.DEBUG/logging.ERROR
- **simplify** (typing.Optional[typing.Callable]) – see ODEModel._simplify()
- **cache_simplify** (bool) – Whether to cache calls to the simplify method. Can e.g. decrease import times for models with events.

add_component(*component*, *insert_first=False*)

Adds a new ModelQuantity to the model.

Parameters

- **component** (*amici.ode_model.ModelQuantity*) – model quantity to be added
- **insert_first** (typing.Optional[bool]) – whether to add quantity first or last, relevant when components may refer to other components of the same type.

Return type None

add_conservation_law(*state*, *total_abundance*, *coefficients*)

Adds a new conservation law to the model. A conservation law is defined by the conserved quantity $T = \sum_i(a_i * x_i)$, where a_i are coefficients and x_i are different state variables.

Parameters

- **state** (sympy.core.symbol.Symbol) – symbolic identifier of the state that should be replaced by the conservation law (x_j)
- **total_abundance** (sympy.core.symbol.Symbol) – symbolic identifier of the total abundance (T/a_j)
- **coefficients** (typing.Dict[sympy.core.symbol.Symbol, sympy.core.expr.Expr]) – Dictionary of coefficients {x_i: a_i}

Return type None

colptrs(*name*)

Returns (and constructs if necessary) the column pointers for a sparsified symbolic variable.

Parameters **name** (str) – name of the symbolic variable

Return type typing.Union[typing.List[sympy.core.numbers.Number], typing.List[typing.List[sympy.core.numbers.Number]]]

Returns list containing the column pointers

conservation_law_has_multispecies(*tcl*)

Checks whether a conservation law has multiple species or it just defines one constant species

Parameters **tcl** (*amici.ode_model.ConservationLaw*) – conservation law

Return type bool

Returns boolean indicating if conservation_law is not None

eq(*name*)

Returns (and constructs if necessary) the formulas for a symbolic entity.

Parameters **name** (str) – name of the symbolic variable

Return type sympy.matrices.dense.MutableDenseMatrix

Returns matrix of symbolic formulas

free_symbols()

Returns list of free symbols that appear in ODE RHS and initial conditions.

Return type typing.Set[sympy.core.basic.Basic]

generate_basic_variables()

Generates the symbolic identifiers for all variables in `ODEModel._variable_prototype`

Return type None

get_appearance_counts(*idxs*)

Counts how often a state appears in the time derivative of another state and expressions for a subset of states

Parameters **idxs** (typing.List[int]) – list of state indices for which counts are to be computed

Return type typing.List[int]

Returns list of counts for the states ordered according to the provided indices

get_conservation_laws()

Returns a list of states with conservation law set

Return type typing.List[typing.Tuple[sympy.core.symbol.Symbol, sympy.core.expr.Expr]]

Returns list of state identifiers

get_observable_transformations()

List of observable transformations

Return type typing.List[*amici.import_utils.ObservableTransformation*]

Returns list of transformations

get_solver_indices()

Returns a mapping that maps rdata species indices to solver indices

Return type typing.Dict[int, int]

Returns dictionary mapping rdata species indices to solver indices

import_from_sbml_importer(*si*, *compute_cls=True*)

Imports a model specification from a *amici.sbml_import.SbmlImporter* instance.

Parameters

- **si** (*amici.sbml_import.SbmlImporter*) – imported SBML model
- **compute_cls** (typing.Optional[bool]) – whether to compute conservation laws

Return type None

name(*name*)

Returns (and constructs if necessary) the names of a symbolic variable

Parameters **name** (str) – name of the symbolic variable

Return type typing.List[str]

Returns list of names

num_cons_law()

Number of conservation laws.

Return type int

Returns number of conservation laws

num_const()

Number of Constants.

Return type int

Returns number of constant symbols

num_eventobs()

Number of Event Observables.

Return type int

Returns number of event observable symbols

num_events()

Number of Events.

Return type int

Returns number of event symbols (length of the root vector in AMICI)

num_expr()

Number of Expressions.

Return type int

Returns number of expression symbols

num_obs()

Number of Observables.

Return type int

Returns number of observable symbols

num_par()

Number of Parameters.

Return type int

Returns number of parameter symbols

num_state_reinits()

Number of solver states which would be reinitialized after preequilibration

Return type int

Returns number of state variable symbols with reinitialization

num_states_rdata()

Number of states.

Return type int

Returns number of state variable symbols

num_states_solver()

Number of states after applying conservation laws.

Return type int

Returns number of state variable symbols

parse_events()

This function checks the right-hand side for roots of Heaviside functions or events, collects the roots, removes redundant roots, and replaces the formulae of the found roots by identifiers of AMICI's Heaviside function implementation in the right-hand side

Return type None

rowvals(*name*)

Returns (and constructs if necessary) the row values for a sparsified symbolic variable.

Parameters **name** (str) – name of the symbolic variable

Return type typing.Union[typing.List[sympy.core.numbers.Number], typing.List[typing.List[sympy.core.numbers.Number]]]

Returns list containing the row values

sparseeq(*name*)

Returns (and constructs if necessary) the sparsified formulas for a sparsified symbolic variable.

Parameters **name** – name of the symbolic variable

Return type sympy.matrices.dense.MutableDenseMatrix

Returns linearized matrix containing the symbolic formulas

sparsesym(*name*, *force_generate=True*)

Returns (and constructs if necessary) the sparsified identifiers for a sparsified symbolic variable.

Parameters

- **name** (str) – name of the symbolic variable
- **force_generate** (bool) – whether the symbols should be generated if not available

Return type typing.List[str]

Returns linearized Matrix containing the symbolic identifiers

state_has_conservation_law(*ix*)

Checks whether the state at specified index has a conservation law set

Parameters **ix** (int) – state index

Return type bool

Returns boolean indicating if conservation_law is not None

state_has_fixed_parameter_initial_condition(*ix*)

Checks whether the state at specified index has a fixed parameter initial condition

Parameters **ix** (int) – state index

Return type bool

Returns boolean indicating if any of the initial condition free variables is contained in the model constants

state_is_constant(*ix*)

Checks whether the temporal derivative of the state is zero

Parameters **ix** (int) – state index

Return type bool

Returns boolean indicating if constant over time

sym(*name*)

Returns (and constructs if necessary) the identifiers for a symbolic entity.

Parameters **name** (str) – name of the symbolic variable

Return type sympy.matrices.dense.MutableDenseMatrix

Returns matrix of symbolic identifiers

sym_names()

Returns a list of names of generated symbolic variables

Return type typing.List[str]

Returns list of names

sym_or_eq(*name*, *varname*)

Returns symbols or equations depending on whether a given variable appears in the function signature or not.

Parameters

- **name** (str) – name of function for which the signature should be checked
- **varname** (str) – name of the variable which should be contained in the function signature

Return type sympy.matrices.dense.MutableDenseMatrix

Returns the variable symbols if the variable is part of the signature and the variable equations otherwise.

val(*name*)

Returns (and constructs if necessary) the numeric values of a symbolic entity

Parameters **name** (str) – name of the symbolic variable

Return type typing.List[float]

Returns list containing the numeric values

amici.ode_export.TemplateAmici

class amici.ode_export.**TemplateAmici**(*template*)

Template format used in AMICI (see `string.Template` for more details).

Variables **delimiter** – delimiter that identifies template variables

__init__(*template*)

Methods Summary

__init__(template)	
safe_substitute([mapping])	
substitute([mapping])	

Attributes

braceidpattern	
delimiter	
flags	
idpattern	
pattern	

Methods

__init__(*template*)

safe_substitute(*mapping={}*, */*, ***kws*)

substitute(*mapping={}*, */*, ***kws*)

Functions Summary

apply_template(source_file, target_file, ...)	Load source file, apply template substitution as provided in templateData and save as targetFile.
get_function_extern_declaration(fun, name)	Constructs the extern function declaration for a given function
get_model_override_implementation(fun, name)	Constructs `amici::Model::*` override implementation for a given function
get_sunindex_extern_declaration(fun, name, ...)	Constructs the function declaration for an index function of a given function
get_sunindex_override_implementation(fun, ...)	Constructs the `amici::Model` function implementation for an index function of a given function
is_valid_identifier(x)	Check whether *x* is a valid identifier for conditions, parameters, observables.
remove_argument_types(signature)	Strips argument types from a function signature
smart_is_zero_matrix(x)	A faster implementation of sympy's is_zero_matrix
smart_jacobian(eq, sym_var)	Wrapper around symbolic jacobian with some additional checks that reduce computation time for large matrices
smart_multiply(x, y)	Wrapper around symbolic multiplication with some additional checks that reduce computation time for large matrices
var_in_function_signature(name, varname)	Checks if the values for a symbolic variable is passed in the signature of a function

Functions

amici.ode_export.**apply_template**(*source_file*, *target_file*, *template_data*)

Load source file, apply template substitution as provided in templateData and save as targetFile.

Parameters

- **source_file** (`typing.Union`[`str`, `pathlib.Path`]) – relative or absolute path to template file
- **target_file** (`typing.Union`[`str`, `pathlib.Path`]) – relative or absolute path to output file
- **template_data** (`typing.Dict`[`str`, `str`]) – template keywords to substitute (key is template variable without `TemplateAmici.delimiter`)

Return type `None`

amici.ode_export.**get_function_extern_declaration**(*fun*, *name*)

Constructs the extern function declaration for a given function

Parameters

- **fun** (`str`) – function name
- **name** (`str`) – model name

Return type `str`

Returns C++ function definition string

amici.ode_export.**get_model_override_implementation**(*fun*, *name*, *nobody=False*)

Constructs `amici::Model::*` override implementation for a given function

Parameters

- **fun** (str) – function name
- **name** (str) – model name
- **nobody** (bool) – whether the function has a nontrivial implementation

Return type str

Returns C++ function implementation string

amici.ode_export.**get_sunindex_extern_declaration**(*fun*, *name*, *indextype*)

Constructs the function declaration for an index function of a given function

Parameters

- **fun** (str) – function name
- **name** (str) – model name
- **indextype** (str) – index function {'colptrs', 'rowvals'}

Return type str

Returns C++ function declaration string

amici.ode_export.**get_sunindex_override_implementation**(*fun*, *name*, *indextype*, *nobody=False*)

Constructs the `amici::Model` function implementation for an index function of a given function

Parameters

- **fun** (str) – function name
- **name** (str) – model name
- **indextype** (str) – index function {'colptrs', 'rowvals'}
- **nobody** (bool) – whether the corresponding function has a nontrivial implementation

Return type str

Returns C++ function implementation string

amici.ode_export.**is_valid_identifier**(*x*)

Check whether *x* is a valid identifier for conditions, parameters, observables… . Identifiers may only contain upper and lower case letters, digits and underscores, and must not start with a digit.

Parameters **x** (str) – string to check

Return type bool

Returns `True` if valid, `False` otherwise

amici.ode_export.**remove_argument_types**(*signature*)

Strips argument types from a function signature

Parameters **signature** (str) – function signature

Return type str

Returns string that can be used to construct function calls with the same variable names and ordering as in the function signature amici.ode_export.**smart_is_zero_matrix**(*x*)

A faster implementation of sympy's is_zero_matrix

Avoids repeated indexer type checks and double iteration to distinguish False/None. Found to be about 100x faster for large matrices.

Parameters x (typing.Union[sympy.matrices.dense.MutableDenseMatrix, sympy.matrices.sparse.MutableSparseMatrix]) – Matrix to check

Return type bool

amici.ode_export.**smart_jacobian**(*eq*, *sym_var*)

Wrapper around symbolic jacobian with some additional checks that reduce computation time for large matrices

Parameters

- **eq** (sympy.matrices.dense.MutableDenseMatrix) – equation
- **sym_var** (sympy.matrices.dense.MutableDenseMatrix) – differentiation variable

Return type sympy.matrices.sparse.MutableSparseMatrix

Returns jacobian of eq wrt sym_var

amici.ode_export.**smart_multiply**(*x*, *y*)

Wrapper around symbolic multiplication with some additional checks that reduce computation time for large matrices

Parameters

- **x** (typing.Union[sympy.matrices.dense.MutableDenseMatrix, sympy.matrices.sparse.MutableSparseMatrix]) – educt 1
- **y** (sympy.matrices.dense.MutableDenseMatrix) – educt 2

Return type typing.Union[sympy.matrices.dense.MutableDenseMatrix, sympy.matrices.sparse.MutableSparseMatrix]

Returns product

amici.ode_export.**var_in_function_signature**(*name*, *varname*)

Checks if the values for a symbolic variable is passed in the signature of a function

Parameters

- **name** (str) – name of the function
- **varname** (str) – name of the symbolic variable

Return type bool

Returns boolean indicating whether the variable occurs in the function signature

10.4.12 amici.ode_model

Objects for AMICI's internal ODE model representation

Classes

ConservationLaw(identifier, name, value, ...)	A conservation law defines the absolute the total amount of a (weighted) sum of states
Constant(identifier, name, value)	A Constant is a fixed variable in the model with respect to which sensitivities cannot be computed, abbreviated by `k`.
Event(identifier, name, value, state_update)	An Event defines either a SBML event or a root of the argument of a Heaviside function.
EventObservable(identifier, name, value, event)	An Event Observable links model simulations to event related experimental measurements, abbreviated by `z`.
Expression(identifier, name, value)	An Expression is a recurring elements in symbolic formulas.
LogLikelihood(identifier, name, value)	A LogLikelihood defines the distance between measurements and experiments for a particular observable.
LogLikelihoodRZ(identifier, name, value)	Loglikelihood for event observables regularization
LogLikelihoodY(identifier, name, value)	Loglikelihood for observables
LogLikelihoodZ(identifier, name, value)	Loglikelihood for event observables
ModelQuantity(identifier, name, value)	Base class for model components
Observable(identifier, name, value[, ...])	An Observable links model simulations to experimental measurements, abbreviated by `y`.
Parameter(identifier, name, value)	A Parameter is a free variable in the model with respect to which sensitivities may be computed, abbreviated by `p`.
Sigma(identifier, name, value)	A Standard Deviation Sigma rescales the distance between simulations and measurements when computing residuals or objective functions, abbreviated by `sigma{y,z}`.
SigmaY(identifier, name, value)	Standard deviation for observables
SigmaZ(identifier, name, value)	Standard deviation for event observables
State(identifier, name, init, dt)	A State variable defines an entity that evolves with time according to the provided time derivative, abbreviated by `x`.

amici.ode_model.ConservationLaw

class amici.ode_model.**ConservationLaw**(*identifier*, *name*, *value*, *coefficients*, *state_id*)

A conservation law defines the absolute the total amount of a (weighted) sum of states

__init__(*identifier*, *name*, *value*, *coefficients*, *state_id*)

Create a new ConservationLaw instance.

Parameters

- **identifier** (`sympy.core.symbol.Symbol`) – unique identifier of the ConservationLaw
- **name** (`str`) – individual name of the ConservationLaw (does not need to be unique)
- **value** (`sympy.core.expr.Expr`) – formula (sum of states)
- **coefficients** (`typing.Dict`[`sympy.core.symbol.Symbol`, `sympy.core.expr.Expr`]) – coefficients of the states in the sum
- **state_id** (`sympy.core.symbol.Symbol`) – identifier of the state that this conservation law replaces

Methods Summary

__init__(identifier, name, value, ...)	Create a new ConservationLaw instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_ncoeff(state_id)	Computes the normalized coefficient a_i/a_j where i is the index of the provided state_id and j is the index of the state that is replaced by this conservation law.
get_val()	ModelQuantity value
get_x_rdata()	Returns the expression that allows computation of x_rdata for the state that this conservation law replaces.
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*, *coefficients*, *state_id*)

Create a new ConservationLaw instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the ConservationLaw
- **name** (str) – individual name of the ConservationLaw (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula (sum of states)
- **coefficients** (typing.Dict[sympy.core.symbol.Symbol, sympy.core.expr.Expr]) – coefficients of the states in the sum
- **state_id** (sympy.core.symbol.Symbol) – identifier of the state that this conservation law replaces

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_ncoeff(*state_id*)

Computes the normalized coefficient a_i/a_j where i is the index of the provided state_id and j is the index of the state that is replaced by this conservation law. This can be used to compute both dtotal_cl/dx_rdata (=ncoeff) and dx_rdata/dx_solver (=-ncoeff).

Parameters **state_id** – identifier of the state

Return type typing.Union[sympy.core.expr.Expr, int, float]

Returns normalized coefficent of the state

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

get_x_rdata()

Returns the expression that allows computation of x_rdata for the state that this conservation law replaces.

Returns x_rdata expression

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Constant

class amici.ode_model.**Constant**(*identifier*, *name*, *value*)

A Constant is a fixed variable in the model with respect to which sensitivities cannot be computed, abbreviated by k.

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Constant
- **name** (str) – individual name of the Constant (does not need to be unique)
- **value** (numbers.Number) – numeric value

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Constant
- **name** (str) – individual name of the Constant (does not need to be unique)
- **value** (numbers.Number) – numeric value

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Event

class amici.ode_model.**Event**(*identifier*, *name*, *value*, *state_update*, *initial_value=True*)

An Event defines either a SBML event or a root of the argument of a Heaviside function. The Heaviside functions will be tracked via the vector `h` during simulation and are needed to inform the ODE solver about a discontinuity in either the right-hand side or the states themselves, causing a reinitialization of the solver.

__init__(*identifier*, *name*, *value*, *state_update*, *initial_value=True*)

Create a new Event instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Event
- **name** (str) – individual name of the Event (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula for the root / trigger function
- **state_update** (typing.Optional[sympy.core.expr.Expr]) – formula for the bolus function (None for Heaviside functions, zero vector for events without bolus)
- **initial_value** (typing.Optional[bool]) – initial boolean value of the trigger function at t0. If set to *False*, events may trigger at `t==t0`, otherwise not.

Methods Summary

__init__(identifier, name, value, state_update)	Create a new Event instance.
get_id()	ModelQuantity identifier
get_initial_value()	Return the initial value for the root function.
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*, *state_update*, *initial_value=True*)

Create a new Event instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Event
- **name** (str) – individual name of the Event (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula for the root / trigger function
- **state_update** (typing.Optional[sympy.core.expr.Expr]) – formula for the bolus function (None for Heaviside functions, zero vector for events without bolus)
- **initial_value** (typing.Optional[bool]) – initial boolean value of the trigger function at t0. If set to *False*, events may trigger at `t==t0`, otherwise not.

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_initial_value()

Return the initial value for the root function.

Return type bool

Returns initial value formula

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.EventObservable

class amici.ode_model.**EventObservable**(*identifier*, *name*, *value*, *event*, *measurement_symbol=None*, *transformation='lin'*)

An Event Observable links model simulations to event related experimental measurements, abbreviated by `z`.

Variables **_event** – symbolic event identifier

__init__(*identifier*, *name*, *value*, *event*, *measurement_symbol=None*, *transformation='lin'*)

Create a new EventObservable instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – See *Observable.__init__()*.
- **name** (str) – See *Observable.__init__()*.
- **value** (sympy.core.expr.Expr) – See *Observable.__init__()*.
- **transformation** (typing.Optional[*amici.import_utils.ObservableTransformation*]) – See *Observable.__init__()*.
- **event** (sympy.core.symbol.Symbol) – Symbolic identifier of the corresponding event.

Methods Summary

__init__(identifier, name, value, event[, ...])	Create a new EventObservable instance.
get_event()	Get the symbolic identifier of the corresponding event.
get_id()	ModelQuantity identifier
get_measurement_symbol()	**rtype** sympy.core.symbol.Symbol
get_name()	ModelQuantity name
get_regularization_symbol()	**rtype** sympy.core.symbol.Symbol
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*, *event*, *measurement_symbol=None*, *transformation='lin'*)

Create a new EventObservable instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – See *Observable.__init__()*.
- **name** (str) – See *Observable.__init__()*.
- **value** (sympy.core.expr.Expr) – See *Observable.__init__()*.
- **transformation** (typing.Optional[*amici.import_utils.ObservableTransformation*]) – See *Observable.__init__()*.
- **event** (sympy.core.symbol.Symbol) – Symbolic identifier of the corresponding event.

get_event()

Get the symbolic identifier of the corresponding event.

Return type sympy.core.symbol.Symbol

Returns symbolic identifier

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_measurement_symbol()

Return type sympy.core.symbol.Symbol

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_regularization_symbol()

Return type sympy.core.symbol.Symbol

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Expression

class amici.ode_model.**Expression**(*identifier*, *name*, *value*)

An Expression is a recurring elements in symbolic formulas. Specifying this may yield more compact expression which may lead to substantially shorter model compilation times, but may also reduce model simulation time. Abbreviated by `w`.

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Expression
- **name** (str) – individual name of the Expression (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Expression
- **name** (str) – individual name of the Expression (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.LogLikelihood

class amici.ode_model.**LogLikelihood**(*identifier*, *name*, *value*)

A LogLikelihood defines the distance between measurements and experiments for a particular observable. The final LogLikelihood value in the simulation will be the sum of all specified LogLikelihood instances evaluated at all timepoints, abbreviated by `Jy`.

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.LogLikelihoodRZ

class amici.ode_model.**LogLikelihoodRZ**(*identifier*, *name*, *value*)

Loglikelihood for event observables regularization

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.LogLikelihoodY

class amici.ode_model.**LogLikelihoodY**(*identifier*, *name*, *value*)

Loglikelihood for observables

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.LogLikelihoodZ

class amici.ode_model.**LogLikelihoodZ**(*identifier*, *name*, *value*)

Loglikelihood for event observables

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the LogLikelihood
- **name** (str) – individual name of the LogLikelihood (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.ModelQuantity

class amici.ode_model.**ModelQuantity**(*identifier*, *name*, *value*)

Base class for model components

__init__(*identifier*, *name*, *value*)

Create a new ModelQuantity instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the quantity
- **name** (str) – individual name of the quantity (does not need to be unique)
- **value** (typing.Union[typing.SupportsFloat, numbers.Number, sympy.core.expr.Expr]) – either formula, numeric value or initial value

Methods Summary

__init__(identifier, name, value)	Create a new ModelQuantity instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new ModelQuantity instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the quantity
- **name** (str) – individual name of the quantity (does not need to be unique)
- **value** (typing.Union[typing.SupportsFloat, numbers.Number, sympy.core.expr.Expr]) – either formula, numeric value or initial value

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Observable

class amici.ode_model.**Observable**(*identifier*, *name*, *value*, *measurement_symbol=None*, *transformation=ObservableTransformation.LIN*)

An Observable links model simulations to experimental measurements, abbreviated by `y`.

Variables

- **_measurement_symbol** – sympy symbol used in the objective function to represent measurements to this observable
- **trafo** – observable transformation, only applies when evaluating objective function or residuals

__init__(*identifier*, *name*, *value*, *measurement_symbol=None*, *transformation=ObservableTransformation.LIN*)

Create a new Observable instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Observable
- **name** (str) – individual name of the Observable (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula
- **transformation** (typing.Optional[*amici.import_utils.ObservableTransformation*]) – observable transformation, only applies when evaluating objective function or residuals

Methods Summary

__init__(identifier, name, value[, ...])	Create a new Observable instance.
get_id()	ModelQuantity identifier
get_measurement_symbol()	**rtype** `sympy.core.symbol.Symbol`
get_name()	ModelQuantity name
get_regularization_symbol()	**rtype** `sympy.core.symbol.Symbol`
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*, *measurement_symbol=None*, *transformation=ObservableTransformation.LIN*)

Create a new Observable instance.

Parameters

- **identifier** (`sympy.core.symbol.Symbol`) – unique identifier of the Observable
- **name** (`str`) – individual name of the Observable (does not need to be unique)
- **value** (`sympy.core.expr.Expr`) – formula
- **transformation** (`typing.Optional`[*amici.import_utils.ObservableTransformation*]) – observable transformation, only applies when evaluating objective function or residuals

get_id()

ModelQuantity identifier

Return type `sympy.core.symbol.Symbol`

Returns identifier of the ModelQuantity

get_measurement_symbol()

Return type `sympy.core.symbol.Symbol`

get_name()

ModelQuantity name

Return type `str`

Returns name of the ModelQuantity

get_regularization_symbol()

Return type `sympy.core.symbol.Symbol`

get_val()

ModelQuantity value

Return type `sympy.core.expr.Expr`

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Parameter

class amici.ode_model.**Parameter**(*identifier*, *name*, *value*)

A Parameter is a free variable in the model with respect to which sensitivities may be computed, abbreviated by p.

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Parameter
- **name** (str) – individual name of the Parameter (does not need to be unique)
- **value** (numbers.Number) – numeric value

Methods Summary

__init__(identifier, name, value)	Create a new Expression instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Expression instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Parameter
- **name** (str) – individual name of the Parameter (does not need to be unique)
- **value** (numbers.Number) – numeric value

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.Sigma

class amici.ode_model.**Sigma**(*identifier*, *name*, *value*)

A Standard Deviation Sigma rescales the distance between simulations and measurements when computing residuals or objective functions, abbreviated by `sigma{y,z}`.

__init__(*identifier*, *name*, *value*)

Create a new Standard Deviation instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Standard Deviation
- **name** (str) – individual name of the Standard Deviation (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Standard Deviation instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Standard Deviation instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Standard Deviation
- **name** (str) – individual name of the Standard Deviation (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

`get_name()`

ModelQuantity name

Return type `str`

Returns name of the ModelQuantity

`get_val()`

ModelQuantity value

Return type `sympy.core.expr.Expr`

Returns value of the ModelQuantity

`set_val(val)`

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.SigmaY

`class amici.ode_model.SigmaY(identifier, name, value)`

Standard deviation for observables

`__init__(identifier, name, value)`

Create a new Standard Deviation instance.

Parameters

- **identifier** (`sympy.core.symbol.Symbol`) – unique identifier of the Standard Deviation
- **name** (`str`) – individual name of the Standard Deviation (does not need to be unique)
- **value** (`sympy.core.expr.Expr`) – formula

Methods Summary

`__init__`(identifier, name, value)	Create a new Standard Deviation instance.
`get_id`()	ModelQuantity identifier
`get_name`()	ModelQuantity name
`get_val`()	ModelQuantity value
`set_val`(val)	Set ModelQuantity value

Methods

`__init__(identifier, name, value)`

Create a new Standard Deviation instance.

Parameters

- **identifier** (`sympy.core.symbol.Symbol`) – unique identifier of the Standard Deviation
- **name** (`str`) – individual name of the Standard Deviation (does not need to be unique)

- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.SigmaZ

class amici.ode_model.**SigmaZ**(*identifier*, *name*, *value*)

Standard deviation for event observables

__init__(*identifier*, *name*, *value*)

Create a new Standard Deviation instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Standard Deviation
- **name** (str) – individual name of the Standard Deviation (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

Methods Summary

__init__(identifier, name, value)	Create a new Standard Deviation instance.
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *value*)

Create a new Standard Deviation instance.

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the Standard Deviation
- **name** (str) – individual name of the Standard Deviation (does not need to be unique)
- **value** (sympy.core.expr.Expr) – formula

get_id()

ModelQuantity identifier

Return type sympy.core.symbol.Symbol

Returns identifier of the ModelQuantity

get_name()

ModelQuantity name

Return type str

Returns name of the ModelQuantity

get_val()

ModelQuantity value

Return type sympy.core.expr.Expr

Returns value of the ModelQuantity

set_val(*val*)

Set ModelQuantity value

Returns value of the ModelQuantity

amici.ode_model.State

class amici.ode_model.**State**(*identifier*, *name*, *init*, *dt*)

A State variable defines an entity that evolves with time according to the provided time derivative, abbreviated by `x`.

Variables

- **_conservation_law** – algebraic formula that allows computation of this state according to a conservation law
- **_dt** – algebraic formula that defines the temporal derivative of this state

__init__(*identifier*, *name*, *init*, *dt*)

Create a new State instance. Extends *ModelQuantity.__init__()* by `dt`

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the state
- **name** (str) – individual name of the state (does not need to be unique)
- **init** (sympy.core.expr.Expr) – initial value

- **dt** (sympy.core.expr.Expr) – time derivative

Methods Summary

__init__(identifier, name, init, dt)	Create a new State instance.
get_dt()	Gets the time derivative
get_dx_rdata_dx_solver(state_id)	Returns the expression that allows computation of dx_rdata_dx_solver for this state, accounting for conservation laws.
get_free_symbols()	Gets the set of free symbols in time derivative and initial conditions
get_id()	ModelQuantity identifier
get_name()	ModelQuantity name
get_val()	ModelQuantity value
get_x_rdata()	Returns the expression that allows computation of x_rdata for this state, accounting for conservation laws.
has_conservation_law()	Checks whether this state has a conservation law assigned.
set_conservation_law(law)	Sets the conservation law of a state.
set_dt(dt)	Sets the time derivative
set_val(val)	Set ModelQuantity value

Methods

__init__(*identifier*, *name*, *init*, *dt*)

Create a new State instance. Extends *ModelQuantity.__init__()* by dt

Parameters

- **identifier** (sympy.core.symbol.Symbol) – unique identifier of the state
- **name** (str) – individual name of the state (does not need to be unique)
- **init** (sympy.core.expr.Expr) – initial value
- **dt** (sympy.core.expr.Expr) – time derivative

get_dt()

Gets the time derivative

Return type sympy.core.expr.Expr

Returns time derivative

get_dx_rdata_dx_solver(*state_id*)

Returns the expression that allows computation of dx_rdata_dx_solver for this state, accounting for conservation laws.

Returns dx_rdata_dx_solver expression

get_free_symbols()

Gets the set of free symbols in time derivative and initial conditions

Return type typing.Set[sympy.core.basic.Basic]

Returns free symbols

`get_id()`

ModelQuantity identifier

Return type `sympy.core.symbol.Symbol`

Returns identifier of the ModelQuantity

`get_name()`

ModelQuantity name

Return type `str`

Returns name of the ModelQuantity

`get_val()`

ModelQuantity value

Return type `sympy.core.expr.Expr`

Returns value of the ModelQuantity

`get_x_rdata()`

Returns the expression that allows computation of x_rdata for this state, accounting for conservation laws.

Returns x_rdata expression

`has_conservation_law()`

Checks whether this state has a conservation law assigned.

Returns True if assigned, False otherwise

`set_conservation_law(`*law*`)`

Sets the conservation law of a state.

If a conservation law is set, the respective state will be replaced by an algebraic formula according to the respective conservation law.

Parameters **law** (*amici.ode_model.ConservationLaw*) – linear sum of states that if added to this state remain constant over time

Return type `None`

`set_dt(`*dt*`)`

Sets the time derivative

Parameters **dt** (`sympy.core.expr.Expr`) – time derivative

Return type `None`

`set_val(`*val*`)`

Set ModelQuantity value

Returns value of the ModelQuantity

10.4.13 amici.plotting

Plotting

Plotting related functions

Functions Summary

plotObservableTrajectories(rdata[, ...])	Plot observable trajectories
plotStateTrajectories(rdata[, ...])	Plot state trajectories
plot_jacobian(rdata)	Plot Jacobian as heatmap.
plot_observable_trajectories(rdata[, ...])	Plot observable trajectories
plot_state_trajectories(rdata[, ...])	Plot state trajectories

Functions

amici.plotting.**plotObservableTrajectories**(*rdata*, *observable_indices=None*, *ax=None*, *model=None*)

Plot observable trajectories

Parameters

- **rdata** (amici.numpy.ReturnDataView) – AMICI simulation results as returned by *amici.amici.runAmiciSimulation()*
- **observable_indices** (typing.Optional[typing.Iterable[int]]) – Indices of observables for which trajectories are to be plotted
- **ax** (typing.Optional[matplotlib.axes._axes.Axes]) – matplotlib Axes instance to plot into
- **model** (typing.Optional[*amici.amici.Model*]) – amici model instance

Return type None

amici.plotting.**plotStateTrajectories**(*rdata*, *state_indices=None*, *ax=None*, *model=None*)

Plot state trajectories

Parameters

- **rdata** (amici.numpy.ReturnDataView) – AMICI simulation results as returned by *amici.amici.runAmiciSimulation()*
- **state_indices** (typing.Optional[typing.Iterable[int]]) – Indices of states for which trajectories are to be plotted
- **ax** (typing.Optional[matplotlib.axes._axes.Axes]) – matplotlib Axes instance to plot into
- **model** (typing.Optional[*amici.amici.Model*]) – amici model instance

Return type None

amici.plotting.**plot_jacobian**(*rdata*)

Plot Jacobian as heatmap.

`amici.plotting.`**`plot_observable_trajectories`**(*rdata*, *observable_indices=None*, *ax=None*, *model=None*)

Plot observable trajectories

Parameters

- **rdata** (`amici.numpy.ReturnDataView`) – AMICI simulation results as returned by *`amici.amici.runAmiciSimulation()`*
- **observable_indices** (`typing.Optional[typing.Iterable[int]]`) – Indices of observables for which trajectories are to be plotted
- **ax** (`typing.Optional[matplotlib.axes._axes.Axes]`) – matplotlib Axes instance to plot into
- **model** (`typing.Optional[`*`amici.amici.Model`*`]`) – amici model instance

Return type `None`

`amici.plotting.`**`plot_state_trajectories`**(*rdata*, *state_indices=None*, *ax=None*, *model=None*)

Plot state trajectories

Parameters

- **rdata** (`amici.numpy.ReturnDataView`) – AMICI simulation results as returned by *`amici.amici.runAmiciSimulation()`*
- **state_indices** (`typing.Optional[typing.Iterable[int]]`) – Indices of states for which trajectories are to be plotted
- **ax** (`typing.Optional[matplotlib.axes._axes.Axes]`) – matplotlib Axes instance to plot into
- **model** (`typing.Optional[`*`amici.amici.Model`*`]`) – amici model instance

Return type `None`

10.4.14 amici.pandas

Pandas Wrappers

This module contains convenience wrappers that allow for easy interconversion between C++ objects from *`amici.amici`* and pandas DataFrames

Functions Summary

constructEdataFromDataFrame(df, model, condition)	Constructs an ExpData instance according to the provided Model and DataFrame.
getDataObservablesAsDataFrame(model, edata_list)	Write Observables from experimental data as DataFrame.
getEdataFromDataFrame(model, df[, by_id])	Constructs a ExpData instances according to the provided Model and DataFrame.
getResidualsAsDataFrame(model, edata_list, ...)	Convert a list of ReturnData and ExpData to pandas DataFrame with residuals.
getSimulationObservablesAsDataFrame(model, ...)	Write Observables from simulation results as DataFrame.
getSimulationStatesAsDataFrame(model, ...[, ...])	Get model state according to lists of ReturnData and ExpData.
get_expressions_as_dataframe(model, ...[, by_id])	Get values of model expressions from lists of ReturnData as DataFrame.

Functions

amici.pandas.**constructEdataFromDataFrame**(*df*, *model*, *condition*, *by_id=False*)

Constructs an ExpData instance according to the provided Model and DataFrame.

Parameters

- **df** (pandas.core.frame.DataFrame) – pd.DataFrame with Observable Names/Ids as columns. Standard deviations may be specified by appending '_std' as suffix.
- **model** (typing.Union[*amici.amici.ModelPtr*, *amici.amici.Model*]) – Model instance.
- **condition** (pandas.core.series.Series) – pd.Series with FixedParameter Names/Ids as columns. Preequilibration conditions may be specified by appending '_preeq' as suffix. Presimulation conditions may be specified by appending '_presim' as suffix.
- **by_id** (typing.Optional[bool]) – Indicate whether in the arguments, column headers are based on ids or names. This should correspond to the way *df* and *condition* was created in the first place.

Return type *amici.amici.ExpData*

Returns ExpData instance.

amici.pandas.**getDataObservablesAsDataFrame**(*model*, *edata_list*, *by_id=False*)

Write Observables from experimental data as DataFrame.

Parameters

- **model** (typing.Union[*amici.amici.ModelPtr*, *amici.amici.Model*]) – Model instance.
- **edata_list** (typing.Union[typing.List[*amici.amici.ExpData*], typing.List[*amici.amici.ExpDataPtr*], *amici.amici.ExpData*, *amici.amici.ExpDataPtr*]) – list of ExpData instances with experimental data. May also be a single ExpData instance.
- **by_id** (typing.Optional[bool]) – If True, uses observable ids as column names in the generated DataFrame, otherwise the possibly more descriptive observable names are used.

Return type `pandas.core.frame.DataFrame`

Returns pandas DataFrame with conditions/timepoints as rows and observables as columns.

amici.pandas.**getEdataFromDataFrame**(*model*, *df*, *by_id=False*)

Constructs a ExpData instances according to the provided Model and DataFrame.

Parameters

- **df** (`pandas.core.frame.DataFrame`) – dataframe with Observable Names/Ids, FixedParameter Names/Ids and time as columns. Standard deviations may be specified by appending '_std' as suffix. Preequilibration fixedParameters may be specified by appending '_preeq' as suffix. Presimulation fixedParameters may be specified by appending '_presim' as suffix. Presimulation time may be specified as 't_presim' column.
- **model** (`typing.Union`[*amici.amici.ModelPtr*, *amici.amici.Model*]) – Model instance.
- **by_id** (`typing.Optional`[`bool`]) – Whether the column names in *df* are based on ids or names, corresponding to how the dataframe was created in the first place.

Return type `typing.List`[*amici.amici.ExpData*]

Returns list of ExpData instances.

amici.pandas.**getResidualsAsDataFrame**(*model*, *edata_list*, *rdata_list*, *by_id=False*)

Convert a list of ReturnData and ExpData to pandas DataFrame with residuals.

Parameters

- **model** (*amici.amici.Model*) – Model instance.
- **edata_list** (`typing.Union`[`typing.List`[*amici.amici.ExpData*], `typing.List`[*amici.amici.ExpDataPtr*], *amici.amici.ExpData*, *amici.amici.ExpDataPtr*]) – list of ExpData instances with experimental data. May also be a single ExpData instance.
- **rdata_list** (`typing.Union`[`typing.List`[`amici.numpy.ReturnDataView`], `amici.numpy.ReturnDataView`]) – list of ReturnData instances corresponding to ExpData. May also be a single ReturnData instance.
- **by_id** (`typing.Optional`[`bool`]) – bool, optional (default = False) If True, ids are used as identifiers, otherwise the possibly more descriptive names.

Return type `pandas.core.frame.DataFrame`

Returns pandas DataFrame with conditions and residuals.

amici.pandas.**getSimulationObservablesAsDataFrame**(*model*, *edata_list*, *rdata_list*, *by_id=False*)

Write Observables from simulation results as DataFrame.

Parameters

- **model** (*amici.amici.Model*) – Model instance.
- **edata_list** (`typing.Union`[`typing.List`[*amici.amici.ExpData*], `typing.List`[*amici.amici.ExpDataPtr*], *amici.amici.ExpData*, *amici.amici.ExpDataPtr*]) – list of ExpData instances with experimental data. May also be a single ExpData instance.
- **rdata_list** (`typing.Union`[`typing.List`[`amici.numpy.ReturnDataView`], `amici.numpy.ReturnDataView`]) – list of ReturnData instances corresponding to ExpData. May also be a single ReturnData instance.

- **by_id** (typing.Optional[bool]) – If True, ids are used as identifiers, otherwise the possibly more descriptive names.

Return type pandas.core.frame.DataFrame

Returns pandas DataFrame with conditions/timepoints as rows and observables as columns.

amici.pandas.**getSimulationStatesAsDataFrame**(*model*, *edata_list*, *rdata_list*, *by_id=False*)

Get model state according to lists of ReturnData and ExpData.

Parameters

- **model** (*amici.amici.Model*) – Model instance.
- **edata_list** (typing.Union[typing.List[*amici.amici.ExpData*], typing.List[*amici.amici.ExpDataPtr*], *amici.amici.ExpData*, *amici.amici.ExpDataPtr*]) – list of ExpData instances with experimental data. May also be a single ExpData instance.
- **rdata_list** (typing.Union[typing.List[amici.numpy.ReturnDataView], amici.numpy.ReturnDataView]) – list of ReturnData instances corresponding to ExpData. May also be a single ReturnData instance.
- **by_id** (typing.Optional[bool]) – If True, ids are used as identifiers, otherwise the possibly more descriptive names.

Return type pandas.core.frame.DataFrame

Returns pandas DataFrame with conditions/timepoints as rows and state variables as columns.

amici.pandas.**get_expressions_as_dataframe**(*model*, *edata_list*, *rdata_list*, *by_id=False*)

Get values of model expressions from lists of ReturnData as DataFrame.

Parameters

- **model** (*amici.amici.Model*) – Model instance.
- **edata_list** (typing.Union[typing.List[*amici.amici.ExpData*], typing.List[*amici.amici.ExpDataPtr*], *amici.amici.ExpData*, *amici.amici.ExpDataPtr*]) – list of ExpData instances with experimental data. May also be a single ExpData instance.
- **rdata_list** (typing.Union[typing.List[amici.numpy.ReturnDataView], amici.numpy.ReturnDataView]) – list of ReturnData instances corresponding to ExpData. May also be a single ReturnData instance.
- **by_id** (typing.Optional[bool]) – If True, ids are used as identifiers, otherwise the possibly more descriptive names.

Return type pandas.core.frame.DataFrame

Returns pandas DataFrame with conditions/timepoints as rows and model expressions as columns.

10.4.15 amici.logging

Logging

This module provides custom logging functionality for other amici modules

Functions Summary

get_logger([logger_name, log_level])	Returns (if extistant) or creates an AMICI logger
log_execution_time(description, logger)	Parameterized function decorator that enables automatic execution time tracking
set_log_level(logger, log_level)	**rtype** None

Functions

amici.logging.**get_logger**(*logger_name='amici'*, *log_level=None*, ***kwargs*)

Returns (if extistant) or creates an AMICI logger

If the AMICI base logger has already been set up, this method will return it or any of its descendant loggers without overriding the settings - i.e. any values supplied as kwargs will be ignored.

Parameters

- **logger_name** (typing.Optional[str]) – Get a logger for a specific namespace, typically __name__ for code outside of classes or self.__module__ inside a class
- **log_level** (typing.Optional[int]) – Override the default or preset log level for the requested logger. None or False uses the default or preset value. True evaluates to logging.DEBUG. Any integer is used directly.
- **console_output** – Set up a default console log handler if True (default). Only used when the AMICI logger hasn't been set up yet.
- **file_output** – Supply a filename to copy all log output to that file, or set to False to disable (default). Only used when the AMICI logger hasn't been set up yet.
- **capture_warnings** – Capture warnings from Python's warnings module if True (default). Only used when the AMICI logger hasn't been set up yet..

Return type logging.Logger

Returns A logging.Logger object with the requested name

amici.logging.**log_execution_time**(*description*, *logger*)

Parameterized function decorator that enables automatic execution time tracking

Parameters

- **description** (str) – Description of what the decorated function does
- **logger** (logging.Logger) – Logger to which execution timing will be printed

Return type typing.Callable

amici.logging.**set_log_level**(*logger*, *log_level*)

Return type None

10.4.16 amici.gradient_check

Finite Difference Check

This module provides functions to automatically check correctness of amici computed sensitivities using finite difference approximations

Functions Summary

check_derivatives(model, solver[, edata, ...])	Finite differences check for likelihood gradient.
check_finite_difference(x0, model, solver, ...)	Checks the computed sensitivity based derivatives against a finite difference approximation.

Functions

amici.gradient_check.**check_derivatives**(*model*, *solver*, *edata=None*, *atol=0.0001*, *rtol=0.0001*, *epsilon=0.001*, *check_least_squares=True*, *skip_zero_pars=False*)

Finite differences check for likelihood gradient.

Parameters

- **model** (*amici.amici.Model*) – amici model
- **solver** (*amici.amici.Solver*) – amici solver
- **edata** (typing.Optional[amici.swig_wrappers.ExpData]) – exp data
- **atol** (typing.Optional[float]) – absolute tolerance for comparison
- **rtol** (typing.Optional[float]) – relative tolerance for comparison
- **epsilon** (typing.Optional[float]) – finite difference step-size
- **check_least_squares** (bool) – whether to check least squares related values.
- **skip_zero_pars** (bool) – whether to perform FD checks for parameters that are zero

Return type None

amici.gradient_check.**check_finite_difference**(*x0*, *model*, *solver*, *edata*, *ip*, *fields*, *atol=0.0001*, *rtol=0.0001*, *epsilon=0.001*)

Checks the computed sensitivity based derivatives against a finite difference approximation.

Parameters

- **x0** (typing.Sequence[float]) – parameter value at which to check finite difference approximation
- **model** (*amici.amici.Model*) – amici model
- **solver** (*amici.amici.Solver*) – amici solver
- **edata** (amici.swig_wrappers.ExpData) – exp data
- **ip** (int) – parameter index
- **fields** (typing.List[str]) – rdata fields for which to check the gradient
- **atol** (typing.Optional[float]) – absolute tolerance for comparison

- **rtol** (typing.Optional[float]) – relative tolerance for comparison
- **epsilon** (typing.Optional[float]) – finite difference step-size

Return type None

10.4.17 amici.parameter_mapping

Parameter mapping

When performing parameter inference, often parameters need to be mapped from simulation to estimation parameters, and parameters can differ between conditions. This can be handled using the *ParameterMapping*.

Note: While the parameter mapping can be used directly with AMICI, it was developed for usage together with PEtab, for which the whole workflow of generating the mapping is automatized.

Classes

ParameterMapping([parameter_mappings])	Parameter mapping for multiple conditions.
ParameterMappingForCondition([map_sim_var, ...])	Parameter mapping for condition.

amici.parameter_mapping.ParameterMapping

class amici.parameter_mapping.**ParameterMapping**(*parameter_mappings=None*)

Parameter mapping for multiple conditions.

This can be used like a list of *ParameterMappingForCondition*s.

Parameters **parameter_mappings** (typing.Optional[typing.List[*amici.parameter_mapping.ParameterMappingForCondition*]]) – List of parameter mappings for specific conditions.

__init__(*parameter_mappings=None*)

Methods Summary

__init__([parameter_mappings])	
append(parameter_mapping_for_condition)	Append a condition specific parameter mapping.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

free_symbols	Get IDs of all (symbolic) parameters present in this mapping

Methods

__init__(*parameter_mappings=None*)

append(*parameter_mapping_for_condition*)

Append a condition specific parameter mapping.

count(*value*) → integer -- return number of occurrences of value

index(*value*[, *start*[, *stop*]]) → integer -- return first index of value.

Raises ValueError if the value is not present.

Supporting start and stop arguments is optional, but recommended.

amici.parameter_mapping.ParameterMappingForCondition

class amici.parameter_mapping.**ParameterMappingForCondition**(*map_sim_var=None*, *scale_map_sim_var=None*, *map_preeq_fix=None*, *scale_map_preeq_fix=None*, *map_sim_fix=None*, *scale_map_sim_fix=None*)

Parameter mapping for condition.

Contains mappings for free parameters, fixed parameters, and fixed preequilibration parameters, both for parameters and scales.

In the scale mappings, for each simulation parameter the scale on which the value is passed (and potentially gradients are to be returned) is given. In the parameter mappings, for each simulation parameter a corresponding optimization parameter (or a numeric value) is given.

If a mapping is not passed, the parameter mappings are assumed to be empty, and if a scale mapping is not passed, all scales are set to linear.

Parameters

- **map_sim_var** (typing.Optional[typing.Dict[str, typing.Union[str, numbers.Number]]]) – Mapping for free simulation parameters.
- **scale_map_sim_var** (typing.Optional[typing.Dict[str, str]]) – Scales for free simulation parameters.
- **map_preeq_fix** (typing.Optional[typing.Dict[str, typing.Union[str, numbers.Number]]]) – Mapping for fixed preequilibration parameters.
- **scale_map_preeq_fix** (typing.Optional[typing.Dict[str, str]]) – Scales for fixed preequilibration parameters.
- **map_sim_fix** (typing.Optional[typing.Dict[str, typing.Union[str, numbers.Number]]]) – Mapping for fixed simulation parameters.

- **scale_map_sim_fix** (typing.Optional[typing.Dict[str, str]]) – Scales for fixed simulation parameters.

__init__(*map_sim_var=None*, *scale_map_sim_var=None*, *map_preeq_fix=None*, *scale_map_preeq_fix=None*, *map_sim_fix=None*, *scale_map_sim_fix=None*)

Methods Summary

__init__([map_sim_var, scale_map_sim_var, ...])	

Attributes

free_symbols	Get IDs of all (symbolic) parameters present in this mapping

Methods

__init__(*map_sim_var=None*, *scale_map_sim_var=None*, *map_preeq_fix=None*, *scale_map_preeq_fix=None*, *map_sim_fix=None*, *scale_map_sim_fix=None*)

Functions Summary

amici_to_petab_scale(amici_scale)	Convert amici scale id to petab scale id.
fill_in_parameters(edatas, ...)	Fill fixed and dynamic parameters into the edatas (in-place).
fill_in_parameters_for_condition(edata, ...)	Fill fixed and dynamic parameters into the edata for condition (in-place).
petab_to_amici_scale(petab_scale)	Convert petab scale id to amici scale id.
scale_parameter(value, petab_scale)	Bring parameter from linear scale to target scale.
scale_parameters_dict(value_dict, ...)	Bring parameters from linear scale to target scale.
unscale_parameter(value, petab_scale)	Bring parameter from scale to linear scale.
unscale_parameters_dict(value_dict, ...)	Bring parameters from target scale to linear scale.

Functions

amici.parameter_mapping.**amici_to_petab_scale**(*amici_scale*)

Convert amici scale id to petab scale id.

Return type str

amici.parameter_mapping.**fill_in_parameters**(*edatas*, *problem_parameters*, *scaled_parameters*, *parameter_mapping*, *amici_model*)

Fill fixed and dynamic parameters into the edatas (in-place).

Parameters

- **edatas** (typing.List[amici.swig_wrappers.ExpData]) – List of experimental datas *amici.amici.ExpData* with everything except parameters filled.

- **problem_parameters** (typing.Dict[str, numbers.Number]) – Problem parameters as parameterId=>value dict. Only parameters included here will be set. Remaining parameters will be used as currently set in *amici_model*.
- **scaled_parameters** (bool) – If True, problem_parameters are assumed to be on the scale provided in the parameter mapping. If False, they are assumed to be in linear scale.
- **parameter_mapping** (*amici.parameter_mapping.ParameterMapping*) – Parameter mapping for all conditions.
- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model.

Return type None

amici.parameter_mapping.**fill_in_parameters_for_condition**(*edata*, *problem_parameters*, *scaled_parameters*, *parameter_mapping*, *amici_model*)

Fill fixed and dynamic parameters into the edata for condition (in-place).

Parameters

- **edata** (amici.swig_wrappers.ExpData) – Experimental data object to fill parameters into.
- **problem_parameters** (typing.Dict[str, numbers.Number]) – Problem parameters as parameterId=>value dict. Only parameters included here will be set. Remaining parameters will be used as already set in *amici_model* and *edata*.
- **scaled_parameters** (bool) – If True, problem_parameters are assumed to be on the scale provided in the parameter mapping. If False, they are assumed to be in linear scale.
- **parameter_mapping** (*amici.parameter_mapping.ParameterMappingForCondition*) – Parameter mapping for current condition.
- **amici_model** (typing.Union[*amici.amici.Model*, *amici.amici.ModelPtr*]) – AMICI model

Return type None

amici.parameter_mapping.**petab_to_amici_scale**(*petab_scale*)

Convert petab scale id to amici scale id.

Return type int

amici.parameter_mapping.**scale_parameter**(*value*, *petab_scale*)

Bring parameter from linear scale to target scale.

Parameters

- **value** (numbers.Number) – Value to scale
- **petab_scale** (str) – Target scale of `value`

Return type numbers.Number

Returns `value` on target scale

amici.parameter_mapping.**scale_parameters_dict**(*value_dict*, *petab_scale_dict*)

Bring parameters from linear scale to target scale.

Bring values in `value_dict` from linear scale to the scale provided in `petab_scale_dict` (in-place). Both arguments are expected to have the same length and matching keys.

Parameters

- **value_dict** (typing.Dict[typing.Any, numbers.Number]) – Values to scale
- **petab_scale_dict** (typing.Dict[typing.Any, str]) – Target scales of values

Return type None

amici.parameter_mapping.**unscale_parameter**(*value*, *petab_scale*)

Bring parameter from scale to linear scale.

Parameters

- **value** (numbers.Number) – Value to scale
- **petab_scale** (str) – Target scale of value

Return type numbers.Number

Returns value on linear scale

amici.parameter_mapping.**unscale_parameters_dict**(*value_dict*, *petab_scale_dict*)

Bring parameters from target scale to linear scale.

Bring values in value_dict from linear scale to the scale provided in petab_scale_dict (in-place). Both arguments are expected to have the same length and matching keys.

Parameters

- **value_dict** (typing.Dict[typing.Any, numbers.Number]) – Values to scale
- **petab_scale_dict** (typing.Dict[typing.Any, str]) – Target scales of values

Return type None

10.4.18 amici.conserved_quantities_demartino

Functions Summary

compute_moiety_conservation_laws(...[, ...])	Compute moiety conservation laws.

Functions

amici.conserved_quantities_demartino.**compute_moiety_conservation_laws**(*stoichiometric_list*, *num_species*, *num_reactions*, *max_num_monte_carlo=20*, *rng_seed=False*, *species_names=None*)

Compute moiety conservation laws.

According to the algorithm proposed by De Martino et al. (2014) https://doi.org/10.1371/journal.pone.0100750

Parameters

- **stoichiometric_list** (typing.Sequence[float]) – the stoichiometric matrix as a list (species x reactions, column-major ordering)
- **num_species** (int) – total number of species in the reaction network

- **num_reactions** (int) – total number of reactions in the reaction network
- **max_num_monte_carlo** (int) – maximum number of MonteCarlo steps before changing to relaxation
- **rng_seed** (typing.Union[None, bool, int]) – Seed for the random number generator. If *False*, the RNG will not be re-initialized. Other values will be passed to random.seed().
- **species_names** (typing.Optional[typing.Sequence[str]]) – Species names. Optional and only used for logging.

Return type typing.Tuple[typing.List[typing.List[int]], typing.List[typing.List[float]]]

Returns Integer MCLs as list of lists of indices of involved species and list of lists of corresponding coefficients.

10.4.19 amici.conserved_quantities_rref

Find conserved quantities deterministically

Functions Summary

nullspace_by_rref(mat)	Compute basis of the nullspace of mat based on the reduced row echelon form
pivots(mat)	Get indices of pivot columns in mat, assumed to be in reduced row echelon form
rref(mat[, round_ndigits])	Bring matrix mat to reduced row echelon form

Functions

amici.conserved_quantities_rref.**nullspace_by_rref**(*mat*)

Compute basis of the nullspace of mat based on the reduced row echelon form

Return type numpy.array

amici.conserved_quantities_rref.**pivots**(*mat*)

Get indices of pivot columns in mat, assumed to be in reduced row echelon form

Return type typing.List[int]

amici.conserved_quantities_rref.**rref**(*mat*, *round_ndigits=None*)

Bring matrix mat to reduced row echelon form

see https://en.wikipedia.org/wiki/Row_echelon_form

Parameters

- **mat** (numpy.array) – Numpy float matrix to operate on (will be copied)
- **round_ndigits** (typing.Union[typing.Literal[False], int, None]) – Number of digits to round intermediary results to, or False to disable rounding completely. Helps to avoid numerical artifacts.

Return type numpy.array

Returns mat in rref form.

CHAPTER

ELEVEN

C++ INTERFACE

11.1 Building the C++ library

The following section describes building the AMICI C++ library:

Note: The AMICI C++ interface only supports simulation of models imported using the *Python interface* and *Matlab interface*. It cannot be used for model import itself.

Prerequisites:

- CBLAS compatible BLAS library
- HDF5 libraries (currently mandatory, see https://github.com/AMICI-dev/AMICI/issues/1252)
- a C++14 compatible compiler
- a C compiler
- Optional: boost for serialization

To use AMICI from C++, run the

```
./scripts/buildSuiteSparse.sh
./scripts/buildSundials.sh
./scripts/buildAmici.sh
```

script to build the AMICI library.

Note: On some systems, the CMake executable may be named something other than `cmake`. In this case, set the `CMAKE` environment variable to the correct name (e.g. `export CMAKE=cmake3`, in case you have CMake available as `cmake3`).

The static library can then be linked from

```
./build/libamici.a
```

In CMake-based packages, amici can be linked via

```
find_package(Amici)
```

For further usage, consult the AMICI *C++ interface documentation*.

11.1.1 Supported CBLAS libraries

The C++ interfaces require a system installation of a CBLAS-compatible *Basic Linear Algebra Subprograms* (BLAS) library. AMICI has been tested with various implementations such as Accelerate, Intel MKL, cblas, openblas and atlas.

11.1.2 Optional SuperLU_MT support

To build AMICI with SuperLU_MT support, run

```
./scripts/buildSuperLUMT.sh
./scripts/buildSundials.sh
cd build/
cmake -DSUNDIALS_SUPERLUMT_ENABLE=ON ..
make
```

11.2 Using AMICI's C++ interface

The various import functions in of the *Python interface* and *Matlab interface* translate models defined in different formats into C++ code. These generated model libraries, together with the AMICI base library can be used in any C++ application for model simulation and sensitivity analysis. This section will give a short overview over the generated files and provide a brief introduction of how this code can be included in other applications. Further details are available in the *C++ API reference*.

11.2.1 AMICI-generated C++ model files

After importing a model using either the *Python interface* or the *Matlab interface*, the specified output directory contains (among others) C++ code for the various model functions.

The content of a model source directory looks something like this (given *MODEL_NAME=model_steadystate*):

```
CMakeLists.txt
main.cpp
model_steadystate_deltaqB.cpp
model_steadystate_deltaqB.h
[... many more files model_steadystate_*.(cpp|h|md5|o) ]
wrapfunctions.cpp
wrapfunctions.h
model_steadystate.h
```

These files provide the implementation of a model-specific subclass of `amici::Model`. The `CMakeLists.txt` file can be used to build the model library using CMake. `main.cpp` contains a simple scaffold for running a model simulation from C++. See next section for more details on these files.

11.2.2 Running a model simulation

AMICI's public API is mostly available through amici/amici.h. This is the only header file that needs to be included for basic usage. All functions there are declared within the *amici namespace*. Additionally, amici/hdf5.h and amici/serialization.h may be handy for specific use cases. The former provides some functions for reading and writing HDF5 files, latter for serialization (requires Boost). All model-specific functions are defined in the namespace `model_$modelname`.

The main function for running an AMICI simulation is `amici::runAmiciSimulation()`. This function requires

- an instance of a `amici::Model` subclass as generated during model import. For the example *model_steadystate* the respective class is provided as `Model_model_steadystate` in `model_steadystate.h` in output directory for the given model.
- a `amici::Solver` instance. This solver instance needs to match the requirements of the model and can be obtained from `amici::AbstractModel::getSolver()`.
- optionally an `amici::ExpData` instance, which contains any experimental data (e.g. measurements, noise model parameters or model inputs) to evaluate residuals or an objective function.

This function returns a `amici::ReturnData` object, which contains all simulation results.

For running simulations for multiple experimental conditions (multiple `amici::ExpData` instances), `amici::runAmiciSimulations()` provides an alternative entry point. If AMICI (and your application) have been compiled with OpenMP support (see installation guide), this allows for running those simulations in parallel.

A scaffold for a standalone simulation program is automatically generated during model import in `main.cpp` in the model output directory. This program shows how to use the above-mentioned classes, how to obtain the simulation results, and may provide a starting point for your own simulation code.

Working with multiple or anonymous models

AMICI model import generates a `amici::Model` subclass for the specific model, based on the name used during import. One the one hand, this allows you to use multiple models with different names within a single application. On the other hand, this requires you to know the name of the model, which can be inconvenient in some cases.

When working with a single model, the `wrapfunctions.h` file generated during model import can be used to avoid specifying model names explicitly. It defines a function `amici::generic_model::getModel()`, that returns an instance of the model class by a generic name.

Note: Including multiple `wrapfunctions.h` files from different models in a single application is not possible. When using multiple models, explicit names have to be used or the different model libraries need to be loaded dynamically at runtime.

11.2.3 Compiling and linking

To run AMICI simulations from within your C++ application, you need to compile and link the following libraries:

- model library
- AMICI base library
- SUNDIALS libraries
- SuiteSparse libraries
- CBLAS-compatible BLAS

- optionally HDF5 (C, HL, and CXX components) set CMake option `ENABLE_HDF5` to `OFF` to build without HDF5-support
- optionally OpenMP (for parallel simulation of multiple conditions, see `amici::runAmiciSimulations()`)
- optionally boost (only when using serialization of AMICI object)

The simplest and recommended way is using the provide CMake files which take care of all these dependencies.

Considering the simple case, that you want to simulate one specific model in your CMake-based C++ application, you can copy or move the generated model directory containing the `CMakeLists.txt` file to your application directory, add *add_subdirectory(yourModelDirectory)* to your project's `CMakeLists.txt` file and build your project using CMake as usual.

11.2.4 Parameter estimation for AMICI models in high-performance computing environments

To perform parameter estimation for large or otherwise computationally demanding AMICI models from C++ in a high-performance computing environment, you may find the parPE library helpful. parPE allows for the private or shared memory parallel evaluation of a cost function requiring multiple simulations of the same model with different inputs. It provides interfaces to different optimizers, such as Ipopt.

11.3 AMICI C++ API

AMICI C++ library functions

11.3.1 Class Hierarchy

11.3.2 File Hierarchy

11.3.3 Full API

Namespaces

Namespace amici

Contents

Namespaces

- *Namespace amici::hdf5*

Classes

- *Struct LogItem*
- *Struct ModelDimensions*
- *Struct ModelState*
- *Struct ModelStateDerived*
- *Struct SimulationState*
- *Class AbstractModel*
- *Class AmiException*
- *Class AmiVector*
- *Class AmiVectorArray*
- *Class BackwardProblem*
- *Class ConditionContext*
- *Class ContextManager*
- *Class CvodeException*
- *Class CVodeSolver*
- *Class ExpData*
- *Class FinalStateStorer*
- *Class ForwardProblem*
- *Class IDAException*
- *Class IDASolver*
- *Class IntegrationFailure*
- *Class IntegrationFailureB*
- *Class Logger*
- *Class Model*
- *Class Model_DAE*
- *Class Model_ODE*
- *Class ModelContext*
- *Class NewtonFailure*
- *Class NewtonSolver*
- *Class NewtonSolverDense*
- *Class NewtonSolverSparse*
- *Class ReturnData*

- *Class SetupFailure*
- *Class SimulationParameters*
- *Class Solver*
- *Class SteadystateProblem*
- *Class SUNLinSolBand*
- *Class SUNLinSolDense*
- *Class SUNLinSolKLU*
- *Class SUNLinSolPCG*
- *Class SUNLinSolSPBCGS*
- *Class SUNLinSolSPFGMR*
- *Class SUNLinSolSPGMR*
- *Class SUNLinSolSPTFQMR*
- *Class SUNLinSolWrapper*
- *Class SUNMatrixWrapper*
- *Class SUNNonLinSolFixedPoint*
- *Class SUNNonLinSolNewton*
- *Class SUNNonLinSolWrapper*

Enums

- *Enum BLASLayout*
- *Enum BLASTranspose*
- *Enum FixedParameterContext*
- *Enum InternalSensitivityMethod*
- *Enum InterpolationType*
- *Enum LinearMultistepMethod*
- *Enum LinearSolver*
- *Enum LogSeverity*
- *Enum ModelQuantity*
- *Enum NewtonDampingFactorMode*
- *Enum NonlinearSolverIteration*
- *Enum ObservableScaling*
- *Enum ParameterScaling*
- *Enum RDataReporting*
- *Enum SecondOrderMode*
- *Enum SensitivityMethod*
- *Enum SensitivityOrder*

- Enum SteadyStateContext
- Enum SteadyStateSensitivityMode
- Enum SteadyStateStatus

Functions

- Template Function amici::addSlice(const gsl::span<T const>, gsl::span<T>)
- Template Function amici::addSlice(std::vector<T> const&, gsl::span<T>)
- Function amici::amici_daxpy
- Function amici::amici_dgemm
- Function amici::amici_dgemv
- Function amici::backtraceString
- Template Function amici::checkBufferSize
- Function amici::checkSigmaPositivity(std::vector<realtype> const&, char const *)
- Function amici::checkSigmaPositivity(realtype, char const *)
- Template Function amici::deserializeFromChar
- Template Function amici::deserializeFromString
- Function amici::dotProd
- Function amici::getScaledParameter
- Function amici::getUnscaledParameter
- Template Function amici::is_equal
- Function amici::linearSum
- Function amici::N_VGetArrayPointerConst
- Function amici::operator==(ExpData const&, ExpData const&)
- Function amici::operator==(const Model&, const Model&)
- Function amici::operator==(const ModelDimensions&, const ModelDimensions&)
- Function amici::operator==(const ModelState&, const ModelState&)
- Function amici::operator==(const SimulationParameters&, const SimulationParameters&)
- Function amici::operator==(const Solver&, const Solver&)
- Function amici::printfToString
- Function amici::regexErrorToString
- Function amici::runAmiciSimulation
- Function amici::runAmiciSimulations
- Function amici::scaleParameters
- Template Function amici::serializeToChar
- Template Function amici::serializeToStdVec
- Template Function amici::serializeToString

- *Function amici::simulation_status_to_str*
- *Template Function amici::slice(std::vector<T>&, int, unsigned)*
- *Template Function amici::slice(std::vector<T> const&, int, unsigned)*
- *Function amici::unravel_index(size_t, size_t)*
- *Function amici::unravel_index(sunindextype, SUNMatrix)*
- *Function amici::unscaleParameters*
- *Function amici::wrapErrHandlerFn*
- *Template Function amici::writeSlice(const gsl::span<T const>, gsl::span<T>)*
- *Template Function amici::writeSlice(std::vector<T> const&, std::vector<T>&)*
- *Template Function amici::writeSlice(std::vector<T> const&, gsl::span<T>)*
- *Function amici::writeSlice(AmiVector const&, gsl::span<realtype>)*

Typedefs

- *Typedef amici::const_N_Vector*
- *Typedef amici::realtype*

Variables

- *Variable amici::AMICI_CONV_FAILURE*
- *Variable amici::AMICI_DAMPING_FACTOR_ERROR*
- *Variable amici::AMICI_DATA_RETURN*
- *Variable amici::AMICI_ERR_FAILURE*
- *Variable amici::AMICI_ERROR*
- *Variable amici::AMICI_FIRST_RHSFUNC_ERR*
- *Variable amici::AMICI_ILL_INPUT*
- *Variable amici::AMICI_MAX_TIME_EXCEEDED*
- *Variable amici::AMICI_NO_STEADY_STATE*
- *Variable amici::AMICI_NORMAL*
- *Variable amici::AMICI_NOT_IMPLEMENTED*
- *Variable amici::AMICI_ONE_STEP*
- *Variable amici::AMICI_ONEOUTPUT*
- *Variable amici::AMICI_PREEQUILIBRATE*
- *Variable amici::AMICI_RECOVERABLE_ERROR*
- *Variable amici::AMICI_RHSFUNC_FAIL*
- *Variable amici::AMICI_ROOT_RETURN*
- *Variable amici::AMICI_SINGULAR_JACOBIAN*

Namespace amici::hdf5

Contents

Functions

Namespace boost

Contents

Namespaces

Namespace boost::serialization

Contents

Functions

Namespace gsl

Contents

Functions

- *Function gsl::make_span(N_Vector)*
- *Function gsl::make_span(SUNMatrix)*

Namespace std

STL namespace.

Classes and Structs

Struct LogItem

- Defined in file_include_amici_logging.h

Struct Documentation

struct **LogItem**

A log item.

Public Functions

LogItem() = default

Default ctor.

inline **LogItem**(*LogSeverity* severity, std::string const &identifier, std::string const &message)

Construct a *LogItem*.

Parameters

- **severity** –
- **identifier** –
- **message** –

Public Members

LogSeverity **severity**

Severity level

std::string **identifier**

Short identifier for the logged event

std::string **message**

A more detailed and readable message

Struct ModelDimensions

- Defined in file_include_amici_model_dimensions.h

Inheritance Relationships

Derived Types

- `public amici::Model` (*Class Model*)
- `public amici::ReturnData` (*Class ReturnData*)

Struct Documentation

struct **ModelDimensions**

Container for model dimensions.

Holds number of states, observables, etc.

Subclassed by *amici::Model*, *amici::ReturnData*

Public Functions

ModelDimensions() = default

Default ctor

inline **ModelDimensions**(const int nx_rdata, const int nxtrue_rdata, const int nx_solver, const int nxtrue_solver, const int nx_solver_reinit, const int np, const int nk, const int ny, const int nytrue, const int nz, const int nztrue, const int ne, const int nJ, const int nw, const int ndwdx, const int ndwdp, const int ndwdw, const int ndxdotdw, std::vector<int> ndJydy, const int ndxrdatadxsolver, const int ndxrdatadtcl, const int ndtotal_cldx_rdata, const int nnz, const int ubw, const int lbw)

Constructor with model dimensions.

Parameters

- **nx_rdata** – Number of state variables
- **nxtrue_rdata** – Number of state variables of the non-augmented model

- **nx_solver** – Number of state variables with conservation laws applied
- **nxtrue_solver** – Number of state variables of the non-augmented model with conservation laws applied
- **nx_solver_reinit** – Number of state variables with conservation laws subject to reinitialization
- **np** – Number of parameters
- **nk** – Number of constants
- **ny** – Number of observables
- **nytrue** – Number of observables of the non-augmented model
- **nz** – Number of event observables
- **nztrue** – Number of event observables of the non-augmented model
- **ne** – Number of events
- **nJ** – Number of objective functions
- **nw** – Number of repeating elements
- **ndwdx** – Number of nonzero elements in the `x` derivative of the repeating elements
- **ndwdp** – Number of nonzero elements in the `p` derivative of the repeating elements
- **ndwdw** – Number of nonzero elements in the `w` derivative of the repeating elements
- **ndxdotdw** – Number of nonzero elements in the w derivative of $xdot$
- **ndJydy** – Number of nonzero elements in the y derivative of dJy (shape `nytrue`)
- **ndxrdatadxsolver** – Number of nonzero elements in the x derivative of $x_r data$
- **ndxrdatadtcl** – Number of nonzero elements in the tcl derivative of $x_r data$
- **ndtotal_cldx_rdata** – Number of nonzero elements in the $x_r data$ derivative of $total_c l$
- **nnz** – Number of nonzero elements in Jacobian
- **ubw** – Upper matrix bandwidth in the Jacobian
- **lbw** – Lower matrix bandwidth in the Jacobian

Public Members

int **nx_rdata** = {0}

Number of states

int **nxtrue_rdata** = {0}

Number of states in the unaugmented system

int **nx_solver** = {0}

Number of states with conservation laws applied

int **nxtrue_solver** = {0}

Number of states in the unaugmented system with conservation laws applied int **nx_solver_reinit** = {0}

Number of solver states subject to reinitialization

int **np** = {0}

Number of parameters

int **nk** = {0}

Number of constants

int **ny** = {0}

Number of observables

int **nytrue** = {0}

Number of observables in the unaugmented system

int **nz** = {0}

Number of event outputs

int **nztrue** = {0}

Number of event outputs in the unaugmented system

int **ne** = {0}

Number of events

int **nw** = {0}

Number of common expressions

int **ndwdx** = {0}

Number of nonzero elements in the `x` derivative of the repeating elements

int **ndwdp** = {0}

Number of nonzero elements in the `p` derivative of the repeating elements

int **ndwdw** = {0}

Number of nonzero elements in the `w` derivative of the repeating elements

int **ndxdotdw** = {0}

Number of nonzero elements in the w derivative of $xdot$

std::vector<int> **ndJydy**

Number of nonzero elements in the y derivative of dJy (dimension `nytrue`)

int **ndxrdatadxsolver** = {0}

Number of nonzero elements in the x derivative of x_rdata

int **ndxrdatadtcl** = {0}

Number of nonzero elements in the tcl derivative of $x_r data$

int **ndtotal_cldx_rdata** = {0}

Number of nonzero elements in the $x_r data$ derivative of $total_c l$

int **nnz** = {0}

Number of nonzero entries in Jacobian

int **nJ** = {0}

Dimension of the augmented objective function for 2nd order ASA

int **ubw** = {0}

Upper bandwidth of the Jacobian

int **lbw** = {0}

Lower bandwidth of the Jacobian

Struct ModelState

- Defined in file_include_amici_model_state.h

Struct Documentation

struct **ModelState**

Exchange format to store and transfer the state of the model at a specific timepoint.

This is designed to only encompass the minimal number of attributes that need to be transferred.

Public Members

std::vector<*realtype*> **h**

Flag indicating whether a certain Heaviside function should be active or not (dimension: `ne`)

std::vector<*realtype*> **total_cl**

Total abundances for conservation laws (dimension: `nx_rdata - nx_solver`)

std::vector<*realtype*> **stotal_cl**

Sensitivities of total abundances for conservation laws (dimension: `(nx_rdata-nx_solver) x np`, row-major)

std::vector<*realtype*> **unscaledParameters**

Unscaled parameters (dimension: `np`)

std::vector<*realtype*> **fixedParameters**

Constants (dimension: nk)

std::vector<int> **plist**

Indexes of parameters wrt to which sensitivities are computed (dimension: nplist)

Struct ModelStateDerived

- Defined in file_include_amici_model_state.h

Struct Documentation

struct **ModelStateDerived**

Storage for *amici::Model* quantities computed based on *amici::ModelState* for a specific timepoint.

Serves as workspace for a model simulation to avoid repeated reallocation.

Public Functions

ModelStateDerived() = default

explicit **ModelStateDerived**(*ModelDimensions* const &dim)

Constructor from model dimensions.

Parameters dim – *Model* dimensions

Public Members

SUNMatrixWrapper **J_**

Sparse Jacobian (dimension: *amici::Model::nnz*)

SUNMatrixWrapper **JB_**

Sparse Backwards Jacobian (dimension: *amici::Model::nnz*)

SUNMatrixWrapper **dxdotdw_**

Sparse dxdotdw temporary storage (dimension: ndxdotdw)

SUNMatrixWrapper **dwdx_**

Sparse dwdx temporary storage (dimension: ndwdx)

SUNMatrixWrapper **dwdp_**

Sparse dwdp temporary storage (dimension: ndwdp)

SUNMatrixWrapper **M_**

Dense Mass matrix (dimension: nx_solver x nx_solver)

SUNMatrixWrapper **dxdotdp_full**

Temporary storage of `dxdotdp_full` data across functions (Python only) (dimension: `nplist` x `nx_solver`, nnz: dynamic, type `CSC_MAT`)

SUNMatrixWrapper **dxdotdp_explicit**

Temporary storage of `dxdotdp_explicit` data across functions (Python only) (dimension: `nplist` x `nx_solver`, nnz: `ndxdotdp_explicit`, type `CSC_MAT`)

SUNMatrixWrapper **dxdotdp_implicit**

Temporary storage of `dxdotdp_implicit` data across functions, Python-only (dimension: `nplist` x `nx_solver`, nnz: dynamic, type `CSC_MAT`)

SUNMatrixWrapper **dxdotdx_explicit**

Temporary storage of `dxdotdx_explicit` data across functions (Python only) (dimension: `nplist` x `nx_solver`, nnz: `nxdotdotdx_explicit`, type `CSC_MAT`)

SUNMatrixWrapper **dxdotdx_implicit**

Temporary storage of `dxdotdx_implicit` data across functions, Python-only (dimension: `nplist` x `nx_solver`, nnz: dynamic, type `CSC_MAT`)

SUNMatrixWrapper **dx_rdatadx_solver**

Temporary storage for `dx_rdatadx_solver` (dimension: `nx_rdata` x `nx_solver`, nnz: `ndxrdatadxsolver`, type: `CSC_MAT`)

SUNMatrixWrapper **dx_rdatadtcl**

Temporary storage for `dx_rdatadtcl` (dimension: `nx_rdata` x `ncl`, nnz: `ndxrdatadtclr`, type: `CSC_MAT`)

SUNMatrixWrapper **dtotal_cldx_rdata**

Temporary storage for `dtotal_cldx_rdata` (dimension: `ncl` x `nx_rdata`, nnz: `ndtotal_cldx_rdata`, type: `CSC_MAT`)

AmiVectorArray **dxdotdp** = {0, 0}

Temporary storage of `dxdotdp` data across functions, Matlab only (dimension: `nplist` x `nx_solver` , row-major)

std::vector<*SUNMatrixWrapper*> **dJydy_**

Sparse observable derivative of data likelihood, only used if `pythonGenerated == true` (dimension `nytrue`, `nJ` x `ny`, row-major)

std::vector<*realtype*> **dJydy_matlab_**

Observable derivative of data likelihood, only used if `pythonGenerated == false` (dimension `nJ` x `ny` x `nytrue` , row-major)

std::vector<*realtype*> **dJydsigma_**

Observable sigma derivative of data likelihood (dimension nJ x ny x nytrue, row-major) std::vector<*realtype*> **dJydx_**

State derivative of data likelihood (dimension `nJ` x `nx_solver`, row-major)

std::vector<*realtype*> **dJydp_**

Parameter derivative of data likelihood for current timepoint (dimension: nJ x nplist, row-major)

std::vector<*realtype*> **dJzdz_**

event output derivative of event likelihood (dimension nJ x nz x nztrue, row-major)

std::vector<*realtype*> **dJzdsigma_**

event sigma derivative of event likelihood (dimension nJ x nz x nztrue, row-major)

std::vector<*realtype*> **dJrzdz_**

event output derivative of event likelihood at final timepoint (dimension nJ x nz x nztrue, row-major)

std::vector<*realtype*> **dJrzdsigma_**

event sigma derivative of event likelihood at final timepoint (dimension nJ x nz x nztrue, row-major)

std::vector<*realtype*> **dJzdx_**

state derivative of event likelihood (dimension `nJ` x `nx_solver`, row-major)

std::vector<*realtype*> **dJzdp_**

parameter derivative of event likelihood for current timepoint (dimension: nJ x nplist x, row-major)

std::vector<*realtype*> **dzdx_**

state derivative of event output (dimension: nz x `nx_solver`, row-major)

std::vector<*realtype*> **dzdp_**

parameter derivative of event output (dimension: nz x nplist, row-major)

std::vector<*realtype*> **drzdx_**

state derivative of event regularization variable (dimension: `nz` x `nx_solver`, row-major)

std::vector<*realtype*> **drzdp_**

parameter derivative of event regularization variable (dimension: nz x nplist, row-major)

std::vector<*realtype*> **dydp_**

parameter derivative of observable (dimension: ny x nplist, row-major)

std::vector<*realtype*> **dydx_**

state derivative of time-resolved observable (dimension: `nx_solver` x `ny`, row-major)

std::vector<*realtype*> **w_**

temporary storage of w data across functions (dimension: nw) std::vector<*realtype*> **sx_**

temporary storage for flattened sx, (dimension: `nx_solver` x `nplist`, row-major)

std::vector<*realtype*> **x_rdata_**

temporary storage for `x_rdata` (dimension: `nx_rdata`)

std::vector<*realtype*> **sx_rdata_**

temporary storage for `sx_rdata` slice (dimension: `nx_rdata`)

std::vector<*realtype*> **y_**

temporary storage for time-resolved observable (dimension: ny)

std::vector<*realtype*> **sigmay_**

data standard deviation for current timepoint (dimension: ny)

std::vector<*realtype*> **dsigmaydp_**

temporary storage for parameter derivative of data standard deviation, (dimension: ny x nplist, row-major)

std::vector<*realtype*> **dsigmaydy_**

temporary storage for observable derivative of data standard deviation, (dimension: ny x ny, row-major)

std::vector<*realtype*> **z_**

temporary storage for event-resolved observable (dimension: nz)

std::vector<*realtype*> **rz_**

temporary storage for event regularization (dimension: nz)

std::vector<*realtype*> **sigmaz_**

temporary storage for event standard deviation (dimension: nz)

std::vector<*realtype*> **dsigmazdp_**

temporary storage for parameter derivative of event standard deviation, (dimension: nz x nplist, row-major)

std::vector<*realtype*> **deltax_**

temporary storage for change in x after event (dimension: `nx_solver`)

std::vector<*realtype*> **deltasx_**

temporary storage for change in sx after event (dimension: `nx_solver` x `nplist`, row-major)

std::vector<*realtype*> **deltaxB_**

temporary storage for change in xB after event (dimension: `nx_solver`)

std::vector<*realtype*> **deltaqB_**

temporary storage for change in qB after event (dimension: nJ x nplist, row-major)

AmiVector **x_pos_tmp_** = {0}

temporary storage of positified state variables according to stateIsNonNegative (dimension: `nx_solver`)

Struct SimulationState

- Defined in file_include_amici_model_state.h

Struct Documentation

struct **SimulationState**

implements an exchange format to store and transfer the state of a simulation at a specific timepoint.

Public Members

realtype **t**

timepoint

AmiVector **x**

state variables

AmiVector **dx**

state variables

AmiVectorArray **sx**

state variable sensitivity

ModelState **state**

state of the model that was used for simulation

Class AbstractModel

- Defined in file_include_amici_abstract_model.h

Inheritance Relationships

Derived Type

- `public amici::Model` (*Class Model*)

Class Documentation

class **AbstractModel**

Abstract base class of *amici::Model* defining functions that need to be implemented in an AMICI model.

Some functions have empty default implementations or throw. This class shall not have any data members.

Subclassed by *amici::Model*

Public Functions

virtual **~AbstractModel**() = default

virtual std::unique_ptr<*Solver*> **getSolver**() = 0

Retrieves the solver object.

Returns The *Solver* instance

virtual void **froot**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx, gsl::span<*realtype*> root) = 0

Root function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **root** – array to which values of the root function will be written

virtual void **fxdot**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx, *AmiVector* &xdot) = 0

Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – array to which values of the residual function will be written

virtual void **fsxdot**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx, int ip, const *AmiVector* &sx, const *AmiVector* &sdx, *AmiVector* &sxdot) = 0

Sensitivity Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **ip** – parameter index
- **sx** – sensitivity state
- **sdx** – time derivative of sensitivity state (DAE only)
- **sxdot** – array to which values of the sensitivity residual function will be written

virtual void **fxBdot_ss**(const *realtype* t, const *AmiVector* &xB, const *AmiVector* &dxB, *AmiVector* &xBdot) = 0

Residual function backward when running in steady state mode.

Parameters

- **t** – time
- **xB** – adjoint state
- **dxB** – time derivative of state (DAE only)
- **xBdot** – array to which values of the residual function will be written

virtual void **fJSparseB_ss**(SUNMatrix JB) = 0

Sparse Jacobian function backward, steady state case.

Parameters **JB** – sparse matrix to which values of the Jacobian will be written

virtual void **writeSteadystateJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot) = 0

Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.

Parameters

- **t** – timepoint
- **cj** – scalar in Jacobian
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint state right hand side

virtual void **fJ**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) = 0

Dense Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – dense matrix to which values of the jacobian will be written

virtual void **fJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) = 0

Dense Jacobian function.

Parameters

- **t** – time

- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

virtual void **fJSparse**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) = 0

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – sparse matrix to which values of the Jacobian will be written

virtual void **fJSparseB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) = 0

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

virtual void **fJDiag**(const *realtype* t, *AmiVector* &Jdiag, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx) = 0

Diagonal Jacobian function.

Parameters

- **t** – time
- **Jdiag** – array to which the diagonal of the Jacobian will be written
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state

- **dx** – time derivative of state (DAE only)

virtual void **fdxdotdp**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx) = 0

Model-specific sparse implementation of explicit parameter derivative of right hand side.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)

virtual void **fJv**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, const *AmiVector* &v, *AmiVector* &nJv, *realtype* cj) = 0

Jacobian multiply function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **v** – multiplication vector (unused)
- **nJv** – array to which result of multiplication will be written
- **cj** – scaling factor (inverse of timestep, DAE only)

virtual std::string **getAmiciVersion**() const

Returns the AMICI version that was used to generate the model.

Returns AMICI version string

virtual std::string **getAmiciCommit**() const

Returns the AMICI commit that was used to generate the model.

Returns AMICI commit string

virtual void **fx0**(*realtype* *x0, const *realtype* t, const *realtype* *p, const *realtype* *k)

Model-specific implementation of fx0.

Parameters

- **x0** – initial state
- **t** – initial time
- **p** – parameter vector
- **k** – constant vector

virtual bool **isFixedParameterStateReinitializationAllowed**() const

Function indicating whether reinitialization of states depending on fixed parameters is permissible.

Returns flag indicating whether reinitialization of states depending on fixed parameters is permissible

virtual void **fx0_fixedParameters**(*realtype* *x0, const *realtype* t, const *realtype* *p, const *realtype* *k, gsl::span<const int> reinitialization_state_idxs)

Model-specific implementation of fx0_fixedParameters.

Parameters

- **x0** – initial state
- **t** – initial time
- **p** – parameter vector
- **k** – constant vector
- **reinitialization_state_idxs** – Indices of states to be reinitialized based on provided constants / fixed parameters.

virtual void **fsx0_fixedParameters**(*realtype* *sx0, const *realtype* t, const *realtype* *x0, const *realtype* *p, const *realtype* *k, int ip, gsl::span<const int> reinitialization_state_idxs)

Model-specific implementation of fsx0_fixedParameters.

Parameters

- **sx0** – initial state sensitivities
- **t** – initial time
- **x0** – initial state
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index
- **reinitialization_state_idxs** – Indices of states to be reinitialized based on provided constants / fixed parameters.

virtual void **fsx0**(*realtype* *sx0, const *realtype* t, const *realtype* *x0, const *realtype* *p, const *realtype* *k, int ip)

Model-specific implementation of fsx0.

Parameters

- **sx0** – initial state sensitivities
- **t** – initial time
- **x0** – initial state
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index

virtual void **fdx0**(*AmiVector* &x0, *AmiVector* &dx0)

Initial value for time derivative of states (only necessary for DAEs)

Parameters

- **x0** – Vector with the initial states
- **dx0** – Vector to which the initial derivative states will be written (only DAE)

virtual void **fstau**(*realtype* *stau, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *tcl, const *realtype* *sx, int ip, int ie)

Model-specific implementation of fstau.

Parameters

- **stau** – total derivative of event timepoint
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **tcl** – total abundances for conservation laws
- **sx** – current state sensitivity
- **ip** – sensitivity index
- **ie** – event index

virtual void **fy**(*realtype* *y, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w)

Model-specific implementation of fy.

Parameters

- **y** – model output at current timepoint
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector

virtual void **fdydp**(*realtype* *dydp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *w, const *realtype* *dwdp)

Model-specific implementation of fdydp (MATLAB-only)

Parameters

- **dydp** – partial derivative of observables y w.r.t. model parameters p
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested
- **w** – repeating elements vector
- **dwdp** – Recurring terms in xdot, parameter derivative

virtual void **fdydp**(*realtype* *dydp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *w, const *realtype* *tcl, const *realtype* *dtcldp)

Model-specific implementation of fdydp (Python)

Parameters

- **dydp** – partial derivative of observables y w.r.t. model parameters p
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested
- **w** – repeating elements vector
- **tcl** – total abundances for conservation laws
- **dtcldp** – Sensitivities of total abundances for conservation laws

virtual void **fdydx**(*realtype* *dydx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *dwdx)

Model-specific implementation of fdydx.

Parameters

- **dydx** – partial derivative of observables y w.r.t. model states x
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector
- **dwdx** – Recurring terms in xdot, state derivative

virtual void **fz**(*realtype* *z, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fz.

Parameters

- **z** – value of event output
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fsz**(*realtype* *sz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *sx, int ip)

Model-specific implementation of fsz.

Parameters

- **sz** – Sensitivity of rz, total derivative
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **sx** – current state sensitivity
- **ip** – sensitivity index

virtual void **frz**(*realtype* *rz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of frz.

Parameters

- **rz** – value of root function at current timepoint (non-output events not included)
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fsrz**(*realtype* *srz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *sx, int ip)

Model-specific implementation of fsrz.

Parameters

- **srz** – Sensitivity of rz, total derivative
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **sx** – current state sensitivity
- **h** – Heaviside vector
- **ip** – sensitivity index

virtual void **fdzdp**(*realtype* *dzdp, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip)

Model-specific implementation of fdzdp.

Parameters

- **dzdp** – partial derivative of event-resolved output z w.r.t. model parameters p
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested

virtual void **fdzdx**(*realtype* *dzdx, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fdzdx.

Parameters

- **dzdx** – partial derivative of event-resolved output z w.r.t. model states x
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fdrzdp**(*realtype* *drzdp, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip)

Model-specific implementation of fdrzdp.

Parameters

- **drzdp** – partial derivative of root output rz w.r.t. model parameters p
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested

virtual void **fdrzdx**(*realtype* *drzdx, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fdrzdx.

Parameters

- **drzdx** – partial derivative of root output rz w.r.t. model states x

- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fdeltax**(*realtype* *deltax, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ie, const *realtype* *xdot, const *realtype* *xdot_old)

Model-specific implementation of fdeltax.

Parameters

- **deltax** – state update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side

virtual void **fdeltasx**(*realtype* *deltasx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, int ip, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *sx, const *realtype* *stau, const *realtype* *tcl)

Model-specific implementation of fdeltasx.

Parameters

- **deltasx** – sensitivity update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector
- **ip** – sensitivity index
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side
- **sx** – state sensitivity
- **stau** – event-time sensitivity

- **tcl** – total abundances for conservation laws

virtual void **fdeltaxB**(*realtype* *deltaxB, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *xB)

Model-specific implementation of fdeltaxB.

Parameters

- **deltaxB** – adjoint state update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side
- **xB** – current adjoint state

virtual void **fdeltaqB**(*realtype* *deltaqB, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *xB)

Model-specific implementation of fdeltaqB.

Parameters

- **deltaqB** – sensitivity update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – sensitivity index
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side
- **xB** – adjoint state

virtual void **fsigmay**(*realtype* *sigmay, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y)

Model-specific implementation of fsigmay.

Parameters

- **sigmay** – standard deviation of measurements
- **t** – current time

- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t

virtual void **fdsigmaydp**(*realtype* *dsigmaydp, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y, int ip)

Model-specific implementation of fdsigmaydp.

Parameters

- **dsigmaydp** – partial derivative of standard deviation of measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t
- **ip** – sensitivity index

virtual void **fdsigmaydy**(*realtype* *dsigmaydy, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y)

Model-specific implementation of fsigmay.

Parameters

- **dsigmaydy** – partial derivative of standard deviation of measurements w.r.t. model outputs
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t

virtual void **fsigmaz**(*realtype* *sigmaz, const *realtype* t, const *realtype* *p, const *realtype* *k)

Model-specific implementation of fsigmaz.

Parameters

- **sigmaz** – standard deviation of event measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector

virtual void **fdsigmazdp**(*realtype* *dsigmazdp, const *realtype* t, const *realtype* *p, const *realtype* *k, int ip)

Model-specific implementation of fsigmaz.

Parameters

- **dsigmazdp** – partial derivative of standard deviation of event measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index

virtual void **fJy**(*realtype* *nllh, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fJy.

Parameters

- **nllh** – negative log-likelihood for measurements y
- **iy** – output index
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurements at timepoint

virtual void **fJz**(*realtype* *nllh, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fJz.

Parameters

- **nllh** – negative log-likelihood for event measurements z
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurements at timepoint

virtual void **fJrz**(*realtype* *nllh, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz)

Model-specific implementation of fJrz.

Parameters

- **nllh** – regularization for event measurements z
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fdJydy**(*realtype* *dJydy, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fdJydy.

Parameters

- **dJydy** – partial derivative of time-resolved measurement negative log-likelihood Jy
- **iy** – output index

- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurement at timepoint

virtual void **fdJydy_colptrs**(*SUNMatrixWrapper* &dJydy, int index)

Model-specific implementation of fdJydy colptrs.

Parameters

- **dJydy** – sparse matrix to which colptrs will be written
- **index** – ytrue index

virtual void **fdJydy_rowvals**(*SUNMatrixWrapper* &dJydy, int index)

Model-specific implementation of fdJydy rowvals.

Parameters

- **dJydy** – sparse matrix to which rowvals will be written
- **index** – ytrue index

virtual void **fdJydsigma**(*realtype* *dJydsigma, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fdJydsigma.

Parameters

- **dJydsigma** – Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigmay
- **iy** – output index
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurement at timepoint

virtual void **fdJzdz**(*realtype* *dJzdz, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fdJzdz.

Parameters

- **dJzdz** – partial derivative of event measurement negative log-likelihood Jz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurement at timepoint

virtual void **fdJzdsigma**(*realtype* *dJzdsigma, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fdJzdsigma.

Parameters

- **dJzdsigma** – Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurement at timepoint

virtual void **fdJrzdz**(*realtype* *dJrzdz, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *rz, const *realtype* *sigmaz)

Model-specific implementation of fdJrzdz.

Parameters

- **dJrzdz** – partial derivative of event penalization Jrz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **rz** – model root output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fdJrzdsigma**(*realtype* *dJrzdsigma, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *rz, const *realtype* *sigmaz)

Model-specific implementation of fdJrzdsigma.

Parameters

- **dJrzdsigma** – Sensitivity of event penalization Jrz w.r.t. standard deviation sigmaz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **rz** – model root output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fw**(*realtype* *w, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *tcl)

Model-specific implementation of fw.

Parameters

- **w** – Recurring terms in xdot
- **t** – timepoint

- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **tcl** – total abundances for conservation laws

virtual void **fdwdp**(*realtype* *dwdp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl, const *realtype* *stcl)

Model-specific sparse implementation of dwdp.

Parameters

- **dwdp** – Recurring terms in xdot, parameter derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws
- **stcl** – sensitivities of total abundances for conservation laws

virtual void **fdwdp_colptrs**(*SUNMatrixWrapper* &dwdp)

Model-specific implementation for dwdp, column pointers.

Parameters **dwdp** – sparse matrix to which colptrs will be written

virtual void **fdwdp_rowvals**(*SUNMatrixWrapper* &dwdp)

Model-specific implementation for dwdp, row values.

Parameters **dwdp** – sparse matrix to which rowvals will be written

virtual void **fdwdp**(*realtype* *dwdp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl, const *realtype* *stcl, int ip)

Model-specific sensitivity implementation of dwdp.

Parameters

- **dwdp** – Recurring terms in xdot, parameter derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws
- **stcl** – sensitivities of total abundances for conservation laws

- **ip** – sensitivity parameter index

virtual void **fdwdx**(*realtype* *dwdx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl)

Model-specific implementation of dwdx, data part.

Parameters

- **dwdx** – Recurring terms in xdot, state derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws

virtual void **fdwdx_colptrs**(*SUNMatrixWrapper* &dwdx)

Model-specific implementation for dwdx, column pointers.

Parameters **dwdx** – sparse matrix to which colptrs will be written

virtual void **fdwdx_rowvals**(*SUNMatrixWrapper* &dwdx)

Model-specific implementation for dwdx, row values.

Parameters **dwdx** – sparse matrix to which rowvals will be written

virtual void **fdwdw**(*realtype* *dwdw, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl)

Model-specific implementation of fdwdw, no w chainrule (Py)

Parameters

- **dwdw** – partial derivative w wrt w
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – Total abundances for conservation laws

virtual void **fdwdw_colptrs**(*SUNMatrixWrapper* &dwdw)

Model-specific implementation of fdwdw, colptrs part.

Parameters **dwdw** – sparse matrix to which colptrs will be written

virtual void **fdwdw_rowvals**(*SUNMatrixWrapper* &dwdw)

Model-specific implementation of fdwdw, rowvals part.

Parameters **dwdw** – sparse matrix to which rowvals will be written virtual void **fdx_rdatadx_solver**(*realtype* *dx_rdatadx_solver, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k)

Compute dx_rdata / dx_solver.

Parameters

- **dx_rdatadx_solver** – dx_rdata / dx_solver
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws

virtual void **fdx_rdatadx_solver_colptrs**(*SUNMatrixWrapper* &dxrdatadxsolver)

Model-specific implementation of fdx_rdatadx_solver, colptrs part.

Parameters **dxrdatadxsolver** – sparse matrix to which colptrs will be written

virtual void **fdx_rdatadx_solver_rowvals**(*SUNMatrixWrapper* &dxrdatadxsolver)

Model-specific implementation of fdx_rdatadx_solver, rowvals part.

Parameters **dxrdatadxsolver** – sparse matrix to which rowvals will be written

virtual void **fdx_rdatadp**(*realtype* *dx_rdatadp, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k, const int ip)

Compute dx_rdata / dp.

Parameters

- **dx_rdatadp** – dx_rdata / dp
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws
- **ip** – Sensitivity index

virtual void **fdx_rdatadtcl**(*realtype* *dx_rdatadtcl, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k)

Compute dx_rdata / dtcl.

Parameters

- **dx_rdatadtcl** – dx_rdata / dtcl
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws

virtual void **fdx_rdatadtcl_colptrs**(*SUNMatrixWrapper* &dx_rdatadtcl)

Model-specific implementation of fdx_rdatadtcl, colptrs part.

Parameters **dx_rdatadtcl** – sparse matrix to which colptrs will be written virtual void **fdx_rdatadtcl_rowvals**(*SUNMatrixWrapper* &dx_rdatadtcl)

Model-specific implementation of fdx_rdatadtcl, rowvals part.

Parameters **dx_rdatadtcl** – sparse matrix to which rowvals will be written

virtual void **fdtotal_cldp**(*realtype* *dtotal_cldp, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k, const int ip)

Compute dtotal_cl / dp.

Parameters

- **dtotal_cldp** – dtotal_cl / dp
- **x_rdata** – State variables with conservation laws applied
- **p** – parameter vector
- **k** – constant vector
- **ip** – Sensitivity index

virtual void **fdtotal_cldx_rdata**(*realtype* *dtotal_cldx_rdata, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k, const *realtype* *tcl)

Compute dtotal_cl / dx_rdata.

Parameters

- **dtotal_cldx_rdata** – dtotal_cl / dx_rdata
- **x_rdata** – State variables with conservation laws applied
- **p** – parameter vector
- **k** – constant vector
- **tcl** – Total abundances for conservation laws

virtual void **fdtotal_cldx_rdata_colptrs**(*SUNMatrixWrapper* &dtotal_cldx_rdata)

Model-specific implementation of fdtotal_cldx_rdata, colptrs part.

Parameters **dtotal_cldx_rdata** – sparse matrix to which colptrs will be written

virtual void **fdtotal_cldx_rdata_rowvals**(*SUNMatrixWrapper* &dtotal_cldx_rdata)

Model-specific implementation of fdtotal_cldx_rdata, rowvals part.

Parameters **dtotal_cldx_rdata** – sparse matrix to which rowvals will be written

Class AmiException

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public std::exception`

Derived Types

- `public amici::CvodeException` (*Class CvodeException*)
- `public amici::IDAException` (*Class IDAException*)
- `public amici::IntegrationFailure` (*Class IntegrationFailure*)
- `public amici::IntegrationFailureB` (*Class IntegrationFailureB*)
- `public amici::NewtonFailure` (*Class NewtonFailure*)
- `public amici::SetupFailure` (*Class SetupFailure*)

Class Documentation

class **AmiException** : public std::exception

AMICI exception class.

Has a printf style interface to allow easy generation of error messages

Subclassed by *amici::CvodeException*, *amici::IDAException*, *amici::IntegrationFailure*, *amici::IntegrationFailureB*, *amici::NewtonFailure*, *amici::SetupFailure*

Public Functions

AmiException(int const first_frame = 3)

Default ctor.

Parameters **first_frame** – Index of first frame to include

explicit **AmiException**(char const *fmt, ...)

Constructor with printf style interface.

Parameters

- **fmt** – error message with printf format
- **...** – printf formatting variables

char const ***what**() const noexcept override

Override of default error message function.

Returns msg error message

char const ***getBacktrace**() const

Returns the stored backtrace.

Returns trace backtrace

void **storeBacktrace**(int nMaxFrames, int const first_frame)

Stores the current backtrace.

Parameters

- **nMaxFrames** – number of frames to go back in stacktrace
- **first_frame** – Index of first frame to include

Protected Functions

void **storeMessage**(char const *fmt, va_list argptr)

Store the provided message.

Parameters

- **fmt** – error message with printf format
- **argptr** – pointer to variadic argument list

Class AmiVector

- Defined in file_include_amici_vector.h

Class Documentation

class **AmiVector**

AmiVector class provides a generic interface to the NVector_Serial struct

Public Functions

AmiVector() = default

Default constructor.

inline explicit **AmiVector**(const long int length)

empty constructor

Creates an std::vector<realtype> and attaches the data pointer to a newly created N_Vector_Serial. Using N_VMake_Serial ensures that the N_Vector module does not try to deallocate the data vector when calling N_VDestroy_Serial

Parameters **length** – number of elements in vector

inline explicit **AmiVector**(std::vector<*realtype*> rvec)

constructor from std::vector,

Moves data from std::vector and constructs an nvec that points to the data

Parameters **rvec** – vector from which the data will be moved

inline explicit **AmiVector**(gsl::span<*realtype*> rvec)

constructor from gsl::span,

Copy data from gsl::span and constructs a vector

Parameters **rvec** – vector from which the data will be copied inline **AmiVector**(const *AmiVector* &vold)

copy constructor

Parameters **vold** – vector from which the data will be copied

inline **AmiVector**(*AmiVector* &&other) noexcept

move constructor

Parameters **other** – vector from which the data will be moved

~AmiVector()

destructor

AmiVector &**operator=**(*AmiVector* const &other)

copy assignment operator

Parameters **other** – right hand side

Returns left hand side

inline *AmiVector* &**operator*=**(*AmiVector* const &multiplier)

operator *= (element-wise multiplication)

Parameters **multiplier** – multiplier

Returns result

inline *AmiVector* &**operator/=**(*AmiVector* const &divisor)

operator /= (element-wise division)

Parameters **divisor** – divisor

Returns result

inline auto **begin**()

Returns an iterator that points to the first element of the vector.

Returns iterator that points to the first element

inline auto **end**()

Returns an iterator that points to one element after the last element of the vector.

Returns iterator that points to one element after the last element

realtype ***data**()

data accessor

Returns pointer to data array

const *realtype* ***data**() const

const data accessor

Returns const pointer to data array

N_Vector **getNVector**()

N_Vector accessor.

Returns N_Vector

const_N_Vector **getNVector**() const

N_Vector accessor.

Returns N_Vector

std::vector<*realtype*> const &**getVector**() const

Vector accessor.

Returns Vector

int **getLength**() const

returns the length of the vector

Returns length

void **zero**()

fills vector with zero values

void **minus**()

changes the sign of data elements

void **set**(*realtype* val)

sets all data elements to a specific value

Parameters **val** – value for data elements

realtype &**operator[]**(int pos)

accessor to data elements of the vector

Parameters **pos** – index of element

Returns element

realtype &**at**(int pos)

accessor to data elements of the vector

Parameters **pos** – index of element

Returns element

const *realtype* &**at**(int pos) const

accessor to data elements of the vector

Parameters **pos** – index of element

Returns element

void **copy**(const *AmiVector* &other)

copies data from another *AmiVector*

Parameters **other** – data source

inline void **abs**()

Take absolute value (in-place)

Class AmiVectorArray

- Defined in file_include_amici_vector.h

Class Documentation

class **AmiVectorArray**

AmiVectorArray class.

Provides a generic interface to arrays of NVector_Serial structs

Public Functions

AmiVectorArray() = default

Default constructor.

AmiVectorArray(long int length_inner, long int length_outer)

empty constructor

Creates an std::vector<realype> and attaches the data pointer to a newly created N_VectorArray using CloneVectorArrayEmpty ensures that the N_Vector module does not try to deallocate the data vector when calling N_VDestroyVectorArray_Serial

Parameters

- **length_inner** – length of vectors
- **length_outer** – number of vectors

AmiVectorArray(const *AmiVectorArray* &vaold)

copy constructor

Parameters **vaold** – object to copy from

~AmiVectorArray() = default

AmiVectorArray &**operator=**(*AmiVectorArray* const &other)

copy assignment operator

Parameters **other** – right hand side

Returns left hand side

realtype ***data**(int pos)

accessor to data of *AmiVector* elements

Parameters **pos** – index of *AmiVector*

Returns pointer to data array

const *realtype* ***data**(int pos) const

const accessor to data of *AmiVector* elements

Parameters **pos** – index of *AmiVector*

Returns const pointer to data array

realtype &**at**(int ipos, int jpos)

accessor to elements of *AmiVector* elements

Parameters

- **ipos** – inner index in *AmiVector*
- **jpos** – outer index in *AmiVectorArray*

Returns element

const *realtype* &**at**(int ipos, int jpos) const

const accessor to elements of *AmiVector* elements

Parameters

- **ipos** – inner index in *AmiVector*
- **jpos** – outer index in *AmiVectorArray*

Returns element

N_Vector ***getNVectorArray**()

accessor to NVectorArray

Returns N_VectorArray

N_Vector **getNVector**(int pos)

accessor to NVector element

Parameters **pos** – index of corresponding *AmiVector*

Returns N_Vector

const_N_Vector **getNVector**(int pos) const

const accessor to NVector element

Parameters **pos** – index of corresponding *AmiVector*

Returns N_Vector

AmiVector &**operator[]**(int pos)

accessor to *AmiVector* elements

Parameters **pos** – index of *AmiVector*

Returns *AmiVector*

const *AmiVector* &**operator[]**(int pos) const

const accessor to *AmiVector* elements

Parameters **pos** – index of *AmiVector*

Returns const *AmiVector*

int **getLength**() const

length of *AmiVectorArray*

Returns length

void **zero**()

set every *AmiVector* in *AmiVectorArray* to zero

void **flatten_to_vector**(std::vector<*realtype*> &vec) const

flattens the *AmiVectorArray* to a vector in row-major format

Parameters **vec** – vector into which the *AmiVectorArray* will be flattened. Must have length equal to number of elements.

void **copy**(const *AmiVectorArray* &other)

copies data from another *AmiVectorArray*

Parameters **other** – data source

Class BackwardProblem

- Defined in file_include_amici_backwardproblem.h

Class Documentation

class **BackwardProblem**

class to solve backwards problems.

solves the backwards problem for adjoint sensitivity analysis and handles events and data-points

Public Functions

explicit **BackwardProblem**(const *ForwardProblem* &fwd, const *SteadystateProblem* *posteq)

Construct backward problem from forward problem.

Parameters

- **fwd** – pointer to corresponding forward problem
- **posteq** – pointer to postequilibration problem, can be nullptr

void **workBackwardProblem**()

Solve the backward problem.

If adjoint sensitivities are enabled this will also compute sensitivities. workForwardProblem must be called before this is function is called.

inline *realtype* **gett**() const

Accessor for current time t.

Returns t

inline int **getwhich**() const

Accessor for which.

Returns which

inline int ***getwhichptr**()

Accessor for pointer to which.

Returns which

inline std::vector<*realtype*> const &**getdJydx**() const

Accessor for dJydx.

Returns dJydx

inline *AmiVector* const &**getAdjointState**() const

Accessor for xB.

Returns xB

inline *AmiVector* const &**getAdjointQuadrature**() const

Accessor for xQB.

Returns xQB

Class ConditionContext

- Defined in file_include_amici_edata.h

Inheritance Relationships

Base Type

- `public amici::ContextManager` (*Class ContextManager*)

Class Documentation

class **ConditionContext** : public amici::*ContextManager*

The *ConditionContext* class applies condition-specific *amici::Model* settings and restores them when going out of scope.

Public Functions

explicit **ConditionContext**(*Model* *model, *ExpData* const *edata = nullptr, *FixedParameterContext* fpc = *FixedParameterContext*::*simulation*)

Apply condition-specific settings from edata to model while keeping a backup of the original values.

Parameters

- **model** –
- **edata** –
- **fpc** – flag indicating which fixedParameter from edata to apply

ConditionContext &**operator=**(*ConditionContext* const &other) = delete

~ConditionContext()

void **applyCondition**(*ExpData* const *edata, *FixedParameterContext* fpc)

Apply condition-specific settings from edata to the constructor-supplied model, not changing the settings which were backed-up in the constructor call.

Parameters

- **edata** –
- **fpc** – flag indicating which fixedParameter from edata to apply

void **restore**()

Restore original settings on constructor-supplied *amici::Model*. Will be called during destruction. Explicit call is generally not necessary.

Class ContextManager

- Defined in file_include_amici_misc.h

Inheritance Relationships

Derived Types

- `public amici::ConditionContext` (*Class ConditionContext*)
- `public amici::FinalStateStorer` (*Class FinalStateStorer*)
- `public amici::ModelContext` (*Class ModelContext*)

Class Documentation

class **ContextManager**

Generic implementation for a context manager, explicitly deletes copy and move operators for derived classes.

Subclassed by *amici::ConditionContext*, *amici::FinalStateStorer*, *amici::ModelContext*

Public Functions

ContextManager() = default

ContextManager(*ContextManager* &other) = delete

ContextManager(*ContextManager* &&other) = delete

Class CvodeException

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **CvodeException** : public amici::*AmiException*

cvode exception handler class

Public Functions

CvodeException(int error_code, char const *function)

Constructor.

Parameters

- **error_code** – error code returned by cvode function
- **function** – cvode function name

Class CVodeSolver

- Defined in file_include_amici_solver_cvodes.h

Inheritance Relationships

Base Type

- `public amici::Solver` (*Class Solver*)

Class Documentation

class **CVodeSolver** : public amici::*Solver*

The *CVodeSolver* class is a wrapper around the SUNDIALS CVODES solver.

Public Functions

~CVodeSolver() override = default

virtual *Solver* ***clone**() const override

Clone this instance.

Returns The clone

virtual void **reInit**(*realtype* t0, const *AmiVector* &yy0, const *AmiVector* &yp0) const override

Reinitializes the states in the solver after an event occurrence.

Parameters

- **t0** – reinitialization timepoint
- **yy0** – initial state variables
- **yp0** – initial derivative state variables (DAE only)

virtual void **sensReInit**(const *AmiVectorArray* &yyS0, const *AmiVectorArray* &ypS0) const override

Reinitializes the state sensitivities in the solver after an event occurrence.

Parameters

- **yyS0** – new state sensitivity
- **ypS0** – new derivative state sensitivities (DAE only)

virtual void **sensToggleOff**() const override

Switches off computation of state sensitivities without deallocating the memory for sensitivities.

virtual void **reInitB**(int which, *realtype* tB0, const *AmiVector* &yyB0, const *AmiVector* &ypB0) const override

Reinitializes the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **tB0** – reinitialization timepoint
- **yyB0** – new adjoint state
- **ypB0** – new adjoint derivative state

virtual void **quadReInitB**(int which, const *AmiVector* &yQB0) const override

Reinitialize the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **yQB0** – new adjoint quadrature state

virtual int **solve**(*realtype* tout, int itask) const override

Solves the forward problem until a predefined timepoint.

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP

Returns status flag indicating success of execution

virtual int **solveF**(*realtype* tout, int itask, int *ncheckPtr) const override

Solves the forward problem until a predefined timepoint (adjoint only)

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP
- **ncheckPtr** – pointer to a number that counts the internal checkpoints

Returns status flag indicating success of execution

virtual void **solveB**(*realtype* tBout, int itaskB) const override

Solves the backward problem until a predefined timepoint (adjoint only)

Parameters

- **tBout** – timepoint until which simulation should be performed
- **itaskB** – task identifier, can be CV_NORMAL or CV_ONE_STEP

virtual void **getDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getSensDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getQuadDkyB**(*realtype* t, int k, int which) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order
- **which** – index of backward problem

virtual void **getDkyB**(*realtype* t, int k, int which) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order
- **which** – index of backward problem

virtual void **getRootInfo**(int *rootsfound) const override

getRootInfo extracts information which event occurred

Parameters **rootsfound** – array with flags indicating whether the respective event occurred

virtual void **setStopTime**(*realtype* tstop) const override

Sets a timepoint at which the simulation will be stopped.

Parameters **tstop** – timepoint until which simulation should be performed

virtual void **turnOffRootFinding**() const override

Disable rootfinding.

virtual const *Model* ***getModel**() const override

Accessor function to the model stored in the user data

Returns user data model

virtual void **setLinearSolver**() const override

Sets the linear solver for the forward problem.

virtual void **setLinearSolverB**(int which) const override

Sets the linear solver for the backward problem.

Parameters **which** – index of the backward problem virtual void **setNonLinearSolver**() const override

Set the non-linear solver for the forward problem.

virtual void **setNonLinearSolverSens**() const override

Set the non-linear solver for sensitivities.

virtual void **setNonLinearSolverB**(int which) const override

Set the non-linear solver for the backward problem.

Parameters `which` – index of the backward problem

Solver() = default

Default constructor.

Solver(const *Solver* &other)

Solver copy constructor.

Parameters `other` –

Protected Functions

virtual void **calcIC**(*realtype* tout1) const override

Calculates consistent initial conditions, assumes initial states to be correct (DAE only)

Parameters `tout1` – next timepoint to be computed (sets timescale)

virtual void **calcICB**(int which, *realtype* tout1) const override

Calculates consistent initial conditions for the backwards problem, assumes initial states to be correct (DAE only)

Parameters

- `which` – identifier of the backwards problem
- `tout1` – next timepoint to be computed (sets timescale)

virtual void **getB**(int which) const override

extracts the adjoint state at the current timepoint from solver memory and writes it to the xB member variable

Parameters `which` – index of the backwards problem

virtual void **getSens**() const override

extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

virtual void **getQuadB**(int which) const override

extracts the adjoint quadrature state at the current timepoint from solver memory and writes it to the xQB member variable

Parameters `which` – index of the backwards problem

virtual void **getQuad**(*realtype* &t) const override

extracts the quadrature at the current timepoint from solver memory and writes it to the xQ member variable

Parameters `t` – timepoint for quadrature extraction virtual void **getQuadDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **reInitPostProcessF**(*realtype* tnext) const override

reInitPostProcessF postprocessing of the solver memory after a discontinuity in the forward problem

Parameters **tnext** – next timepoint (defines integration direction)

virtual void **reInitPostProcessB**(*realtype* tnext) const override

reInitPostProcessB postprocessing of the solver memory after a discontinuity in the backward problem

Parameters **tnext** – next timepoint (defines integration direction)

void **reInitPostProcess**(void *cv_mem, *realtype* *t, *AmiVector* *yout, *realtype* tout) const

Postprocessing of the solver memory after a discontinuity.

Parameters

- **cv_mem** – pointer to CVODES solver memory object
- **t** – pointer to integration time
- **yout** – new state vector
- **tout** – anticipated next integration timepoint.

virtual void **allocateSolver**() const override

Create specifies solver method and initializes solver memory for the forward problem.

virtual void **setSStolerances**(double rtol, double atol) const override

sets scalar relative and absolute tolerances for the forward problem

Parameters

- **rtol** – relative tolerances
- **atol** – absolute tolerances

virtual void **setSensSStolerances**(double rtol, const double *atol) const override

activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

- **rtol** – relative tolerances
- **atol** – array of absolute tolerances for every sensitivity variable

virtual void **setSensErrCon**(bool error_corr) const override

SetSensErrCon specifies whether error control is also enforced for sensitivities for the forward problem

Parameters **error_corr** – activation flag

virtual void **setQuadErrConB**(int which, bool flag) const override

Specifies whether error control is also enforced for the backward quadrature problem.

Parameters

- **which** – identifier of the backwards problem
- **flag** – activation flag

virtual void **setQuadErrCon**(bool flag) const override

Specifies whether error control is also enforced for the forward quadrature problem.

Parameters **flag** – activation flag

virtual void **setErrHandlerFn**() const override

Attaches the error handler function (errMsgIdAndTxt) to the solver.

virtual void **setUserData**() const override

Attaches the user data to the forward problem.

virtual void **setUserDataB**(int which) const override

attaches the user data to the backward problem

Parameters **which** – identifier of the backwards problem

virtual void **setMaxNumSteps**(long int mxsteps) const override

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters **mxsteps** – number of steps

virtual void **setStabLimDet**(int stldet) const override

activates stability limit detection for the forward problem

Parameters **stldet** – flag for stability limit detection (TRUE or FALSE)

virtual void **setStabLimDetB**(int which, int stldet) const override

activates stability limit detection for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **stldet** – flag for stability limit detection (TRUE or FALSE)

virtual void **setId**(const *Model* *model) const override

specify algebraic/differential components (DAE only)

Parameters **model** – model specification

virtual void **setSuppressAlg**(bool flag) const override

deactivates error control for algebraic components (DAE only)

Parameters **flag** – deactivation flag

void **resetState**(void *cv_mem, *const_N_Vector* y0) const

resetState reset the CVODES solver to restart integration after a rhs discontinuity.

Parameters

- **cv_mem** – pointer to CVODES solver memory object
- **y0** – new state vector

virtual void **setSensParams**(const *realtype* *p, const *realtype* *pbar, const int *plist) const override

specifies the scaling and indexes for sensitivity computation

Parameters

- **p** – parameters
- **pbar** – parameter scaling constants
- **plist** – parameter index list

virtual void **adjInit**() const override

initializes the adjoint problem

virtual void **quadInit**(const *AmiVector* &xQ0) const override

initializes the quadratures

Parameters **xQ0** – vector with initial values for xQ

virtual void **allocateSolverB**(int *which) const override

Specifies solver method and initializes solver memory for the backward problem.

Parameters **which** – identifier of the backwards problem

virtual void **setSStolerancesB**(int which, *realtype* relTolB, *realtype* absTolB) const override

sets relative and absolute tolerances for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **relTolB** – relative tolerances
- **absTolB** – absolute tolerances

virtual void **quadSStolerancesB**(int which, *realtype* reltolQB, *realtype* abstolQB) const override

sets relative and absolute tolerances for the quadrature backward problem

Parameters

- **which** – identifier of the backwards problem
- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual void **quadSStolerances**(*realtype* reltolQ, *realtype* abstolQ) const override

sets relative and absolute tolerances for the quadrature problem

Parameters

- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual void **setMaxNumStepsB**(int which, long int mxstepsB) const override

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters

- **which** – identifier of the backwards problem
- **mxstepsB** – number of steps

virtual void **diag**() const override

attaches a diagonal linear solver to the forward problem

virtual void **diagB**(int which) const override

attaches a diagonal linear solver to the backward problem

Parameters **which** – identifier of the backwards problem

virtual void **getNumSteps**(const void *ami_mem, long int *numsteps) const override

reports the number of solver steps

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numsteps** – output array

virtual void **getNumRhsEvals**(const void *ami_mem, long int *numrhsevals) const override

reports the number of right hand evaluations

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numrhsevals** – output array

virtual void **getNumErrTestFails**(const void *ami_mem, long int *numerrtestfails) const override

reports the number of local error test failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numerrtestfails** – output array

virtual void **getNumNonlinSolvConvFails**(const void *ami_mem, long int *numnonlinsolvconvfails) const override

reports the number of nonlinear convergence failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numnonlinsolvconvfails** – output array

virtual void **getLastOrder**(const void *ami_ami_mem, int *order) const override

Reports the order of the integration method during the last internal step.

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **order** – output array

virtual void ***getAdjBmem**(void *ami_mem, int which) const override

Retrieves the solver memory instance for the backward problem.

Parameters

- **which** – identifier of the backwards problem
- **ami_mem** – pointer to the forward solver memory instance

Returns A (void *) pointer to the CVODES memory allocated for the backward problem.

virtual void **init**(*realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const override

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **initSteadystate**(const *realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const override

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **sensInit1**(const *AmiVectorArray* &sx0, const *AmiVectorArray* &sdx0) const override

Initializes the forward sensitivities.

Parameters

- **sx0** – initial states sensitivities
- **sdx0** – initial derivative states sensitivities

virtual void **binit**(int which, *realtype* tf, const *AmiVector* &xB0, const *AmiVector* &dxB0) const override

Initialize the adjoint states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **tf** – final timepoint
- **xB0** – initial adjoint state
- **dxB0** – initial adjoint derivative state

virtual void **qbinit**(int which, const *AmiVector* &xQB0) const override

Initialize the quadrature states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **xQB0** – initial adjoint quadrature state

virtual void **rootInit**(int ne) const override

Initializes the rootfinding for events.

Parameters **ne** – number of different events

virtual void **setDenseJacFn**() const override

Set the dense Jacobian function.

virtual void **setSparseJacFn**() const override

sets the sparse Jacobian function

virtual void **setBandJacFn**() const override

sets the banded Jacobian function

virtual void **setJacTimesVecFn**() const override

sets the Jacobian vector multiplication function

virtual void **setDenseJacFnB**(int which) const override

sets the dense Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setSparseJacFnB**(int which) const override

sets the sparse Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setBandJacFnB**(int which) const override

sets the banded Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setJacTimesVecFnB**(int which) const override

sets the Jacobian vector multiplication function

Parameters **which** – identifier of the backwards problem

virtual void **setSparseJacFn_ss**() const override

sets the sparse Jacobian function for backward steady state case

Friends

template<class **Archive**>
friend void **serialize**(*Archive* &ar, *CVodeSolver* &s, unsigned int)

Serialize *amici::CVodeSolver* to boost archive.

Parameters

- **ar** – Archive
- **s** – *Solver* instance to serialize

friend bool **operator==**(const *CVodeSolver* &a, const *CVodeSolver* &b)

Equality operator.

Parameters

- **a** –
- **b** –

Returns Whether a and b are equal

Class ExpData

- Defined in file_include_amici_edata.h

Inheritance Relationships

Base Type

- `public amici::SimulationParameters` (*Class SimulationParameters*)

Class Documentation

class **ExpData** : public amici::*SimulationParameters*

ExpData carries all information about experimental or condition-specific data.

Public Functions

ExpData() = default

default constructor

ExpData(*ExpData* const&) = default

Copy constructor, needs to be declared to be generated in swig.

ExpData(int nytrue, int nztrue, int nmaxevent)

constructor that only initializes dimensions

Parameters

- **nytrue** – Number of observables
- **nztrue** – Number of event outputs
- **nmaxevent** – Maximal number of events to track

ExpData(int nytrue, int nztrue, int nmaxevent, std::vector<*realtype*> ts)

constructor that initializes timepoints from vectors

Parameters

- **nytrue** – Number of observables
- **nztrue** – Number of event outputs
- **nmaxevent** – Maximal number of events to track
- **ts** – Timepoints (dimension: nt)

ExpData(int nytrue, int nztrue, int nmaxevent, std::vector<*realtype*> ts, std::vector<*realtype*> fixedParameters)

constructor that initializes timepoints and fixed parameters from vectors

Parameters

- **nytrue** – Number of observables
- **nztrue** – Number of event outputs
- **nmaxevent** – Maximal number of events to track

- **ts** – Timepoints (dimension: nt)
- **fixedParameters** – *Model* constants (dimension: nk)

ExpData(int nytrue, int nztrue, int nmaxevent, std::vector<*realtype*> ts, std::vector<*realtype*> const &observedData, std::vector<*realtype*> const &observedDataStdDev, std::vector<*realtype*> const &observedEvents, std::vector<*realtype*> const &observedEventsStdDev)

constructor that initializes timepoints and data from vectors

Parameters

- **nytrue** – Number of observables
- **nztrue** – Number of event outputs
- **nmaxevent** – Maximal number of events to track
- **ts** – Timepoints (dimension: nt)
- **observedData** – observed data (dimension: nt x nytrue, row-major)
- **observedDataStdDev** – standard deviation of observed data (dimension: nt x nytrue, row-major)
- **observedEvents** – observed events (dimension: nmaxevents x nztrue, row-major)
- **observedEventsStdDev** – standard deviation of observed events/roots (dimension: nmaxevents x nztrue, row-major)

explicit **ExpData**(*Model* const &model)

constructor that initializes with *Model*

Parameters **model** – pointer to model specification object

ExpData(*ReturnData* const &rdata, *realtype* sigma_y, *realtype* sigma_z)

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** – return data pointer with stored simulation results
- **sigma_y** – scalar standard deviations for all observables
- **sigma_z** – scalar standard deviations for all event observables

ExpData(*ReturnData* const &rdata, std::vector<*realtype*> sigma_y, std::vector<*realtype*> sigma_z)

constructor that initializes with returnData, adds noise according to specified sigmas

Parameters

- **rdata** – return data pointer with stored simulation results
- **sigma_y** – vector of standard deviations for observables (dimension: nytrue or nt x nytrue, row-major)
- **sigma_z** – vector of standard deviations for event observables (dimension: nztrue or nmaxevent x nztrue, row-major)

~ExpData() = default

int **nytrue**() const

number of observables of the non-augmented model

Returns number of observables of the non-augmented model int **nztrue**() const

number of event observables of the non-augmented model

Returns number of event observables of the non-augmented model

int **nmaxevent**() const

maximal number of events to track

Returns maximal number of events to track

int **nt**() const

number of timepoints

Returns number of timepoints

void **setTimepoints**(std::vector<*realtype*> const &ts)

Set function that copies data from input to ExpData::ts.

Parameters **ts** – timepoints

std::vector<*realtype*> const &**getTimepoints**() const

get function that copies data from ExpData::ts to output

Returns ExpData::ts

realtype **getTimepoint**(int it) const

get function that returns timepoint at index

Parameters **it** – timepoint index

Returns timepoint timepoint at index

void **setObservedData**(std::vector<*realtype*> const &observedData)

set function that copies data from input to ExpData::my

Parameters **observedData** – observed data (dimension: nt x nytrue, row-major)

void **setObservedData**(std::vector<*realtype*> const &observedData, int iy)

set function that copies observed data for specific observable

Parameters

- **observedData** – observed data (dimension: nt)
- **iy** – observed data index

bool **isSetObservedData**(int it, int iy) const

get function that checks whether data at specified indices has been set

Parameters

- **it** – time index
- **iy** – observable index

Returns boolean specifying if data was set

std::vector<*realtype*> const &**getObservedData**() const

get function that copies data from ExpData::observedData to output

Returns observed data (dimension: nt x nytrue, row-major) *realtype* const ***getObservedDataPtr**(int it) const

get function that returns a pointer to observed data at index

Parameters **it** – timepoint index

Returns pointer to observed data at index (dimension: nytrue)

void **setObservedDataStdDev**(std::vector<*realtype*> const &observedDataStdDev)

set function that copies data from input to ExpData::observedDataStdDev

Parameters **observedDataStdDev** – standard deviation of observed data (dimension: nt x nytrue, row-major)

void **setObservedDataStdDev**(*realtype* stdDev)

set function that sets all ExpData::observedDataStdDev to the input value

Parameters **stdDev** – standard deviation (dimension: scalar)

void **setObservedDataStdDev**(std::vector<*realtype*> const &observedDataStdDev, int iy)

set function that copies standard deviation of observed data for specific observable

Parameters

- **observedDataStdDev** – standard deviation of observed data (dimension: nt)
- **iy** – observed data index

void **setObservedDataStdDev**(*realtype* stdDev, int iy)

set function that sets all standard deviation of a specific observable to the input value

Parameters

- **stdDev** – standard deviation (dimension: scalar)
- **iy** – observed data index

bool **isSetObservedDataStdDev**(int it, int iy) const

get function that checks whether standard deviation of data at specified indices has been set

Parameters

- **it** – time index
- **iy** – observable index

Returns boolean specifying if standard deviation of data was set

std::vector<*realtype*> const &**getObservedDataStdDev**() const

get function that copies data from ExpData::observedDataStdDev to output

Returns standard deviation of observed data

realtype const ***getObservedDataStdDevPtr**(int it) const

get function that returns a pointer to standard deviation of observed data at index

Parameters **it** – timepoint index

Returns pointer to standard deviation of observed data at index

void **setObservedEvents**(std::vector<*realtype*> const &observedEvents)

set function that copies observed event data from input to ExpData::observedEvents

Parameters **observedEvents** – observed data (dimension: nmaxevent x nztrue, row-major) void **setObservedEvents**(std::vector<*realtype*> const &observedEvents, int iz)

set function that copies observed event data for specific event observable

Parameters

- **observedEvents** – observed data (dimension: nmaxevent)
- **iz** – observed event data index

bool **isSetObservedEvents**(int ie, int iz) const

get function that checks whether event data at specified indices has been set

Parameters

- **ie** – event index
- **iz** – event observable index

Returns boolean specifying if data was set

std::vector<*realtype*> const &**getObservedEvents**() const

get function that copies data from ExpData::mz to output

Returns observed event data

realtype const ***getObservedEventsPtr**(int ie) const

get function that returns a pointer to observed data at ieth occurrence

Parameters **ie** – event occurrence

Returns pointer to observed event data at ieth occurrence

void **setObservedEventsStdDev**(std::vector<*realtype*> const &observedEventsStdDev)

set function that copies data from input to ExpData::observedEventsStdDev

Parameters **observedEventsStdDev** – standard deviation of observed event data

void **setObservedEventsStdDev**(*realtype* stdDev)

set function that sets all ExpData::observedDataStdDev to the input value

Parameters **stdDev** – standard deviation (dimension: scalar)

void **setObservedEventsStdDev**(std::vector<*realtype*> const &observedEventsStdDev, int iz)

set function that copies standard deviation of observed data for specific observable

Parameters

- **observedEventsStdDev** – standard deviation of observed data (dimension: nmaxevent)
- **iz** – observed data index

void **setObservedEventsStdDev**(*realtype* stdDev, int iz)

set function that sets all standard deviation of a specific observable to the input value

Parameters

- **stdDev** – standard deviation (dimension: scalar)
- **iz** – observed data index

bool **isSetObservedEventsStdDev**(int ie, int iz) const

get function that checks whether standard deviation of even data at specified indices has been set

Parameters

- **ie** – event index

- **iz** – event observable index

Returns boolean specifying if standard deviation of event data was set

std::vector<*realtype*> const &**getObservedEventsStdDev**() const

get function that copies data from ExpData::observedEventsStdDev to output

Returns standard deviation of observed event data

realtype const ***getObservedEventsStdDevPtr**(int ie) const

get function that returns a pointer to standard deviation of observed event data at ie-th occurrence

Parameters **ie** – event occurrence

Returns pointer to standard deviation of observed event data at ie-th occurrence

Public Members

std::string **id**

Arbitrary (not necessarily unique) identifier.

Protected Functions

void **applyDimensions**()

resizes observedData, observedDataStdDev, observedEvents and observedEventsStdDev

void **applyDataDimension**()

resizes observedData and observedDataStdDev

void **applyEventDimension**()

resizes observedEvents and observedEventsStdDev

void **checkDataDimension**(std::vector<*realtype*> const &input, char const *fieldname) const

checker for dimensions of input observedData or observedDataStdDev

Parameters

- **input** – vector input to be checked
- **fieldname** – name of the input

void **checkEventsDimension**(std::vector<*realtype*> const &input, char const *fieldname) const

checker for dimensions of input observedEvents or observedEventsStdDev

Parameters

- **input** – vector input to be checked
- **fieldname** – name of the input

Protected Attributes

int **nytrue_** = {0}

number of observables

int **nztrue_** = {0}

number of event observables

int **nmaxevent_** = {0}

maximal number of event occurrences

std::vector<*realtype*> **observed_data_**

observed data (dimension: nt x nytrue, row-major)

std::vector<*realtype*> **observed_data_std_dev_**

standard deviation of observed data (dimension: nt x nytrue, row-major)

std::vector<*realtype*> **observed_events_**

observed events (dimension: nmaxevents x nztrue, row-major)

std::vector<*realtype*> **observed_events_std_dev_**

standard deviation of observed events/roots (dimension: nmaxevents x nztrue, row-major)

Friends

inline friend bool **operator==**(*ExpData* const &lhs, *ExpData* const &rhs)

Equality operator.

Parameters

- **lhs** – some object
- **rhs** – another object

Returns `true`, if both arguments are equal; `false` otherwise.

Class FinalStateStorer

- Defined in file_include_amici_forwardproblem.h

Inheritance Relationships

Base Type

- `public amici::ContextManager` (*Class ContextManager*)

Class Documentation

class **FinalStateStorer** : public amici::*ContextManager*

stores the stimulation state when it goes out of scope

Public Functions

inline explicit **FinalStateStorer**(*ForwardProblem* *fwd)

constructor, attaches problem pointer

Parameters **fwd** – problem from which the simulation state is to be stored

FinalStateStorer &**operator=**(*FinalStateStorer* const &other) = delete

inline **~FinalStateStorer**()

destructor, stores simulation state

Class ForwardProblem

- Defined in file_include_amici_forwardproblem.h

Class Documentation

class **ForwardProblem**

The *ForwardProblem* class groups all functions for solving the forward problem.

Public Functions

ForwardProblem(*ExpData* const *edata, *Model* *model, *Solver* *solver, *SteadystateProblem* const *preeq)

Constructor.

Parameters

- **edata** – pointer to *ExpData* instance
- **model** – pointer to *Model* instance
- **solver** – pointer to *Solver* instance
- **preeq** – preequilibration with which to initialize the forward problem, pass nullptr for no initialization

~ForwardProblem() = default

void **workForwardProblem**()

Solve the forward problem.

If forward sensitivities are enabled this will also compute sensitivities.

void **getAdjointUpdates**(*Model* &model, *ExpData* const &edata)

computes adjoint updates dJydx according to provided model and expdata

Parameters

- **model** – *Model* instance
- **edata** – experimental data

inline *realtype* **getTime**() const

Accessor for t.

Returns t

inline *AmiVector* const &**getState**() const

Accessor for x.

Returns x

inline *AmiVector* const &**getStateDerivative**() const

Accessor for dx.

Returns dx

inline *AmiVectorArray* const &**getStateSensitivity**() const

Accessor for sx.

Returns sx

inline std::vector<*AmiVector*> const &**getStatesAtDiscontinuities**() const

Accessor for x_disc.

Returns x_disc

inline std::vector<*AmiVector*> const &**getRHSAtDiscontinuities**() const

Accessor for xdot_disc.

Returns xdot_disc

inline std::vector<*AmiVector*> const &**getRHSBeforeDiscontinuities**() const

Accessor for xdot_old_disc.

Returns xdot_old_disc

inline std::vector<int> const &**getNumberOfRoots**() const

Accessor for nroots.

Returns nroots

inline std::vector<*realtype*> const &**getDiscontinuities**() const

Accessor for discs.

Returns discs

inline std::vector<std::vector<int>> const &**getRootIndexes**() const

Accessor for rootidx.

Returns rootidx

inline std::vector<*realtype*> const &**getDJydx**() const

Accessor for dJydx.

Returns dJydx

inline std::vector<*realtype*> const &**getDJzdx**() const

Accessor for dJzdx.

Returns dJzdx

inline *AmiVector* ***getStatePointer**()

Accessor for pointer to x.

Returns &x

inline *AmiVector* ***getStateDerivativePointer**()

Accessor for pointer to dx.

Returns &dx

inline *AmiVectorArray* ***getStateSensitivityPointer**()

accessor for pointer to sx

Returns &sx

inline *AmiVectorArray* ***getStateDerivativeSensitivityPointer**()

Accessor for pointer to sdx.

Returns &sdx

inline int **getCurrentTimeIteration**() const

Accessor for it.

Returns it

inline *realtype* **getFinalTime**() const

Returns final time point for which simulations are available.

Returns time point

inline int **getEventCounter**() const

Returns maximal event index for which simulations are available.

Returns index

inline int **getRootCounter**() const

Returns maximal event index for which the timepoint is available.

Returns index

inline *SimulationState* const &**getSimulationStateTimepoint**(int it) const

Retrieves the carbon copy of the simulation state variables at the specified timepoint index.

Parameters **it** – timepoint index

Returns state

inline *SimulationState* const &**getSimulationStateEvent**(int iroot) const

Retrieves the carbon copy of the simulation state variables at the specified event index.

Parameters **iroot** – event index

Returns *SimulationState*

inline *SimulationState* const &**getInitialSimulationState**() const

Retrieves the carbon copy of the simulation state variables at the initial timepoint.

Returns *SimulationState*

inline *SimulationState* const &**getFinalSimulationState**() const

Retrieves the carbon copy of the simulation state variables at the final timepoint (or when simulation failed)

Returns *SimulationState*

Public Members

Model ***model**

pointer to model instance

Solver ***solver**

pointer to solver instance

ExpData const ***edata**

pointer to experimental data instance

Class IDAException

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **IDAException** : public amici::*AmiException*

ida exception handler class

Public Functions

IDAException(int error_code, char const *function)

Constructor.

Parameters

- **error_code** – error code returned by ida function
- **function** – ida function name

Class IDASolver

- Defined in file_include_amici_solver_idas.h

Inheritance Relationships

Base Type

- `public amici::Solver` (*Class Solver*)

Class Documentation

class **IDASolver** : public amici::*Solver*

The *IDASolver* class is a wrapper around the SUNDIALS IDAS solver.

Public Functions

~IDASolver() override = default

virtual *Solver* ***clone**() const override

Clone this instance.

Returns The clone

virtual void **reInitPostProcessF**(*realtype* tnext) const override

reInitPostProcessF postprocessing of the solver memory after a discontinuity in the forward problem

Parameters **tnext** – next timepoint (defines integration direction)

virtual void **reInitPostProcessB**(*realtype* tnext) const override

reInitPostProcessB postprocessing of the solver memory after a discontinuity in the backward problem

Parameters **tnext** – next timepoint (defines integration direction)

virtual void **reInit**(*realtype* t0, const *AmiVector* &yy0, const *AmiVector* &yp0) const override

Reinitializes the states in the solver after an event occurrence.

Parameters

- **t0** – reinitialization timepoint
- **yy0** – initial state variables
- **yp0** – initial derivative state variables (DAE only)

virtual void **sensReInit**(const *AmiVectorArray* &yyS0, const *AmiVectorArray* &ypS0) const override

Reinitializes the state sensitivities in the solver after an event occurrence.

Parameters

- **yyS0** – new state sensitivity
- **ypS0** – new derivative state sensitivities (DAE only)

virtual void **sensToggleOff**() const override

Switches off computation of state sensitivities without deallocating the memory for sensitivities.

virtual void **reInitB**(int which, *realtype* tB0, const *AmiVector* &yyB0, const *AmiVector* &ypB0) const override

Reinitializes the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **tB0** – reinitialization timepoint
- **yyB0** – new adjoint state
- **ypB0** – new adjoint derivative state

virtual void **quadReInitB**(int which, const *AmiVector* &yQB0) const override

Reinitialize the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **yQB0** – new adjoint quadrature state

virtual void **quadSStolerancesB**(int which, *realtype* reltolQB, *realtype* abstolQB) const override

sets relative and absolute tolerances for the quadrature backward problem

Parameters

- **which** – identifier of the backwards problem
- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual void **quadSStolerances**(*realtype* reltolQ, *realtype* abstolQ) const override

sets relative and absolute tolerances for the quadrature problem

Parameters

- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual int **solve**(*realtype* tout, int itask) const override

Solves the forward problem until a predefined timepoint.

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP

Returns status flag indicating success of execution

virtual int **solveF**(*realtype* tout, int itask, int *ncheckPtr) const override

Solves the forward problem until a predefined timepoint (adjoint only)

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP
- **ncheckPtr** – pointer to a number that counts the internal checkpoints

Returns status flag indicating success of execution

virtual void **solveB**(*realtype* tBout, int itaskB) const override

Solves the backward problem until a predefined timepoint (adjoint only)

Parameters

- **tBout** – timepoint until which simulation should be performed
- **itaskB** – task identifier, can be CV_NORMAL or CV_ONE_STEP

virtual void **getRootInfo**(int *rootsfound) const override

getRootInfo extracts information which event occurred

Parameters **rootsfound** – array with flags indicating whether the respective event occurred

virtual void **getDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getSens**() const override

extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

virtual void **getSensDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getB**(int which) const override

extracts the adjoint state at the current timepoint from solver memory and writes it to the xB member variable

Parameters **which** – index of the backwards problem

virtual void **getDkyB**(*realtype* t, int k, int which) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order
- **which** – index of backward problem

virtual void **getQuadB**(int which) const override

extracts the adjoint quadrature state at the current timepoint from solver memory and writes it to the xQB member variable

Parameters **which** – index of the backwards problem virtual void **getQuadDkyB**(*realtype* t, int k, int which) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order
- **which** – index of backward problem

virtual void **getQuad**(*realtype* &t) const override

extracts the quadrature at the current timepoint from solver memory and writes it to the xQ member variable

Parameters **t** – timepoint for quadrature extraction

virtual void **getQuadDky**(*realtype* t, int k) const override

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **calcIC**(*realtype* tout1) const override

Calculates consistent initial conditions, assumes initial states to be correct (DAE only)

Parameters **tout1** – next timepoint to be computed (sets timescale)

virtual void **calcICB**(int which, *realtype* tout1) const override

Calculates consistent initial conditions for the backwards problem, assumes initial states to be correct (DAE only)

Parameters

- **which** – identifier of the backwards problem
- **tout1** – next timepoint to be computed (sets timescale)

virtual void **setStopTime**(*realtype* tstop) const override

Sets a timepoint at which the simulation will be stopped.

Parameters **tstop** – timepoint until which simulation should be performed

virtual void **turnOffRootFinding**() const override

Disable rootfinding.

virtual const *Model* ***getModel**() const override

Accessor function to the model stored in the user data

Returns user data model

virtual void **setLinearSolver**() const override

Sets the linear solver for the forward problem.

virtual void **setLinearSolverB**(int which) const override

Sets the linear solver for the backward problem.

Parameters **which** – index of the backward problem

virtual void **setNonLinearSolver**() const override

Set the non-linear solver for the forward problem.

virtual void **setNonLinearSolverSens**() const override

Set the non-linear solver for sensitivities.

virtual void **setNonLinearSolverB**(int which) const override

Set the non-linear solver for the backward problem.

Parameters **which** – index of the backward problem

Solver() = default

Default constructor.

Solver(const *Solver* &other)

Solver copy constructor.

Parameters **other** –

Protected Functions

void **reInitPostProcess**(void *ida_mem, *realtype* *t, *AmiVector* *yout, *AmiVector* *ypout, *realtype* tout) const

Postprocessing of the solver memory after a discontinuity.

Parameters

- **ida_mem** – pointer to IDAS solver memory object
- **t** – pointer to integration time
- **yout** – new state vector
- **ypout** – new state derivative vector
- **tout** – anticipated next integration timepoint.

virtual void **allocateSolver**() const override

Create specifies solver method and initializes solver memory for the forward problem.

virtual void **setSStolerances**(*realtype* rtol, *realtype* atol) const override

sets scalar relative and absolute tolerances for the forward problem

Parameters

- **rtol** – relative tolerances
- **atol** – absolute tolerances

virtual void **setSensSStolerances**(*realtype* rtol, const *realtype* *atol) const override

activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

- **rtol** – relative tolerances
- **atol** – array of absolute tolerances for every sensitivity variable

virtual void **setSensErrCon**(bool error_corr) const override

SetSensErrCon specifies whether error control is also enforced for sensitivities for the forward problem

Parameters **error_corr** – activation flag virtual void **setQuadErrConB**(int which, bool flag) const override

Specifies whether error control is also enforced for the backward quadrature problem.

Parameters

- **which** – identifier of the backwards problem
- **flag** – activation flag

virtual void **setQuadErrCon**(bool flag) const override

Specifies whether error control is also enforced for the forward quadrature problem.

Parameters **flag** – activation flag

virtual void **setErrHandlerFn**() const override

Attaches the error handler function (errMsgIdAndTxt) to the solver.

virtual void **setUserData**() const override

Attaches the user data to the forward problem.

virtual void **setUserDataB**(int which) const override

attaches the user data to the backward problem

Parameters **which** – identifier of the backwards problem

virtual void **setMaxNumSteps**(long int mxsteps) const override

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters **mxsteps** – number of steps

virtual void **setStabLimDet**(int stldet) const override

activates stability limit detection for the forward problem

Parameters **stldet** – flag for stability limit detection (TRUE or FALSE)

virtual void **setStabLimDetB**(int which, int stldet) const override

activates stability limit detection for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **stldet** – flag for stability limit detection (TRUE or FALSE)

virtual void **setId**(const *Model* *model) const override

specify algebraic/differential components (DAE only)

Parameters **model** – model specification

virtual void **setSuppressAlg**(bool flag) const override

deactivates error control for algebraic components (DAE only)

Parameters **flag** – deactivation flag void **resetState**(void *ida_mem, *const_N_Vector* yy0, *const_N_Vector* yp0) const

resetState reset the IDAS solver to restart integration after a rhs discontinuity.

Parameters

- **ida_mem** – pointer to IDAS solver memory object
- **yy0** – new state vector
- **yp0** – new state derivative vector

virtual void **setSensParams**(const *realtype* *p, const *realtype* *pbar, const int *plist) const override

specifies the scaling and indexes for sensitivity computation

Parameters

- **p** – parameters
- **pbar** – parameter scaling constants
- **plist** – parameter index list

virtual void **adjInit**() const override

initializes the adjoint problem

virtual void **quadInit**(const *AmiVector* &xQ0) const override

initializes the quadratures

Parameters **xQ0** – vector with initial values for xQ

virtual void **allocateSolverB**(int *which) const override

Specifies solver method and initializes solver memory for the backward problem.

Parameters **which** – identifier of the backwards problem

virtual void **setMaxNumStepsB**(int which, long int mxstepsB) const override

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters

- **which** – identifier of the backwards problem
- **mxstepsB** – number of steps

virtual void **setSStolerancesB**(int which, *realtype* relTolB, *realtype* absTolB) const override

sets relative and absolute tolerances for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **relTolB** – relative tolerances
- **absTolB** – absolute tolerances

virtual void **diag**() const override

attaches a diagonal linear solver to the forward problem virtual void **diagB**(int which) const override

attaches a diagonal linear solver to the backward problem

Parameters **which** – identifier of the backwards problem

virtual void **getNumSteps**(const void *ami_mem, long int *numsteps) const override

reports the number of solver steps

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numsteps** – output array

virtual void **getNumRhsEvals**(const void *ami_mem, long int *numrhsevals) const override

reports the number of right hand evaluations

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numrhsevals** – output array

virtual void **getNumErrTestFails**(const void *ami_mem, long int *numerrtestfails) const override

reports the number of local error test failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numerrtestfails** – output array

virtual void **getNumNonlinSolvConvFails**(const void *ami_mem, long int *numnonlinsolvconvfails) const override

reports the number of nonlinear convergence failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numnonlinsolvconvfails** – output array

virtual void **getLastOrder**(const void *ami_mem, int *order) const override

Reports the order of the integration method during the last internal step.

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **order** – output array

virtual void ***getAdjBmem**(void *ami_mem, int which) const override

Retrieves the solver memory instance for the backward problem.

Parameters

- **which** – identifier of the backwards problem
- **ami_mem** – pointer to the forward solver memory instance

Returns A (void *) pointer to the CVODES memory allocated for the backward problem.

virtual void **init**(*realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const override

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **initSteadystate**(const *realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const override

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **sensInit1**(const *AmiVectorArray* &sx0, const *AmiVectorArray* &sdx0) const override

Initializes the forward sensitivities.

Parameters

- **sx0** – initial states sensitivities
- **sdx0** – initial derivative states sensitivities

virtual void **binit**(int which, *realtype* tf, const *AmiVector* &xB0, const *AmiVector* &dxB0) const override

Initialize the adjoint states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **tf** – final timepoint
- **xB0** – initial adjoint state
- **dxB0** – initial adjoint derivative state

virtual void **qbinit**(int which, const *AmiVector* &xQB0) const override

Initialize the quadrature states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **xQB0** – initial adjoint quadrature state

virtual void **rootInit**(int ne) const override

Initializes the rootfinding for events.

Parameters **ne** – number of different events

virtual void **setDenseJacFn**() const override

Set the dense Jacobian function.

virtual void **setSparseJacFn**() const override
sets the sparse Jacobian function

virtual void **setBandJacFn**() const override
sets the banded Jacobian function

virtual void **setJacTimesVecFn**() const override
sets the Jacobian vector multiplication function

virtual void **setDenseJacFnB**(int which) const override
sets the dense Jacobian function

Parameters which – identifier of the backwards problem

virtual void **setSparseJacFnB**(int which) const override
sets the sparse Jacobian function

Parameters which – identifier of the backwards problem

virtual void **setBandJacFnB**(int which) const override
sets the banded Jacobian function

Parameters which – identifier of the backwards problem

virtual void **setJacTimesVecFnB**(int which) const override
sets the Jacobian vector multiplication function

Parameters which – identifier of the backwards problem

virtual void **setSparseJacFn_ss**() const override
sets the sparse Jacobian function for backward steady state case

Class IntegrationFailure

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **IntegrationFailure** : public amici::*AmiException*

Integration failure exception for the forward problem.

This exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user

Public Functions

IntegrationFailure(int code, *realtype* t)

Constructor.

Parameters

- **code** – error code returned by cvode/ida
- **t** – time of integration failure

Public Members

int **error_code**

error code returned by cvodes/idas

realtype **time**

time of integration failure

Class IntegrationFailureB

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **IntegrationFailureB** : public amici::*AmiException*

Integration failure exception for the backward problem.

This exception should be thrown when an integration failure occurred for this exception we can assume that we can recover from the exception and return a solution struct to the user

Public Functions

IntegrationFailureB(int code, *realtype* t)

Constructor.

Parameters

- **code** – error code returned by cvode/ida
- **t** – time of integration failure

Public Members

int **error_code**

error code returned by cvode/ida

realtype **time**

time of integration failure

Class Logger

- Defined in file_include_amici_logging.h

Class Documentation

class **Logger**

A logger, holding a list of error messages.

Public Functions

Logger() = default

void **log**(*LogSeverity* severity, std::string const &identifier, std::string const &message)

Add a log entry.

Parameters

- **severity** – Severity level
- **identifier** – Short identifier for the logged event
- **message** – A more detailed message

void **log**(*LogSeverity* severity, std::string const &identifier, char const *format, ...)

Add a log entry with printf-like message formatting.

Parameters

- **severity** – Severity level
- **identifier** – Short identifier for the logged event
- **format** – printf format string

Public Members

std::vector<*LogItem*> **items**

The log items

Class Model

- Defined in file_include_amici_model.h

Inheritance Relationships

Base Types

- `public amici::AbstractModel` (*Class AbstractModel*)
- `public amici::ModelDimensions` (*Struct ModelDimensions*)

Derived Types

- `public amici::Model_DAE` (*Class Model_DAE*)
- `public amici::Model_ODE` (*Class Model_ODE*)

Class Documentation

class **Model** : public amici::*AbstractModel*, public amici::*ModelDimensions*

The *Model* class represents an AMICI ODE/DAE model.

The model can compute various model related quantities based on symbolically generated code.

Subclassed by *amici::Model_DAE*, *amici::Model_ODE*

Public Functions

Model() = default

Default constructor

Model(*ModelDimensions* const &model_dimensions, *SimulationParameters* simulation_parameters, amici::*SecondOrderMode* o2mode, std::vector<amici::*realtype*> idlist, std::vector<int> z2event, bool pythonGenerated = false, int ndxdotdp_explicit = 0, int ndxdotdx_explicit = 0, int w_recursion_depth = 0)

Constructor with model dimensions.

Parameters

- **model_dimensions** – *Model* dimensions
- **simulation_parameters** – Simulation parameters
- **o2mode** – Second order sensitivity mode

- **idlist** – Indexes indicating algebraic components (DAE only)
- **z2event** – Mapping of event outputs to events
- **pythonGenerated** – Flag indicating matlab or python wrapping
- **ndxdotdp_explicit** – Number of nonzero elements in dxdotdp_explicit
- **ndxdotdx_explicit** – Number of nonzero elements in dxdotdx_explicit
- **w_recursion_depth** – Recursion depth of fw

~Model() override = default

Destructor.

Model &**operator=**(*Model* const &other) = delete

Copy assignment is disabled until const members are removed.

Parameters **other** – Object to copy from

Returns

virtual *Model* ***clone**() const = 0

Clone this instance.

Returns The clone

void **initialize**(*AmiVector* &x, *AmiVector* &dx, *AmiVectorArray* &sx, *AmiVectorArray* &sdx, bool computeSensitivities, std::vector<int> &roots_found)

Initialize model properties.

Parameters

- **x** – Reference to state variables
- **dx** – Reference to time derivative of states (DAE only)
- **sx** – Reference to state variable sensitivities
- **sdx** – Reference to time derivative of state sensitivities (DAE only)
- **computeSensitivities** – Flag indicating whether sensitivities are to be computed
- **roots_found** – boolean indicators indicating whether roots were found at t0 by this fun

void **initializeB**(*AmiVector* &xB, *AmiVector* &dxB, *AmiVector* &xQB, bool posteq) const

Initialize model properties.

Parameters

- **xB** – Adjoint state variables
- **dxB** – Time derivative of adjoint states (DAE only)
- **xQB** – Adjoint quadratures
- **posteq** – Flag indicating whether postequilibration was performed

void **initializeStates**(*AmiVector* &x)

Initialize initial states.

Parameters **x** – State vector to be initialized void **initializeStateSensitivities**(*AmiVectorArray* &sx, const *AmiVector* &x)

Initialize initial state sensitivities.

Parameters

- **sx** – Reference to state variable sensitivities
- **x** – Reference to state variables

void **initEvents**(const *AmiVector* &x, const *AmiVector* &dx, std::vector<int> &roots_found)

Initialize the Heaviside variables `h` at the initial time `t0`.

Heaviside variables activate/deactivate on event occurrences.

Parameters

- **x** – Reference to state variables
- **dx** – Reference to time derivative of states (DAE only)
- **roots_found** – boolean indicators indicating whether roots were found at t0 by this fun

int **nplist**() const

Get number of parameters wrt to which sensitivities are computed.

Returns Length of sensitivity index vector

int **np**() const

Get total number of model parameters.

Returns Length of parameter vector

int **nk**() const

Get number of constants.

Returns Length of constant vector

int **ncl**() const

Get number of conservation laws.

Returns Number of conservation laws (i.e., difference between `nx_rdata` and `nx_solver`).

int **nx_reinit**() const

Get number of solver states subject to reinitialization.

Returns *Model* member `nx_solver_reinit`

const double ***k**() const

Get fixed parameters.

Returns Pointer to constants array

int **nMaxEvent**() const

Get maximum number of events that may occur for each type.

Returns Maximum number of events that may occur for each type

void **setNMaxEvent**(int nmaxevent)

Set maximum number of events that may occur for each type.

Parameters **nmaxevent** – Maximum number of events that may occur for each type int **nt**() const

Get number of timepoints.

Returns Number of timepoints

std::vector<*ParameterScaling*> const &**getParameterScale**() const

Get parameter scale for each parameter.

Returns Vector of parameter scales

void **setParameterScale**(*ParameterScaling* pscale)

Set parameter scale for each parameter.

NOTE: Resets initial state sensitivities.

Parameters **pscale** – Scalar parameter scale to be set for all parameters

void **setParameterScale**(const std::vector<*ParameterScaling*> &pscaleVec)

Set parameter scale for each parameter.

NOTE: Resets initial state sensitivities.

Parameters **pscaleVec** – Vector of parameter scales

std::vector<*realtype*> const &**getUnscaledParameters**() const

Get parameters with transformation according to parameter scale applied.

Returns Unscaled parameters

std::vector<*realtype*> const &**getParameters**() const

Get parameter vector.

Returns The user-set parameters (see also `Model::getUnscaledParameters`)

realtype **getParameterById**(std::string const &par_id) const

Get value of first model parameter with the specified ID.

Parameters **par_id** – Parameter ID

Returns Parameter value

realtype **getParameterByName**(std::string const &par_name) const

Get value of first model parameter with the specified name.

Parameters **par_name** – Parameter name

Returns Parameter value

void **setParameters**(std::vector<*realtype*> const &p)

Set the parameter vector.

Parameters **p** – Vector of parameters

void **setParameterById**(std::map<std::string, *realtype*> const &p, bool ignoreErrors = false)

Set model parameters according to the parameter IDs and mapped values.

Parameters

- **p** – Map of parameters IDs and values
- **ignoreErrors** – Ignore errors such as parameter IDs in p which are not model parameters

void **setParameterById**(std::string const &par_id, *realtype* value)

Set value of first model parameter with the specified ID.

Parameters

- **par_id** – Parameter ID
- **value** – Parameter value

int **setParametersByIdRegex**(std::string const &par_id_regex, *realtype* value)

Set all values of model parameters with IDs matching the specified regular expression.

Parameters

- **par_id_regex** – Parameter ID regex
- **value** – Parameter value

Returns Number of parameter IDs that matched the regex

void **setParameterByName**(std::string const &par_name, *realtype* value)

Set value of first model parameter with the specified name.

Parameters

- **par_name** – Parameter name
- **value** – Parameter value

void **setParameterByName**(std::map<std::string, *realtype*> const &p, bool ignoreErrors = false)

Set model parameters according to the parameter name and mapped values.

Parameters

- **p** – Map of parameters names and values
- **ignoreErrors** – Ignore errors such as parameter names in p which are not model parameters

int **setParametersByNameRegex**(std::string const &par_name_regex, *realtype* value)

Set all values of all model parameters with names matching the specified regex.

Parameters

- **par_name_regex** – Parameter name regex
- **value** – Parameter value

Returns Number of fixed parameter names that matched the regex

std::vector<*realtype*> const &**getFixedParameters**() const

Get values of fixed parameters.

Returns Vector of fixed parameters with same ordering as in *Model::getFixedParameterIds*

realtype **getFixedParameterById**(std::string const &par_id) const

Get value of fixed parameter with the specified ID.

Parameters **par_id** – Parameter ID

Returns Parameter value

realtype **getFixedParameterByName**(std::string const &par_name) const

Get value of fixed parameter with the specified name.

If multiple parameters have the same name, the first parameter with matching name is returned.

Parameters **par_name** – Parameter name

Returns Parameter value

void **setFixedParameters**(std::vector<*realtype*> const &k)

Set values for constants.

Parameters **k** – Vector of fixed parameters

void **setFixedParameterById**(std::string const &par_id, *realtype* value)

Set value of first fixed parameter with the specified ID.

Parameters

- **par_id** – Fixed parameter id
- **value** – Fixed parameter value

int **setFixedParametersByIdRegex**(std::string const &par_id_regex, *realtype* value)

Set values of all fixed parameters with the ID matching the specified regex.

Parameters

- **par_id_regex** – Fixed parameter name regex
- **value** – Fixed parameter value

Returns Number of fixed parameter IDs that matched the regex

void **setFixedParameterByName**(std::string const &par_name, *realtype* value)

Set value of first fixed parameter with the specified name.

Parameters

- **par_name** – Fixed parameter ID
- **value** – Fixed parameter value

int **setFixedParametersByNameRegex**(std::string const &par_name_regex, *realtype* value)

Set value of all fixed parameters with name matching the specified regex.

Parameters

- **par_name_regex** – Fixed parameter name regex
- **value** – Fixed parameter value

Returns Number of fixed parameter names that matched the regex

virtual std::string **getName**() const

Get the model name.

Returns *Model* name

virtual bool **hasParameterNames**() const

Report whether the model has parameter names set.

Returns Boolean indicating whether parameter names were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getParameterNames**() const

Get names of the model parameters.

Returns The parameter names

virtual bool **hasStateNames()** const

Report whether the model has state names set.

Returns Boolean indicating whether state names were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getStateNames()** const

Get names of the model states.

Returns State names

virtual std::vector<std::string> **getStateNamesSolver()** const

Get names of the solver states.

Returns State names

virtual bool **hasFixedParameterNames()** const

Report whether the model has fixed parameter names set.

Returns Boolean indicating whether fixed parameter names were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getFixedParameterNames()** const

Get names of the fixed model parameters.

Returns Fixed parameter names

virtual bool **hasObservableNames()** const

Report whether the model has observable names set.

Returns Boolean indicating whether observable names were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getObservableNames()** const

Get names of the observables.

Returns Observable names

virtual bool **hasExpressionNames()** const

Report whether the model has expression names set.

Returns Boolean indicating whether expression names were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getExpressionNames()** const

Get names of the expressions.

Returns Expression names

virtual bool **hasParameterIds()** const

Report whether the model has parameter IDs set.

Returns Boolean indicating whether parameter IDs were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getParameterIds()** const

Get IDs of the model parameters.

Returns Parameter IDs

virtual bool **hasStateIds**() const

Report whether the model has state IDs set.

Returns Boolean indicating whether state IDs were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getStateIds**() const

Get IDs of the model states.

Returns State IDs

virtual std::vector<std::string> **getStateIdsSolver**() const

Get IDs of the solver states.

Returns State IDs

virtual bool **hasFixedParameterIds**() const

Report whether the model has fixed parameter IDs set.

Returns Boolean indicating whether fixed parameter IDs were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getFixedParameterIds**() const

Get IDs of the fixed model parameters.

Returns Fixed parameter IDs

virtual bool **hasObservableIds**() const

Report whether the model has observable IDs set.

Returns Boolean indicating whether observable ids were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getObservableIds**() const

Get IDs of the observables.

Returns Observable IDs

virtual bool **hasExpressionIds**() const

Report whether the model has expression IDs set.

Returns Boolean indicating whether expression ids were set. Also returns `true` if the number of corresponding variables is just zero.

virtual std::vector<std::string> **getExpressionIds**() const

Get IDs of the expression.

Returns Expression IDs

virtual bool **hasQuadraticLLH**() const

Checks whether the defined noise model is gaussian, i.e., the nllh is quadratic.

Returns boolean flag

std::vector<*realtype*> const &**getTimepoints**() const

Get the timepoint vector.

Returns Timepoint vector

realtype **getTimepoint**(int it) const

Get simulation timepoint for time index `it`.

Parameters **it** – Time index

Returns Timepoint

void **setTimepoints**(std::vector<*realtype*> const &ts)

Set the timepoint vector.

Parameters **ts** – New timepoint vector

double **t0**() const

Get simulation start time.

Returns Simulation start time

void **setT0**(double t0)

Set simulation start time.

Parameters **t0** – Simulation start time

std::vector<bool> const &**getStateIsNonNegative**() const

Get flags indicating whether states should be treated as non-negative.

Returns Vector of flags

void **setStateIsNonNegative**(std::vector<bool> const &stateIsNonNegative)

Set flags indicating whether states should be treated as non-negative.

Parameters **stateIsNonNegative** – Vector of flags

void **setAllStatesNonNegative**()

Set flags indicating that all states should be treated as non-negative.

inline *ModelState* const &**getModelState**() const

Get the current model state.

Returns Current model state

inline void **setModelState**(*ModelState* const &state)

Set the current model state.

Parameters **state** – *Model* state

inline void **setMinimumSigmaResiduals**(double min_sigma)

Sets the estimated lower boundary for sigma_y. When :meth:`setAddSigmaResiduals` is activated, this lower boundary must ensure that log(sigma) + min_sigma > 0.

Parameters **min_sigma** – lower boundary

inline *realtype* **getMinimumSigmaResiduals**() const

Gets the specified estimated lower boundary for sigma_y.

Returns lower boundary

inline void **setAddSigmaResiduals**(bool sigma_res)

Specifies whether residuals should be added to account for parameter dependent sigma.

If set to true, additional residuals of the form $\sqrt{\log(\sigma) + C}$ will be added. This enables least-squares optimization for variables with Gaussian noise assumption and parameter dependent standard deviation sigma. The constant C can be set via :meth:`setMinimumSigmaResiduals`.

Parameters **sigma_res** – if true, additional residuals are added

inline bool **getAddSigmaResiduals**() const

Checks whether residuals should be added to account for parameter dependent sigma.

Returns sigma_res

std::vector<int> const &**getParameterList**() const

Get the list of parameters for which sensitivities are computed.

Returns List of parameter indices

int **plist**(int pos) const

Get entry in parameter list by index.

Parameters **pos** – Index in sensitivity parameter list

Returns Index in parameter list

void **setParameterList**(std::vector<int> const &plist)

Set the list of parameters for which sensitivities are to be computed.

NOTE: Resets initial state sensitivities.

Parameters **plist** – List of parameter indices

std::vector<*realtype*> **getInitialStates**()

Get the initial states.

Returns Initial state vector

void **setInitialStates**(std::vector<*realtype*> const &x0)

Set the initial states.

Parameters **x0** – Initial state vector

bool **hasCustomInitialStates**() const

Return whether custom initial states have been set.

Returns `true` if has custom initial states, otherwise `false`

std::vector<*realtype*> **getInitialStateSensitivities**()

Get the initial states sensitivities.

Returns vector of initial state sensitivities

void **setInitialStateSensitivities**(std::vector<*realtype*> const &sx0)

Set the initial state sensitivities.

Parameters **sx0** – vector of initial state sensitivities with chainrule applied. This could be a slice of *ReturnData::sx* or *ReturnData::sx0*

bool **hasCustomInitialStateSensitivities**() const

Return whether custom initial state sensitivities have been set.

Returns `true` if has custom initial state sensitivities, otherwise `false`.

void **setUnscaledInitialStateSensitivities**(std::vector<*realtype*> const &sx0)

Set the initial state sensitivities.

Parameters **sx0** – Vector of initial state sensitivities without chainrule applied. This could be the readin from a `model.sx0data` saved to HDF5.

void **setSteadyStateSensitivityMode**(*SteadyStateSensitivityMode* mode)

Set the mode how sensitivities are computed in the steadystate simulation.

Parameters **mode** – Steadystate sensitivity mode

SteadyStateSensitivityMode **getSteadyStateSensitivityMode**() const

Gets the mode how sensitivities are computed in the steadystate simulation.

Returns Mode

void **setReinitializeFixedParameterInitialStates**(bool flag)

Set whether initial states depending on fixed parameters are to be reinitialized after preequilibration and presimulation.

Parameters **flag** – Fixed parameters reinitialized?

bool **getReinitializeFixedParameterInitialStates**() const

Get whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation.

Returns flag `true` / `false`

void **requireSensitivitiesForAllParameters**()

Require computation of sensitivities for all parameters p [0..np[in natural order.

NOTE: Resets initial state sensitivities.

void **getExpression**(gsl::span<*realtype*> w, const *realtype* t, const *AmiVector* &x)

Get time-resolved `w`.

Parameters

- **w** – Buffer (shape `nw`)
- **t** – Current timepoint
- **x** – Current state

void **getObservable**(gsl::span<*realtype*> y, const *realtype* t, const *AmiVector* &x)

Get time-resolved observables.

Parameters

- **y** – Buffer (shape `ny`)
- **t** – Current timepoint
- **x** – Current state

virtual *ObservableScaling* **getObservableScaling**(int iy) const

Get scaling type for observable.

Parameters **iy** – observable index

Returns scaling type

void **getObservableSensitivity**(gsl::span<*realtype*> sy, const *realtype* t, const *AmiVector* &x, const *AmiVectorArray* &sx)

Get sensitivity of time-resolved observables.

Total derivative $sy = dydx * sx + dydp$ (only for forward sensitivities).

Parameters

- **sy** – buffer (shape `ny` x `nplist`, row-major)

- **t** – Timpoint
- **x** – State variables
- **sx** – State sensitivities

void **getObservableSigma**(gsl::span<*realtype*> sigmay, const int it, const *ExpData* *edata)

Get time-resolved observable standard deviations.

Parameters

- **sigmay** – Buffer (shape `ny`)
- **it** – Timepoint index
- **edata** – Pointer to experimental data instance (optional, pass `nullptr` to ignore)

void **getObservableSigmaSensitivity**(gsl::span<*realtype*> ssigmay, gsl::span<const *realtype*> sy, const int it, const *ExpData* *edata)

Sensitivity of time-resolved observable standard deviation.

Total derivative (can be used with both adjoint and forward sensitivity).

Parameters

- **ssigmay** – Buffer (shape `ny` x `nplist`, row-major)
- **sy** – Sensitivity of time-resolved observables for current timepoint
- **it** – Timepoint index
- **edata** – Pointer to experimental data instance (optional, pass `nullptr` to ignore)

void **addObservableObjective**(*realtype* &Jy, const int it, const *AmiVector* &x, const *ExpData* &edata)

Add time-resolved measurement negative log-likelihood Jy.

Parameters

- **Jy** – Buffer (shape 1)
- **it** – Timepoint index
- **x** – State variables
- **edata** – Experimental data

void **addObservableObjectiveSensitivity**(std::vector<*realtype*> &sllh, std::vector<*realtype*> &s2llh, const int it, const *AmiVector* &x, const *AmiVectorArray* &sx, const *ExpData* &edata)

Add sensitivity of time-resolved measurement negative log-likelihood Jy.

Parameters

- **sllh** – First-order buffer (shape `nplist`)
- **s2llh** – Second-order buffer (shape `nJ - 1` x `nplist`, row-major)
- **it** – Timepoint index
- **x** – State variables
- **sx** – State sensitivities
- **edata** – Experimental data

void **addPartialObservableObjectiveSensitivity**(std::vector<*realtype*> &sllh, std::vector<*realtype*> &s2llh, const int it, const *AmiVector* &x, const *ExpData* &edata)

Add sensitivity of time-resolved measurement negative log-likelihood Jy.

Partial derivative (to be used with adjoint sensitivities).

Parameters

- **sllh** – First order output buffer (shape `nplist`)
- **s2llh** – Second order output buffer (shape `nJ - 1` x `nplist`, row-major)
- **it** – Timepoint index
- **x** – State variables
- **edata** – Experimental data

void **getAdjointStateObservableUpdate**(gsl::span<*realtype*> dJydx, const int it, const *AmiVector* &x, const *ExpData* &edata)

Get state sensitivity of the negative loglikelihood Jy, partial derivative (to be used with adjoint sensitivities).

Parameters

- **dJydx** – Output buffer (shape `nJ` x `nx_solver`, row-major)
- **it** – Timepoint index
- **x** – State variables
- **edata** – Experimental data instance

void **getEvent**(gsl::span<*realtype*> z, const int ie, const *realtype* t, const *AmiVector* &x)

Get event-resolved observables.

Parameters

- **z** – Output buffer (shape `nz`)
- **ie** – Event index
- **t** – Timepoint
- **x** – State variables

void **getEventSensitivity**(gsl::span<*realtype*> sz, const int ie, const *realtype* t, const *AmiVector* &x, const *AmiVectorArray* &sx)

Get sensitivities of event-resolved observables.

Total derivative (only forward sensitivities).

Parameters

- **sz** – Output buffer (shape `nz x nplist`, row-major)
- **ie** – Event index
- **t** – Timepoint
- **x** – State variables
- **sx** – State sensitivities

void **getUnobservedEventSensitivity**(gsl::span<*realtype*> sz, const int ie)

Get sensitivity of `z` at final timepoint.

Ignores sensitivity of timepoint. Total derivative.

Parameters

- **sz** – Output buffer (shape `nz x nplist`, row-major)
- **ie** – Event index

void **getEventRegularization**(gsl::span<*realtype*> rz, const int ie, const *realtype* t, const *AmiVector* &x)

Get regularization for event-resolved observables.

Parameters

- **rz** – Output buffer (shape `nz`)
- **ie** – Event index
- **t** – Timepoint
- **x** – State variables

void **getEventRegularizationSensitivity**(gsl::span<*realtype*> srz, const int ie, const *realtype* t, const *AmiVector* &x, const *AmiVectorArray* &sx)

Get sensitivities of regularization for event-resolved observables.

Total derivative. Only forward sensitivities.

Parameters

- **srz** – Output buffer (shape `nz x nplist`, row-major)
- **ie** – Event index
- **t** – Timepoint
- **x** – State variables
- **sx** – State sensitivities

void **getEventSigma**(gsl::span<*realtype*> sigmaz, const int ie, const int nroots, const *realtype* t, const *ExpData* *edata)

Get event-resolved observable standard deviations.

Parameters

- **sigmaz** – Output buffer (shape `nz`)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **edata** – Pointer to experimental data (optional, pass `nullptr` to ignore)

void **getEventSigmaSensitivity**(gsl::span<*realtype*> ssigmaz, const int ie, const int nroots, const *realtype* t, const *ExpData* *edata)

Get sensitivities of event-resolved observable standard deviations.

Total derivative (only forward sensitivities).

Parameters

- **ssigmaz** – Output buffer (shape `nz x nplist`, row-major)

- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **edata** – Pointer to experimental data (optional, pass `nullptr` to ignore)

void **addEventObjective**(*realtype* &Jz, const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Add event-resolved observable negative log-likelihood.

Parameters

- **Jz** – Output buffer (shape 1)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **x** – State variables
- **edata** – Experimental data

void **addEventObjectiveRegularization**(*realtype* &Jrz, const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Add event-resolved observable negative log-likelihood.

Parameters

- **Jrz** – Output buffer (shape 1)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **x** – State variables
- **edata** – Experimental data

void **addEventObjectiveSensitivity**(std::vector<*realtype*> &sllh, std::vector<*realtype*> &s2llh, const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *AmiVectorArray* &sx, const *ExpData* &edata)

Add sensitivity of time-resolved measurement negative log-likelihood Jy.

Total derivative (to be used with forward sensitivities).

Parameters

- **sllh** – First order buffer (shape `nplist`)
- **s2llh** – Second order buffer (shape `nJ-1` x `nplist`, row-major)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **x** – State variables
- **sx** – State sensitivities
- **edata** – Experimental data

void **addPartialEventObjectiveSensitivity**(std::vector<*realtype*> &sllh, std::vector<*realtype*> &s2llh, const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Add sensitivity of time-resolved measurement negative log-likelihood Jy.

Partial derivative (to be used with adjoint sensitivities).

Parameters

- **sllh** – First order buffer (shape `nplist`)
- **s2llh** – Second order buffer (shape `(nJ-1)` x `nplist`, row-major)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **x** – State variables
- **edata** – Experimental data

void **getAdjointStateEventUpdate**(gsl::span<*realtype*> dJzdx, const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

State sensitivity of the negative loglikelihood Jz.

Partial derivative (to be used with adjoint sensitivities).

Parameters

- **dJzdx** – Output buffer (shape `nJ` x `nx_solver`, row-major)
- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Timepoint
- **x** – State variables
- **edata** – Experimental data

void **getEventTimeSensitivity**(std::vector<*realtype*> &stau, const *realtype* t, const int ie, const *AmiVector* &x, const *AmiVectorArray* &sx)

Sensitivity of event timepoint, total derivative.

Only forward sensitivities.

Parameters

- **stau** – Timepoint sensitivity (shape `nplist`)
- **t** – Timepoint
- **ie** – Event index
- **x** – State variables
- **sx** – State sensitivities

void **addStateEventUpdate**(*AmiVector* &x, const int ie, const *realtype* t, const *AmiVector* &xdot, const *AmiVector* &xdot_old)

Update state variables after event.

Parameters

- **x** – Current state (will be overwritten)
- **ie** – Event index
- **t** – Current timepoint
- **xdot** – Current residual function values
- **xdot_old** – Value of residual function before event

void **addStateSensitivityEventUpdate**(*AmiVectorArray* &sx, const int ie, const *realtype* t, const *AmiVector* &x_old, const *AmiVector* &xdot, const *AmiVector* &xdot_old, const std::vector<*realtype*> &stau)

Update state sensitivity after event.

Parameters

- **sx** – Current state sensitivity (will be overwritten)
- **ie** – Event index
- **t** – Current timepoint
- **x_old** – Current state
- **xdot** – Current residual function values
- **xdot_old** – Value of residual function before event
- **stau** – Timepoint sensitivity, to be computed with `Model::getEventTimeSensitivity`

void **addAdjointStateEventUpdate**(*AmiVector* &xB, const int ie, const *realtype* t, const *AmiVector* &x, const *AmiVector* &xdot, const *AmiVector* &xdot_old)

Update adjoint state after event.

Parameters

- **xB** – Current adjoint state (will be overwritten)
- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state
- **xdot** – Current residual function values
- **xdot_old** – Value of residual function before event

void **addAdjointQuadratureEventUpdate**(*AmiVector* xQB, const int ie, const *realtype* t, const *AmiVector* &x, const *AmiVector* &xB, const *AmiVector* &xdot, const *AmiVector* &xdot_old)

Update adjoint quadratures after event.

Parameters

- **xQB** – Current quadrature state (will be overwritten)
- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state
- **xB** – Current adjoint state
- **xdot** – Current residual function values

- **xdot_old** – Value of residual function before event

void **updateHeaviside**(const std::vector<int> &rootsfound)

Update the Heaviside variables h on event occurrences.

Parameters **rootsfound** – Provides the direction of the zero-crossing, so adding it will give the right update to the Heaviside variables (zero if no root was found)

void **updateHeavisideB**(const int *rootsfound)

Updates the Heaviside variables h on event occurrences in the backward problem.

Parameters **rootsfound** – Provides the direction of the zero-crossing, so adding it will give the right update to the Heaviside variables (zero if no root was found)

int **checkFinite**(gsl::span<const *realtype*> array, *ModelQuantity* model_quantity) const

Check if the given array has only finite elements.

For (1D) spans.

Parameters

- **array** –
- **model_quantity** – The model quantity array corresponds to

Returns

int **checkFinite**(gsl::span<const *realtype*> array, *ModelQuantity* model_quantity, size_t num_cols) const

Check if the given array has only finite elements.

For flattened 2D arrays.

Parameters

- **array** – Flattened matrix
- **model_quantity** – The model quantity array corresponds to
- **num_cols** – Number of columns of the non-flattened matrix

Returns

int **checkFinite**(SUNMatrix m, *ModelQuantity* model_quantity, *realtype* t) const

Check if the given array has only finite elements.

For SUNMatrix.

Parameters

- **m** – Matrix to check
- **model_quantity** – The model quantity m corresponds to
- **t** – current timepoint

Returns

void **setAlwaysCheckFinite**(bool alwaysCheck)

Set whether the result of every call to Model::f* should be checked for finiteness.

Parameters **alwaysCheck** –

bool **getAlwaysCheckFinite**() const

Get setting of whether the result of every call to Model::f* should be checked for finiteness.

Returns that

void **fx0**(*AmiVector* &x)

Compute/get initial states.

Parameters **x** – Output buffer.

void **fx0_fixedParameters**(*AmiVector* &x)

Set only those initial states that are specified via fixed parameters.

Parameters **x** – Output buffer.

void **fsx0**(*AmiVectorArray* &sx, const *AmiVector* &x)

Compute/get initial value for initial state sensitivities.

Parameters

- **sx** – Output buffer for state sensitivities
- **x** – State variables

void **fsx0_fixedParameters**(*AmiVectorArray* &sx, const *AmiVector* &x)

Get only those initial states sensitivities that are affected from *amici::Model::fx0_fixedParameters*.

Parameters

- **sx** – Output buffer for state sensitivities
- **x** – State variables

virtual void **fsdx0**()

Compute sensitivity of derivative initial states sensitivities sdx0.

Only necessary for DAEs.

void **fx_rdata**(*AmiVector* &x_rdata, const *AmiVector* &x_solver)

Expand conservation law for states.

Parameters

- **x_rdata** – Output buffer for state variables with conservation laws expanded (stored in *amici::ReturnData*).
- **x_solver** – State variables with conservation laws applied (solver returns this)

void **fsx_rdata**(*AmiVectorArray* &sx_rdata, const *AmiVectorArray* &sx_solver, const *AmiVector* &x_solver)

Expand conservation law for state sensitivities.

Parameters

- **sx_rdata** – Output buffer for state variables sensitivities with conservation laws expanded (stored in *amici::ReturnData*).
- **sx_solver** – State variables sensitivities with conservation laws applied (solver returns this)
- **x_solver** – State variables with conservation laws applied (solver returns this)

void **setReinitializationStateIdxs**(const std::vector<int> &idxs)

Set indices of states to be reinitialized based on provided constants / fixed parameters.

Parameters **idxs** – Array of state indices

std::vector<int> const &**getReinitializationStateIdxs**() const

Return indices of states to be reinitialized based on provided constants / fixed parameters.

Returns Those indices.

const *AmiVectorArray* &**get_dxdotdp**() const

getter for dxdotdp (matlab generated)

Returns dxdotdp

const *SUNMatrixWrapper* &**get_dxdotdp_full**() const

getter for dxdotdp (python generated)

Returns dxdotdp

virtual void **fdeltaqB**(*realtype* *deltaqB, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *xB)

Model-specific implementation of fdeltaqB.

Parameters

- **deltaqB** – sensitivity update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – sensitivity index
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side
- **xB** – adjoint state

virtual void **fdeltasx**(*realtype* *deltasx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, int ip, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *sx, const *realtype* *stau, const *realtype* *tcl)

Model-specific implementation of fdeltasx.

Parameters

- **deltasx** – sensitivity update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector
- **ip** – sensitivity index
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side

- **sx** – state sensitivity
- **stau** – event-time sensitivity
- **tcl** – total abundances for conservation laws

virtual void **fdeltax**(*realtype* *deltax, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ie, const *realtype* *xdot, const *realtype* *xdot_old)

Model-specific implementation of fdeltax.

Parameters

- **deltax** – state update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side

virtual void **fdeltaxB**(*realtype* *deltaxB, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ie, const *realtype* *xdot, const *realtype* *xdot_old, const *realtype* *xB)

Model-specific implementation of fdeltaxB.

Parameters

- **deltaxB** – adjoint state update
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ie** – event index
- **xdot** – new model right hand side
- **xdot_old** – previous model right hand side
- **xB** – current adjoint state

virtual void **fdJrzdsigma**(*realtype* *dJrzdsigma, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *rz, const *realtype* *sigmaz)

Model-specific implementation of fdJrzdsigma.

Parameters

- **dJrzdsigma** – Sensitivity of event penalization Jrz w.r.t. standard deviation sigmaz
- **iz** – event output index
- **p** – parameter vector

- **k** – constant vector
- **rz** – model root output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fdJrzdz**(*realtype* *dJrzdz, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *rz, const *realtype* *sigmaz)

Model-specific implementation of fdJrzdz.

Parameters

- **dJrzdz** – partial derivative of event penalization Jrz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **rz** – model root output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fdJydsigma**(*realtype* *dJydsigma, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fdJydsigma.

Parameters

- **dJydsigma** – Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigmay
- **iy** – output index
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurement at timepoint

virtual void **fdJydy**(*realtype* *dJydy, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fdJydy.

Parameters

- **dJydy** – partial derivative of time-resolved measurement negative log-likelihood Jy
- **iy** – output index
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurement at timepoint

virtual void **fdJydy_colptrs**(*SUNMatrixWrapper* &dJydy, int index)

Model-specific implementation of fdJydy colptrs.

Parameters

- **dJydy** – sparse matrix to which colptrs will be written
- **index** – ytrue index

virtual void **fdJydy_rowvals**(*SUNMatrixWrapper* &dJydy, int index)

Model-specific implementation of fdJydy rowvals.

Parameters

- **dJydy** – sparse matrix to which rowvals will be written
- **index** – `ytrue` index

virtual void **fdJzdsigma**(*realtype* *dJzdsigma, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fdJzdsigma.

Parameters

- **dJzdsigma** – Sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurement at timepoint

virtual void **fdJzdz**(*realtype* *dJzdz, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fdJzdz.

Parameters

- **dJzdz** – partial derivative of event measurement negative log-likelihood Jz
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurement at timepoint

virtual void **fdrzdp**(*realtype* *drzdp, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip)

Model-specific implementation of fdrzdp.

Parameters

- **drzdp** – partial derivative of root output rz w.r.t. model parameters p

- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested

virtual void **fdrzdx**(*realtype* *drzdx, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fdrzdx.

Parameters

- **drzdx** – partial derivative of root output rz w.r.t. model states x
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fdsigmaydp**(*realtype* *dsigmaydp, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y, int ip)

Model-specific implementation of fdsigmaydp.

Parameters

- **dsigmaydp** – partial derivative of standard deviation of measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t
- **ip** – sensitivity index

virtual void **fdsigmaydy**(*realtype* *dsigmaydy, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y)

Model-specific implementation of fsigmay.

Parameters

- **dsigmaydy** – partial derivative of standard deviation of measurements w.r.t. model outputs
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t

virtual void **fdsigmazdp**(*realtype* *dsigmazdp, const *realtype* t, const *realtype* *p, const *realtype* *k, int ip)

Model-specific implementation of fsigmaz.

Parameters

- **dsigmazdp** – partial derivative of standard deviation of event measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index

virtual void **fdwdp**(*realtype* *dwdp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl, const *realtype* *stcl)

Model-specific sparse implementation of dwdp.

Parameters

- **dwdp** – Recurring terms in xdot, parameter derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws
- **stcl** – sensitivities of total abundances for conservation laws

virtual void **fdwdp**(*realtype* *dwdp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl, const *realtype* *stcl, int ip)

Model-specific sensitivity implementation of dwdp.

Parameters

- **dwdp** – Recurring terms in xdot, parameter derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws
- **stcl** – sensitivities of total abundances for conservation laws
- **ip** – sensitivity parameter index

virtual void **fdwdp_colptrs**(*SUNMatrixWrapper* &dwdp)

Model-specific implementation for dwdp, column pointers.

Parameters **dwdp** – sparse matrix to which colptrs will be written

virtual void **fdwdp_rowvals**(*SUNMatrixWrapper* &dwdp)

Model-specific implementation for dwdp, row values.

Parameters **dwdp** – sparse matrix to which rowvals will be written

virtual void **fdwdx**(*realtype* *dwdx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl)

Model-specific implementation of dwdx, data part.

Parameters

- **dwdx** – Recurring terms in xdot, state derivative
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – total abundances for conservation laws

virtual void **fdwdx_colptrs**(*SUNMatrixWrapper* &dwdx)

Model-specific implementation for dwdx, column pointers.

Parameters **dwdx** – sparse matrix to which colptrs will be written

virtual void **fdwdx_rowvals**(*SUNMatrixWrapper* &dwdx)

Model-specific implementation for dwdx, row values.

Parameters **dwdx** – sparse matrix to which rowvals will be written

virtual void **fdwdw**(*realtype* *dwdw, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *tcl)

Model-specific implementation of fdwdw, no w chainrule (Py)

Parameters

- **dwdw** – partial derivative w wrt w
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **tcl** – Total abundances for conservation laws

virtual void **fdwdw_colptrs**(*SUNMatrixWrapper* &dwdw)

Model-specific implementation of fdwdw, colptrs part.

Parameters **dwdw** – sparse matrix to which colptrs will be written

virtual void **fdwdw_rowvals**(*SUNMatrixWrapper* &dwdw)

Model-specific implementation of fdwdw, rowvals part.

Parameters **dwdw** – sparse matrix to which rowvals will be written

virtual void **fdydp**(*realtype* *dydp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *w, const *realtype* *dwdp)

Model-specific implementation of fdydp (MATLAB-only)

Parameters

- **dydp** – partial derivative of observables y w.r.t. model parameters p
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested
- **w** – repeating elements vector
- **dwdp** – Recurring terms in xdot, parameter derivative

virtual void **fdydp**(*realtype* *dydp, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *w, const *realtype* *tcl, const *realtype* *dtcldp)

Model-specific implementation of fdydp (Python)

Parameters

- **dydp** – partial derivative of observables y w.r.t. model parameters p
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested
- **w** – repeating elements vector
- **tcl** – total abundances for conservation laws
- **dtcldp** – Sensitivities of total abundances for conservation laws

virtual void **fdydx**(*realtype* *dydx, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *dwdx)

Model-specific implementation of fdydx.

Parameters

- **dydx** – partial derivative of observables y w.r.t. model states x

- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector
- **dwdx** – Recurring terms in xdot, state derivative

virtual void **fdzdp**(*realtype* *dzdp, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip)

Model-specific implementation of fdzdp.

Parameters

- **dzdp** – partial derivative of event-resolved output z w.r.t. model parameters p
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **ip** – parameter index w.r.t. which the derivative is requested

virtual void **fdzdx**(*realtype* *dzdx, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fdzdx.

Parameters

- **dzdx** – partial derivative of event-resolved output z w.r.t. model states x
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fJrz**(*realtype* *nllh, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz)

Model-specific implementation of fJrz.

Parameters

- **nllh** – regularization for event measurements z
- **iz** – event output index
- **p** – parameter vector

- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint

virtual void **fJy**(*realtype* *nllh, int iy, const *realtype* *p, const *realtype* *k, const *realtype* *y, const *realtype* *sigmay, const *realtype* *my)

Model-specific implementation of fJy.

Parameters

- **nllh** – negative log-likelihood for measurements y
- **iy** – output index
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint
- **sigmay** – measurement standard deviation at timepoint
- **my** – measurements at timepoint

virtual void **fJz**(*realtype* *nllh, int iz, const *realtype* *p, const *realtype* *k, const *realtype* *z, const *realtype* *sigmaz, const *realtype* *mz)

Model-specific implementation of fJz.

Parameters

- **nllh** – negative log-likelihood for event measurements z
- **iz** – event output index
- **p** – parameter vector
- **k** – constant vector
- **z** – model event output at timepoint
- **sigmaz** – event measurement standard deviation at timepoint
- **mz** – event measurements at timepoint

virtual void **frz**(*realtype* *rz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of frz.

Parameters

- **rz** – value of root function at current timepoint (non-output events not included)
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fsigmay**(*realtype* *sigmay, const *realtype* t, const *realtype* *p, const *realtype* *k, const *realtype* *y)

Model-specific implementation of fsigmay.

Parameters

- **sigmay** – standard deviation of measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector
- **y** – model output at timepoint t

virtual void **fsigmaz**(*realtype* *sigmaz, const *realtype* t, const *realtype* *p, const *realtype* *k)

Model-specific implementation of fsigmaz.

Parameters

- **sigmaz** – standard deviation of event measurements
- **t** – current time
- **p** – parameter vector
- **k** – constant vector

virtual void **fsrz**(*realtype* *srz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *sx, int ip)

Model-specific implementation of fsrz.

Parameters

- **srz** – Sensitivity of rz, total derivative
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **sx** – current state sensitivity
- **h** – Heaviside vector
- **ip** – sensitivity index

virtual void **fstau**(*realtype* *stau, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *tcl, const *realtype* *sx, int ip, int ie)

Model-specific implementation of fstau.

Parameters

- **stau** – total derivative of event timepoint
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector

- **h** – Heaviside vector
- **tcl** – total abundances for conservation laws
- **sx** – current state sensitivity
- **ip** – sensitivity index
- **ie** – event index

virtual void **fsx0**(*realtype* *sx0, const *realtype* t, const *realtype* *x0, const *realtype* *p, const *realtype* *k, int ip)

Model-specific implementation of fsx0.

Parameters

- **sx0** – initial state sensitivities
- **t** – initial time
- **x0** – initial state
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index

virtual void **fsx0_fixedParameters**(*realtype* *sx0, const *realtype* t, const *realtype* *x0, const *realtype* *p, const *realtype* *k, int ip, gsl::span<const int> reinitialization_state_idxs)

Model-specific implementation of fsx0_fixedParameters.

Parameters

- **sx0** – initial state sensitivities
- **t** – initial time
- **x0** – initial state
- **p** – parameter vector
- **k** – constant vector
- **ip** – sensitivity index
- **reinitialization_state_idxs** – Indices of states to be reinitialized based on provided constants / fixed parameters.

virtual void **fsz**(*realtype* *sz, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *sx, int ip)

Model-specific implementation of fsz.

Parameters

- **sz** – Sensitivity of rz, total derivative
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector

- **h** – Heaviside vector
- **sx** – current state sensitivity
- **ip** – sensitivity index

virtual void **fw**(*realtype* *w, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *tcl)

Model-specific implementation of fw.

Parameters

- **w** – Recurring terms in xdot
- **t** – timepoint
- **x** – vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **tcl** – total abundances for conservation laws

virtual void **fx0**(*realtype* *x0, const *realtype* t, const *realtype* *p, const *realtype* *k)

Model-specific implementation of fx0.

Parameters

- **x0** – initial state
- **t** – initial time
- **p** – parameter vector
- **k** – constant vector

virtual void **fx0_fixedParameters**(*realtype* *x0, const *realtype* t, const *realtype* *p, const *realtype* *k, gsl::span<const int> reinitialization_state_idxs)

Model-specific implementation of fx0_fixedParameters.

Parameters

- **x0** – initial state
- **t** – initial time
- **p** – parameter vector
- **k** – constant vector
- **reinitialization_state_idxs** – Indices of states to be reinitialized based on provided constants / fixed parameters.

virtual void **fy**(*realtype* *y, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w)

Model-specific implementation of fy.

Parameters

- **y** – model output at current timepoint
- **t** – current time
- **x** – current state

- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector
- **w** – repeating elements vector

virtual void **fz**(*realtype* *z, int ie, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h)

Model-specific implementation of fz.

Parameters

- **z** – value of event output
- **ie** – event index
- **t** – current time
- **x** – current state
- **p** – parameter vector
- **k** – constant vector
- **h** – Heaviside vector

virtual void **fdx_rdatadx_solver**(*realtype* *dx_rdatadx_solver, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k)

Compute dx_rdata / dx_solver.

Parameters

- **dx_rdatadx_solver** – dx_rdata / dx_solver
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws

virtual void **fdx_rdatadx_solver_colptrs**(*SUNMatrixWrapper* &dxrdatadxsolver)

Model-specific implementation of fdx_rdatadx_solver, colptrs part.

Parameters **dxrdatadxsolver** – sparse matrix to which colptrs will be written

virtual void **fdx_rdatadx_solver_rowvals**(*SUNMatrixWrapper* &dxrdatadxsolver)

Model-specific implementation of fdx_rdatadx_solver, rowvals part.

Parameters **dxrdatadxsolver** – sparse matrix to which rowvals will be written

virtual void **fdx_rdatadp**(*realtype* *dx_rdatadp, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k, const int ip)

Compute dx_rdata / dp.

Parameters

- **dx_rdatadp** – dx_rdata / dp
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied

- **tcl** – Total abundances for conservation laws
- **ip** – Sensitivity index

virtual void **fdx_rdatadtcl**(*realtype* *dx_rdatadtcl, const *realtype* *x, const *realtype* *tcl, const *realtype* *p, const *realtype* *k)

Compute dx_rdata / dtcl.

Parameters

- **dx_rdatadtcl** – dx_rdata / dtcl
- **p** – parameter vector
- **k** – constant vector
- **x** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws

virtual void **fdx_rdatadtcl_colptrs**(*SUNMatrixWrapper* &dx_rdatadtcl)

Model-specific implementation of fdx_rdatadtcl, colptrs part.

Parameters **dx_rdatadtcl** – sparse matrix to which colptrs will be written

virtual void **fdx_rdatadtcl_rowvals**(*SUNMatrixWrapper* &dx_rdatadtcl)

Model-specific implementation of fdx_rdatadtcl, rowvals part.

Parameters **dx_rdatadtcl** – sparse matrix to which rowvals will be written

virtual void **fdtotal_cldx_rdata**(*realtype* *dtotal_cldx_rdata, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k, const *realtype* *tcl)

Compute dtotal_cl / dx_rdata.

Parameters

- **dtotal_cldx_rdata** – dtotal_cl / dx_rdata
- **x_rdata** – State variables with conservation laws applied
- **p** – parameter vector
- **k** – constant vector
- **tcl** – Total abundances for conservation laws

virtual void **fdtotal_cldx_rdata_colptrs**(*SUNMatrixWrapper* &dtotal_cldx_rdata)

Model-specific implementation of fdtotal_cldx_rdata, colptrs part.

Parameters **dtotal_cldx_rdata** – sparse matrix to which colptrs will be written

virtual void **fdtotal_cldx_rdata_rowvals**(*SUNMatrixWrapper* &dtotal_cldx_rdata)

Model-specific implementation of fdtotal_cldx_rdata, rowvals part.

Parameters **dtotal_cldx_rdata** – sparse matrix to which rowvals will be written

virtual void **fdtotal_cldp**(*realtype* *dtotal_cldp, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k, const int ip)

Compute dtotal_cl / dp.

Parameters

- **dtotal_cldp** – dtotal_cl / dp
- **x_rdata** – State variables with conservation laws applied

- **p** – parameter vector
- **k** – constant vector
- **ip** – Sensitivity index

Public Members

bool **pythonGenerated**

Flag indicating Matlab- or Python-based model generation

SecondOrderMode **o2mode** = {*SecondOrderMode*::*none*}

Flag indicating whether for `amici::Solver::sensi_` == `amici::SensitivityOrder::second` directional or full second order derivative will be computed

std::vector<*realtype*> **idlist**

Flag array for DAE equations

Logger ***logger** = nullptr

Logger

Protected Functions

void **writeSliceEvent**(gsl::span<const *realtype*> slice, gsl::span<*realtype*> buffer, const int ie)

Write part of a slice to a buffer according to indices specified in z2event.

Parameters

- **slice** – Input data slice
- **buffer** – Output data slice
- **ie** – Event index

void **writeSensitivitySliceEvent**(gsl::span<const *realtype*> slice, gsl::span<*realtype*> buffer, const int ie)

Write part of a sensitivity slice to a buffer according to indices specified in z2event.

Parameters

- **slice** – source data slice
- **buffer** – output data slice
- **ie** – event index

void **writeLLHSensitivitySlice**(const std::vector<*realtype*> &dLLhdp, std::vector<*realtype*> &sllh, std::vector<*realtype*> &s2llh)

Separate first and second order objective sensitivity information and write them into the respective buffers.

Parameters

- **dLLhdp** – Data with mangled first- and second-order information
- **sllh** – First order buffer
- **s2llh** – Second order buffer

void **checkLLHBufferSize**(const std::vector<*realtype*> &sllh, const std::vector<*realtype*> &s2llh) const

Verify that the provided buffers have the expected size.

Parameters

- **sllh** – first order buffer
- **s2llh** – second order buffer

void **initializeVectors**()

Set the nplist-dependent vectors to their proper sizes.

void **fy**(*realtype* t, const *AmiVector* &x)

Compute observables / measurements.

Parameters

- **t** – Current timepoint
- **x** – Current state

void **fdydp**(*realtype* t, const *AmiVector* &x)

Compute partial derivative of observables y w.r.t. model parameters p.

Parameters

- **t** – Current timepoint
- **x** – Current state

void **fdydx**(*realtype* t, const *AmiVector* &x)

Compute partial derivative of observables y w.r.t. state variables x.

Parameters

- **t** – Current timepoint
- **x** – Current state

void **fsigmay**(int it, const *ExpData* *edata)

Compute standard deviation of measurements.

Parameters

- **it** – Timepoint index
- **edata** – Experimental data

void **fdsigmaydp**(int it, const *ExpData* *edata)

Compute partial derivative of standard deviation of measurements w.r.t. model parameters.

Parameters

- **it** – Timepoint index
- **edata** – pointer to *amici::ExpData* data instance holding sigma values

void **fdsigmaydy**(int it, const *ExpData* *edata)

Compute partial derivative of standard deviation of measurements w.r.t. model outputs.

Parameters

- **it** – Timepoint index
- **edata** – pointer to *amici::ExpData* data instance holding sigma values

void **fJy**(*realtype* &Jy, int it, const *AmiVector* &y, const *ExpData* &edata)

Compute negative log-likelihood of measurements y.

Parameters

- **Jy** – Variable to which llh will be added
- **it** – Timepoint index
- **y** – Simulated observable
- **edata** – Pointer to experimental data instance

void **fdJydy**(int it, const *AmiVector* &x, const *ExpData* &edata)

Compute partial derivative of time-resolved measurement negative log-likelihood Jy.

Parameters

- **it** – timepoint index
- **x** – state variables
- **edata** – Pointer to experimental data

void **fdJydsigma**(int it, const *AmiVector* &x, const *ExpData* &edata)

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. standard deviation sigma.

Parameters

- **it** – timepoint index
- **x** – state variables
- **edata** – pointer to experimental data instance

void **fdJydp**(const int it, const *AmiVector* &x, const *ExpData* &edata)

Compute sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. parameters for the given timepoint.

Parameters

- **it** – timepoint index
- **x** – state variables
- **edata** – pointer to experimental data instance

void **fdJydx**(const int it, const *AmiVector* &x, const *ExpData* &edata)

Sensitivity of time-resolved measurement negative log-likelihood Jy w.r.t. state variables.

Parameters

- **it** – Timepoint index
- **x** – State variables
- **edata** – Pointer to experimental data instance

void **fz**(int ie, *realtype* t, const *AmiVector* &x)

Compute event-resolved output.

Parameters

- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state

void **fdzdp**(int ie, *realtype* t, const *AmiVector* &x)

Compute partial derivative of event-resolved output `z` w.r.t. model parameters `p`

Parameters

- **ie** – event index
- **t** – current timepoint
- **x** – current state

void **fdzdx**(int ie, *realtype* t, const *AmiVector* &x)

Compute partial derivative of event-resolved output `z` w.r.t. model states `x`.

Parameters

- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state

void **frz**(int ie, *realtype* t, const *AmiVector* &x)

Compute event root function of events.

Equal to `Model::froot` but does not include non-output events.

Parameters

- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state

void **fdrzdp**(int ie, *realtype* t, const *AmiVector* &x)

Compute sensitivity of event-resolved root output w.r.t. model parameters `p`.

Parameters

- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state

void **fdrzdx**(int ie, *realtype* t, const *AmiVector* &x)

Compute sensitivity of event-resolved measurements rz w.r.t. model states `x`.

Parameters

- **ie** – Event index
- **t** – Current timepoint
- **x** – Current state

void **fsigmaz**(const int ie, const int nroots, const *realtype* t, const *ExpData* *edata)

Compute standard deviation of events.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint

- **edata** – Experimental data

void **fdsigmazdp**(int ie, int nroots, *realtype* t, const *ExpData* *edata)

Compute sensitivity of standard deviation of events measurements w.r.t. model parameters p.

Parameters

- **ie** – Event index
- **nroots** – Event occurrence
- **t** – Current timepoint
- **edata** – Pointer to experimental data instance

void **fJz**(*realtype* &Jz, int nroots, const *AmiVector* &z, const *ExpData* &edata)

Compute negative log-likelihood of event-resolved measurements z.

Parameters

- **Jz** – Variable to which llh will be added
- **nroots** – Event index
- **z** – Simulated event
- **edata** – Experimental data

void **fdJzdz**(const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute partial derivative of event measurement negative log-likelihood Jz.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint
- **x** – State variables
- **edata** – Experimental data

void **fdJzdsigma**(const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint
- **x** – State variables
- **edata** – Pointer to experimental data instance

void **fdJzdp**(const int ie, const int nroots, *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. parameters.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint

- **x** – State variables
- **edata** – Pointer to experimental data instance

void **fdJzdx**(const int ie, const int nroots, *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute sensitivity of event-resolved measurement negative log-likelihood Jz w.r.t. state variables.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint
- **x** – State variables
- **edata** – Experimental data

void **fJrz**(*realtype* &Jrz, int nroots, const *AmiVector* &rz, const *ExpData* &edata)

Compute regularization of negative log-likelihood with roots of event-resolved measurements rz.

Parameters

- **Jrz** – Variable to which regularization will be added
- **nroots** – Event index
- **rz** – Regularization variable
- **edata** – Experimental data

void **fdJrzdz**(const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute partial derivative of event measurement negative log-likelihood J.

Parameters

- **ie** – Event index
- **nroots** – Event index
- **t** – Current timepoint
- **x** – State variables
- **edata** – Experimental data

void **fdJrzdsigma**(const int ie, const int nroots, const *realtype* t, const *AmiVector* &x, const *ExpData* &edata)

Compute sensitivity of event measurement negative log-likelihood Jz w.r.t. standard deviation sigmaz.

Parameters

- **ie** – event index
- **nroots** – event index
- **t** – current timepoint
- **x** – state variables
- **edata** – pointer to experimental data instance

void **fw**(*realtype* t, const *realtype* *x)

Compute recurring terms in xdot.

Parameters

- **t** – Timepoint

- **x** – Array with the states

void **fdwdp**(*realtype* t, const *realtype* *x)

Compute parameter derivative for recurring terms in xdot.

Parameters

- **t** – Timepoint
- **x** – Array with the states

void **fdwdx**(*realtype* t, const *realtype* *x)

Compute state derivative for recurring terms in xdot.

Parameters

- **t** – Timepoint
- **x** – Array with the states

void **fdwdw**(*realtype* t, const *realtype* *x)

Compute self derivative for recurring terms in xdot.

Parameters

- **t** – Timepoint
- **x** – Array with the states

virtual void **fx_rdata**(*realtype* *x_rdata, const *realtype* *x_solver, const *realtype* *tcl, const *realtype* *p, const *realtype* *k)

Compute fx_rdata.

To be implemented by derived class if applicable.

Parameters

- **x_rdata** – State variables with conservation laws expanded
- **x_solver** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws
- **p** – parameter vector
- **k** – constant vector

virtual void **fsx_rdata**(*realtype* *sx_rdata, const *realtype* *sx_solver, const *realtype* *stcl, const *realtype* *p, const *realtype* *k, const *realtype* *x_solver, const *realtype* *tcl, const int ip)

Compute fsx_solver.

To be implemented by derived class if applicable.

Parameters

- **sx_rdata** – State sensitivity variables with conservation laws expanded
- **sx_solver** – State sensitivity variables with conservation laws applied
- **stcl** – Sensitivities of total abundances for conservation laws
- **p** – parameter vector
- **k** – constant vector
- **x_solver** – State variables with conservation laws applied
- **tcl** – Total abundances for conservation laws

- **ip** – Sensitivity index

virtual void **fx_solver**(*realtype* *x_solver, const *realtype* *x_rdata)

Compute fx_solver.

To be implemented by derived class if applicable.

Parameters

- **x_solver** – State variables with conservation laws applied
- **x_rdata** – State variables with conservation laws expanded

virtual void **fsx_solver**(*realtype* *sx_solver, const *realtype* *sx_rdata)

Compute fsx_solver.

To be implemented by derived class if applicable.

Parameters

- **sx_rdata** – State sensitivity variables with conservation laws expanded
- **sx_solver** – State sensitivity variables with conservation laws applied

virtual void **ftotal_cl**(*realtype* *total_cl, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k)

Compute ftotal_cl.

To be implemented by derived class if applicable.

Parameters

- **total_cl** – Total abundances of conservation laws
- **x_rdata** – State variables with conservation laws expanded
- **p** – parameter vector
- **k** – constant vector

virtual void **fstotal_cl**(*realtype* *stotal_cl, const *realtype* *sx_rdata, const int ip, const *realtype* *x_rdata, const *realtype* *p, const *realtype* *k, const *realtype* *tcl)

Compute fstotal_cl.

To be implemented by derived class if applicable.

Parameters

- **stotal_cl** – Sensitivities for the total abundances of conservation laws
- **sx_rdata** – State sensitivity variables with conservation laws expanded
- **ip** – Sensitivity index
- **x_rdata** – State variables with conservation laws expanded
- **p** – parameter vector
- **k** – constant vector
- **tcl** – Total abundances for conservation laws

const_N_Vector **computeX_pos**(*const_N_Vector* x)

Compute non-negative state vector.

Compute non-negative state vector according to stateIsNonNegative. If anyStateNonNegative is set to `false`, i.e., all entries in stateIsNonNegative are `false`, this function directly returns `x`, otherwise all entries of x are copied in to `amici::Model::x_pos_tmp_` and negative values are replaced by `0` where applicable.

Parameters **x** – State vector possibly containing negative values

Returns State vector with negative values replaced by `0` according to stateIsNonNegative

const *realtype* ***computeX_pos**(*AmiVector* const &x)

Compute non-negative state vector.

Compute non-negative state vector according to stateIsNonNegative. If anyStateNonNegative is set to `false`, i.e., all entries in stateIsNonNegative are `false`, this function directly returns `x`, otherwise all entries of x are copied in to `amici::Model::x_pos_tmp_` and negative values are replaced by `0` where applicable.

Parameters **x** – State vector possibly containing negative values

Returns State vector with negative values replaced by `0` according to stateIsNonNegative

Protected Attributes

ModelState **state_**

All variables necessary for function evaluation

ModelStateDerived **derived_state_**

Storage for model quantities beyond *ModelState* for the current timepoint

std::vector<int> **z2event_**

index indicating to which event an event output belongs

std::vector<*realtype*> **x0data_**

state initialization (size nx_solver)

std::vector<*realtype*> **sx0data_**

sensitivity initialization (size nx_rdata x nplist, row-major)

std::vector<bool> **state_is_non_negative_**

vector of bools indicating whether state variables are to be assumed to be positive

std::vector<bool> **root_initial_values_**

Vector of booleans indicating the initial boolean value for every event trigger function. Events at t0 can only trigger if the initial value is set to `false`. Must be specified during model compilation by setting the `initialValue` attribute of an event trigger.

bool **any_state_non_negative_** = {false}

boolean indicating whether any entry in stateIsNonNegative is `true`

int **nmaxevent_** = {10}

maximal number of events to track

SteadyStateSensitivityMode **steadystate_sensitivity_mode_** = {*SteadyStateSensitivityMode*::*newtonOnly*}

flag indicating whether steadystate sensitivities are to be computed via FSA when steadyStateSimulation is used

bool **always_check_finite_** = {false}

Indicates whether the result of every call to `Model::f*` should be checked for finiteness

bool **sigma_res_** = {false}

indicates whether sigma residuals are to be added for every datapoint

realtype **min_sigma_** = {50.0}

offset to ensure positivity of sigma residuals, only has an effect when `sigma_res_` is `true`

Friends

template<class **Archive**>
friend void **serialize**(*Archive* &ar, *Model* &m, unsigned int version)

Serialize *Model* (see `boost::serialization::serialize`).

Parameters

- **ar** – Archive to serialize to
- **m** – Data to serialize
- **version** – Version number

friend bool **operator==**(const *Model* &a, const *Model* &b)

Check equality of data members.

Parameters

- **a** – First model instance
- **b** – Second model instance

Returns Equality

Class Model_DAE

- Defined in file_include_amici_model_dae.h

Inheritance Relationships

Base Type

- `public amici::Model` (*Class Model*)

Class Documentation

class **Model_DAE** : public amici::*Model*

The *Model* class represents an AMICI DAE model.

The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.

Public Functions

Model_DAE() = default

default constructor

inline **Model_DAE**(const *ModelDimensions* &model_dimensions, *SimulationParameters* simulation_parameters, const *SecondOrderMode* o2mode, std::vector<*realtype*> const &idlist, std::vector<int> const &z2event, const bool pythonGenerated = false, const int ndxdotdp_explicit = 0)

Constructor with model dimensions.

Parameters

- **model_dimensions** – *Model* dimensions
- **simulation_parameters** – Simulation parameters
- **o2mode** – second order sensitivity mode
- **idlist** – indexes indicating algebraic components (DAE only)
- **z2event** – mapping of event outputs to events
- **pythonGenerated** – flag indicating matlab or python wrapping
- **ndxdotdp_explicit** – number of nonzero elements dxdotdp_explicit

virtual void **fJ**(*realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) override

Dense Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – dense matrix to which values of the jacobian will be written

void **fJ**(*realtype* t, *realtype* cj, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xdot, SUNMatrix J)

Jacobian of xdot with respect to states x.

Parameters

- **t** – timepoint
- **cj** – scaling factor, inverse of the step size

- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xdot** – Vector with the right hand side
- **J** – Matrix to which the Jacobian will be written

virtual void **fJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) override

Dense Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

void **fJB**(*realtype* t, *realtype* cj, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xB, *const_N_Vector* dxB, SUNMatrix JB)

Jacobian of xBdot with respect to adjoint state xB.

Parameters

- **t** – timepoint
- **cj** – scaling factor, inverse of the step size
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **JB** – Matrix to which the Jacobian will be written

virtual void **fJSparse**(*realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) override

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – sparse matrix to which values of the Jacobian will be written

void **fJSparse**(*realtype* t, *realtype* cj, *const_N_Vector* x, *const_N_Vector* dx, SUNMatrix J)

J in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

- **t** – timepoint
- **cj** – scalar in Jacobian (inverse stepsize)
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **J** – Matrix to which the Jacobian will be written

virtual void **fJSparseB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) override

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

void **fJSparseB**(*realtype* t, *realtype* cj, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xB, *const_N_Vector* dxB, SUNMatrix JB)

JB in sparse form (for sparse solvers from the SuiteSparse Package)

Parameters

- **t** – timepoint
- **cj** – scalar in Jacobian
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **JB** – Matrix to which the Jacobian will be written

virtual void **fJDiag**(*realtype* t, *AmiVector* &JDiag, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx) override

Diagonal of the Jacobian (for preconditioning)

Parameters

- **t** – timepoint
- **JDiag** – Vector to which the Jacobian diagonal will be written

- **cj** – scaling factor, inverse of the step size
- **x** – Vector with the states
- **dx** – Vector with the derivative states

virtual void **fJv**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, const *AmiVector* &v, *AmiVector* &nJv, *realtype* cj) override

Jacobian multiply function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **v** – multiplication vector (unused)
- **nJv** – array to which result of multiplication will be written
- **cj** – scaling factor (inverse of timestep, DAE only)

void **fJv**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* v, N_Vector Jv, *realtype* cj)

Matrix vector product of J with a vector v (for iterative solvers)

Parameters

- **t** – timepoint
- **cj** – scaling factor, inverse of the step size
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **v** – Vector with which the Jacobian is multiplied
- **Jv** – Vector to which the Jacobian vector product will be written

void **fJvB**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xB, *const_N_Vector* dxB, *const_N_Vector* vB, N_Vector JvB, *realtype* cj)

Matrix vector product of JB with a vector v (for iterative solvers)

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **vB** – Vector with which the Jacobian is multiplied
- **JvB** – Vector to which the Jacobian vector product will be written
- **cj** – scalar in Jacobian (inverse stepsize)

virtual void **froot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, gsl::span<*realtype*> root) override

Root function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **root** – array to which values of the root function will be written

void **froot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, gsl::span<*realtype*> root)

Event trigger function for events.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **root** – array with root function values

virtual void **fxdot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, *AmiVector* &xdot) override

Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – array to which values of the residual function will be written

void **fxdot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, N_Vector xdot)

Residual function of the DAE.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xdot** – Vector with the right hand side

void **fxBdot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xB, *const_N_Vector* dxB, N_Vector xBdot)

Right hand side of differential equation for adjoint state xB.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states

- **xBdot** – Vector with the adjoint right hand side

void **fqBdot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, *const_N_Vector* xB, *const_N_Vector* dxB, N_Vector qBdot)

Right hand side of integral equation for quadrature states qB.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **qBdot** – Vector with the adjoint quadrature right hand side

virtual void **fxBdot_ss**(const *realtype* t, const *AmiVector* &xB, const *AmiVector* &dxB, *AmiVector* &xBdot) override

Residual function backward when running in steady state mode.

Parameters

- **t** – time
- **xB** – adjoint state
- **dxB** – time derivative of state (DAE only)
- **xBdot** – array to which values of the residual function will be written

void **fxBdot_ss**(*realtype* t, *const_N_Vector* xB, *const_N_Vector* dxB, N_Vector xBdot) const

Implementation of fxBdot for steady state case at the N_Vector level.

Parameters

- **t** – timepoint
- **xB** – Vector with the adjoint state
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side

void **fqBdot_ss**(*realtype* t, *const_N_Vector* xB, *const_N_Vector* dxB, N_Vector qBdot) const

Implementation of fqBdot for steady state at the N_Vector level.

Parameters

- **t** – timepoint
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **qBdot** – Vector with the adjoint quadrature right hand side

virtual void **fJSparseB_ss**(SUNMatrix JB) override

Sparse Jacobian function backward, steady state case.

Parameters **JB** – sparse matrix to which values of the Jacobian will be written virtual void **writeSteadystateJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot) override

Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.

Parameters

- **t** – timepoint
- **cj** – scalar in Jacobian
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint state right hand side

void **fdxdotdp**(*realtype* t, const *const_N_Vector* x, const *const_N_Vector* dx)

Sensitivity of dx/dt wrt model parameters p.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states

inline virtual void **fdxdotdp**(const *realtype* t, const *AmiVector* &x, const *AmiVector* &dx) override

Model-specific sparse implementation of explicit parameter derivative of right hand side.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)

virtual void **fsxdot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, int ip, const *AmiVector* &sx, const *AmiVector* &sdx, *AmiVector* &sxdot) override

Sensitivity Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **ip** – parameter index
- **sx** – sensitivity state
- **sdx** – time derivative of sensitivity state (DAE only)
- **sxdot** – array to which values of the sensitivity residual function will be written

void **fsxdot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* dx, int ip, *const_N_Vector* sx, *const_N_Vector* sdx, N_Vector sxdot)

Right hand side of differential equation for state sensitivities sx.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **ip** – parameter index
- **sx** – Vector with the state sensitivities
- **sdx** – Vector with the derivative state sensitivities
- **sxdot** – Vector with the sensitivity right hand side

void **fM**(*realtype* t, *const_N_Vector* x)

Mass matrix for DAE systems.

Parameters

- **t** – timepoint
- **x** – Vector with the states

virtual std::unique_ptr<*Solver*> **getSolver**() override

Retrieves the solver object.

Returns The *Solver* instance

Protected Functions

virtual void **fJSparse**(SUNMatrixContent_Sparse JSparse, *realtype* t, const *realtype* *x, const double *p, const double *k, const *realtype* *h, *realtype* cj, const *realtype* *dx, const *realtype* *w, const *realtype* *dwdx) = 0

Model specific implementation for fJSparse.

Parameters

- **JSparse** – Matrix to which the Jacobian will be written
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **cj** – scaling factor, inverse of the step size
- **dx** – Vector with the derivative states
- **w** – vector with helper variables
- **dwdx** – derivative of w wrt x

virtual void **froot**(*realtype* *root, *realtype* t, const *realtype* *x, const double *p, const double *k, const *realtype* *h, const *realtype* *dx)

Model specific implementation for froot.

Parameters

- **root** – values of the trigger function

- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **dx** – Vector with the derivative states

virtual void **fxdot**(*realtype* *xdot, *realtype* t, const *realtype* *x, const double *p, const double *k, const *realtype* *h, const *realtype* *dx, const *realtype* *w) = 0

Model specific implementation for fxdot.

Parameters

- **xdot** – residual function
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **dx** – Vector with the derivative states

virtual void **fdxdotdp**(*realtype* *dxdotdp, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *dx, const *realtype* *w, const *realtype* *dwdp)

Model specific implementation of fdxdotdp.

Parameters

- **dxdotdp** – partial derivative xdot wrt p
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **ip** – parameter index
- **dx** – Vector with the derivative states
- **w** – vector with helper variables
- **dwdp** – derivative of w wrt p

virtual void **fM**(*realtype* *M, const *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k)

Model specific implementation of fM.

Parameters

- **M** – mass matrix
- **t** – timepoint

- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector

Class Model_ODE

- Defined in file_include_amici_model_ode.h

Inheritance Relationships

Base Type

- `public amici::Model` (*Class Model*)

Class Documentation

class **Model_ODE** : public amici::*Model*

The *Model* class represents an AMICI ODE model.

The model does not contain any data, but represents the state of the model at a specific time t. The states must not always be in sync, but may be updated asynchronously.

Public Functions

Model_ODE() = default

default constructor

inline **Model_ODE**(*ModelDimensions* const &model_dimensions, *SimulationParameters* simulation_parameters, const *SecondOrderMode* o2mode, std::vector<*realtype*> const &idlist, std::vector<int> const &z2event, const bool pythonGenerated = false, const int ndxdotdp_explicit = 0, const int ndxdotdx_explicit = 0, const int w_recursion_depth = 0)

Constructor with model dimensions.

Parameters

- **model_dimensions** – *Model* dimensions
- **simulation_parameters** – Simulation parameters
- **o2mode** – second order sensitivity mode
- **idlist** – indexes indicating algebraic components (DAE only)
- **z2event** – mapping of event outputs to events
- **pythonGenerated** – flag indicating matlab or python wrapping
- **ndxdotdp_explicit** – number of nonzero elements dxdotdp_explicit
- **ndxdotdx_explicit** – number of nonzero elements dxdotdx_explicit
- **w_recursion_depth** – Recursion depth of fw

virtual void **fJ**(*realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) override

Dense Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – dense matrix to which values of the jacobian will be written

void **fJ**(*realtype* t, *const_N_Vector* x, *const_N_Vector* xdot, SUNMatrix J)

Implementation of fJ at the N_Vector level.

This function provides an interface to the model specific routines for the solver implementation as well as the *AmiVector* level implementation

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xdot** – Vector with the right hand side
- **J** – Matrix to which the Jacobian will be written

virtual void **fJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) override

Dense Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

void **fJB**(*realtype* t, *const_N_Vector* x, *const_N_Vector* xB, *const_N_Vector* xBdot, SUNMatrix JB)

Implementation of fJB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xB** – Vector with the adjoint states

- **xBdot** – Vector with the adjoint right hand side
- **JB** – Matrix to which the Jacobian will be written

virtual void **fJSparse**(*realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, SUNMatrix J) override

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **J** – sparse matrix to which values of the Jacobian will be written

void **fJSparse**(*realtype* t, *const_N_Vector* x, SUNMatrix J)

Implementation of fJSparse at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation as well as the *AmiVector* level implementation.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **J** – Matrix to which the Jacobian will be written

virtual void **fJSparseB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot, SUNMatrix JB) override

Sparse Jacobian function.

Parameters

- **t** – time
- **cj** – scaling factor (inverse of timestep, DAE only)
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint right hand side (unused)
- **JB** – dense matrix to which values of the jacobian will be written

void **fJSparseB**(*realtype* t, *const_N_Vector* x, *const_N_Vector* xB, *const_N_Vector* xBdot, SUNMatrix JB)

Implementation of fJSparseB at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation.

Parameters

- **t** – timepoint
- **x** – Vector with the states

- **xB** – Vector with the adjoint states
- **xBdot** – Vector with the adjoint right hand side
- **JB** – Matrix to which the Jacobian will be written

void **fJDiag**(*realtype* t, N_Vector JDiag, *const_N_Vector* x)

Implementation of fJDiag at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation.

Parameters

- **t** – timepoint
- **JDiag** – Vector to which the Jacobian diagonal will be written
- **x** – Vector with the states

virtual void **fJDiag**(*realtype* t, *AmiVector* &JDiag, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx) override

Diagonal of the Jacobian (for preconditioning)

Parameters

- **t** – timepoint
- **JDiag** – Vector to which the Jacobian diagonal will be written
- **cj** – scaling factor, inverse of the step size
- **x** – Vector with the states
- **dx** – Vector with the derivative states

virtual void **fJv**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xdot, const *AmiVector* &v, *AmiVector* &nJv, *realtype* cj) override

Jacobian multiply function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – values of residual function (unused)
- **v** – multiplication vector (unused)
- **nJv** – array to which result of multiplication will be written
- **cj** – scaling factor (inverse of timestep, DAE only)

void **fJv**(*const_N_Vector* v, N_Vector Jv, *realtype* t, *const_N_Vector* x)

Implementation of fJv at the N_Vector level.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **v** – Vector with which the Jacobian is multiplied
- **Jv** – Vector to which the Jacobian vector product will be written

void **fJvB**(*const_N_Vector* vB, N_Vector JvB, *realtype* t, *const_N_Vector* x, *const_N_Vector* xB)

Implementation of fJvB at the N_Vector level.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xB** – Vector with the adjoint states
- **vB** – Vector with which the Jacobian is multiplied
- **JvB** – Vector to which the Jacobian vector product will be written

virtual void **froot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, gsl::span<*realtype*> root) override

Root function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **root** – array to which values of the root function will be written

void **froot**(*realtype* t, *const_N_Vector* x, gsl::span<*realtype*> root)

Implementation of froot at the N_Vector level This function provides an interface to the model specific routines for the solver implementation as well as the *AmiVector* level implementation.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **root** – array with root function values

virtual void **fxdot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, *AmiVector* &xdot) override

Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **xdot** – array to which values of the residual function will be written

void **fxdot**(*realtype* t, *const_N_Vector* x, N_Vector xdot)

Implementation of fxdot at the N_Vector level, this function provides an interface to the model specific routines for the solver implementation as well as the *AmiVector* level implementation.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xdot** – Vector with the right hand side

void **fxBdot**(*realtype* t, N_Vector x, N_Vector xB, N_Vector xBdot)

Implementation of fxBdot at the N_Vector level.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xB** – Vector with the adjoint states
- **xBdot** – Vector with the adjoint right hand side

void **fqBdot**(*realtype* t, *const_N_Vector* x, *const_N_Vector* xB, N_Vector qBdot)

Implementation of fqBdot at the N_Vector level.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **xB** – Vector with the adjoint states
- **qBdot** – Vector with the adjoint quadrature right hand side

virtual void **fxBdot_ss**(const *realtype* t, const *AmiVector* &xB, const *AmiVector*&, *AmiVector* &xBdot) override

Residual function backward when running in steady state mode.

Parameters

- **t** – time
- **xB** – adjoint state
- **dxB** – time derivative of state (DAE only)
- **xBdot** – array to which values of the residual function will be written

void **fxBdot_ss**(*realtype* t, *const_N_Vector* xB, N_Vector xBdot) const

Implementation of fxBdot for steady state at the N_Vector level.

Parameters

- **t** – timepoint
- **xB** – Vector with the states
- **xBdot** – Vector with the adjoint right hand side

void **fqBdot_ss**(*realtype* t, N_Vector xB, N_Vector qBdot) const

Implementation of fqBdot for steady state case at the N_Vector level.

Parameters

- **t** – timepoint
- **xB** – Vector with the adjoint states
- **qBdot** – Vector with the adjoint quadrature right hand side

virtual void **fJSparseB_ss**(SUNMatrix JB) override

Sparse Jacobian function backward, steady state case.

Parameters **JB** – sparse matrix to which values of the Jacobian will be written virtual void **writeSteadystateJB**(const *realtype* t, *realtype* cj, const *AmiVector* &x, const *AmiVector* &dx, const *AmiVector* &xB, const *AmiVector* &dxB, const *AmiVector* &xBdot) override

Computes the sparse backward Jacobian for steadystate integration and writes it to the model member.

Parameters

- **t** – timepoint
- **cj** – scalar in Jacobian
- **x** – Vector with the states
- **dx** – Vector with the derivative states
- **xB** – Vector with the adjoint states
- **dxB** – Vector with the adjoint derivative states
- **xBdot** – Vector with the adjoint state right hand side

virtual void **fsxdot**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx, int ip, const *AmiVector* &sx, const *AmiVector* &sdx, *AmiVector* &sxdot) override

Sensitivity Residual function.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)
- **ip** – parameter index
- **sx** – sensitivity state
- **sdx** – time derivative of sensitivity state (DAE only)
- **sxdot** – array to which values of the sensitivity residual function will be written

void **fsxdot**(*realtype* t, *const_N_Vector* x, int ip, *const_N_Vector* sx, N_Vector sxdot)

Implementation of fsxdot at the N_Vector level.

Parameters

- **t** – timepoint
- **x** – Vector with the states
- **ip** – parameter index
- **sx** – Vector with the state sensitivities
- **sxdot** – Vector with the sensitivity right hand side

virtual std::unique_ptr<*Solver*> **getSolver**() override

Retrieves the solver object.

Returns The *Solver* instance

Protected Functions

virtual void **fJSparse**(SUNMatrixContent_Sparse JSparse, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *dwdx)

Model specific implementation for fJSparse (Matlab)

Parameters

- **JSparse** – Matrix to which the Jacobian will be written
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **dwdx** – derivative of w wrt x

virtual void **fJSparse**(*realtype* *JSparse, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w, const *realtype* *dwdx)

Model specific implementation for fJSparse, data only (Py)

Parameters

- **JSparse** – Matrix to which the Jacobian will be written
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables
- **dwdx** – derivative of w wrt x

virtual void **fJSparse_colptrs**(*SUNMatrixWrapper* &JSparse)

Model specific implementation for fJSparse, column pointers.

Parameters **JSparse** – sparse matrix to which colptrs will be written

virtual void **fJSparse_rowvals**(*SUNMatrixWrapper* &JSparse)

Model specific implementation for fJSparse, row values.

Parameters **JSparse** – sparse matrix to which rowvals will be written

virtual void **froot**(*realtype* *root, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *tcl)

Model specific implementation for froot.

Parameters

- **root** – values of the trigger function
- **t** – timepoint

- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **tcl** – total abundances for conservation laws

virtual void **fxdot**(*realtype* *xdot, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w) = 0

Model specific implementation for fxdot.

Parameters

- **xdot** – residual function
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables

virtual void **fdxdotdp**(*realtype* *dxdotdp, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, int ip, const *realtype* *w, const *realtype* *dwdp)

Model specific implementation of fdxdotdp, with w chainrule (Matlab)

Parameters

- **dxdotdp** – partial derivative xdot wrt p
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **ip** – parameter index
- **w** – vector with helper variables
- **dwdp** – derivative of w wrt p

virtual void **fdxdotdp_explicit**(*realtype* *dxdotdp_explicit, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w)

Model specific implementation of fdxdotdp_explicit, no w chainrule (Py)

Parameters

- **dxdotdp_explicit** – partial derivative xdot wrt p
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector

- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables

virtual void **fdxdotdp_explicit_colptrs**(*SUNMatrixWrapper* &dxdotdp)

Model specific implementation of fdxdotdp_explicit, colptrs part.

Parameters **dxdotdp** – sparse matrix to which colptrs will be written

virtual void **fdxdotdp_explicit_rowvals**(*SUNMatrixWrapper* &dxdotdp)

Model specific implementation of fdxdotdp_explicit, rowvals part.

Parameters **dxdotdp** – sparse matrix to which rowvals will be written

virtual void **fdxdotdx_explicit**(*realtype* *dxdotdx_explicit, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w)

Model specific implementation of fdxdotdx_explicit, no w chainrule (Py)

Parameters

- **dxdotdx_explicit** – partial derivative xdot wrt x
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – heavyside vector
- **w** – vector with helper variables

virtual void **fdxdotdx_explicit_colptrs**(*SUNMatrixWrapper* &dxdotdx)

Model specific implementation of fdxdotdx_explicit, colptrs part.

Parameters **dxdotdx** – sparse matrix to which colptrs will be written

virtual void **fdxdotdx_explicit_rowvals**(*SUNMatrixWrapper* &dxdotdx)

Model specific implementation of fdxdotdx_explicit, rowvals part.

Parameters **dxdotdx** – sparse matrix to which rowvals will be written

virtual void **fdxdotdw**(*realtype* *dxdotdw, *realtype* t, const *realtype* *x, const *realtype* *p, const *realtype* *k, const *realtype* *h, const *realtype* *w)

Model specific implementation of fdxdotdw, data part.

Parameters

- **dxdotdw** – partial derivative xdot wrt w
- **t** – timepoint
- **x** – Vector with the states
- **p** – parameter vector
- **k** – constants vector
- **h** – Heaviside vector
- **w** – vector with helper variables

virtual void **fdxdotdw_colptrs**(*SUNMatrixWrapper* &dxdotdw)

Model specific implementation of fdxdotdw, colptrs part.

Parameters **dxdotdw** – sparse matrix to which colptrs will be written

virtual void **fdxdotdw_rowvals**(*SUNMatrixWrapper* &dxdotdw)

Model specific implementation of fdxdotdw, rowvals part.

Parameters **dxdotdw** – sparse matrix to which rowvals will be written

void **fdxdotdw**(*realtype* t, *const_N_Vector* x)

Sensitivity of dx/dt wrt model parameters w.

Parameters

- **t** – timepoint
- **x** – Vector with the states

void **fdxdotdp**(*realtype* t, *const_N_Vector* x)

Explicit sensitivity of dx/dt wrt model parameters p

Parameters

- **t** – timepoint
- **x** – Vector with the states

virtual void **fdxdotdp**(*realtype* t, const *AmiVector* &x, const *AmiVector* &dx) override

Model-specific sparse implementation of explicit parameter derivative of right hand side.

Parameters

- **t** – time
- **x** – state
- **dx** – time derivative of state (DAE only)

Class ModelContext

- Defined in file_include_amici_rdata.h

Inheritance Relationships

Base Type

- `public amici::ContextManager` (*Class ContextManager*)

Class Documentation

class **ModelContext** : public amici::*ContextManager*

The *ModelContext* temporarily stores amici::Model::state and restores it when going out of scope.

Public Functions

explicit **ModelContext**(*Model* *model)

initialize backup of the original values.

Parameters **model** –

ModelContext &**operator=**(const *ModelContext* &other) = delete

~ModelContext()

void **restore**()

Restore original state on constructor-supplied *amici::Model*. Will be called during destruction. Explicit call is generally not necessary.

Class NewtonFailure

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **NewtonFailure** : public amici::*AmiException*

Newton failure exception.

This exception should be thrown when the steady state computation failed to converge for this exception we can assume that we can recover from the exception and return a solution struct to the user

Public Functions

NewtonFailure(int code, char const *function)

Constructor, simply calls *AmiException* constructor.

Parameters

- **function** – name of the function in which the error occurred
- **code** – error code

Public Members

int **error_code**

error code returned by solver

Class NewtonSolver

- Defined in file_include_amici_newton_solver.h

Inheritance Relationships

Derived Types

- `public amici::NewtonSolverDense` (*Class NewtonSolverDense*)
- `public amici::NewtonSolverSparse` (*Class NewtonSolverSparse*)

Class Documentation

class **NewtonSolver**

The *NewtonSolver* class sets up the linear solver for the Newton method.

Subclassed by *amici::NewtonSolverDense*, *amici::NewtonSolverSparse*

Public Functions

explicit **NewtonSolver**(const *Model* &model)

Initializes solver according to the dimensions in the provided model.

Parameters **model** – pointer to the model object

void **getStep**(*AmiVector* &delta, *Model* &model, const *SimulationState* &state)

Computes the solution of one Newton iteration.

Parameters

- **delta** – containing the RHS of the linear system, will be overwritten by solution to the linear system
- **model** – pointer to the model instance
- **state** – current simulation state

void **computeNewtonSensis**(*AmiVectorArray* &sx, *Model* &model, const *SimulationState* &state)

Computes steady state sensitivities.

Parameters

- **sx** – pointer to state variable sensitivities
- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **prepareLinearSystem**(*Model* &model, const *SimulationState* &state) = 0

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **prepareLinearSystemB**(*Model* &model, const *SimulationState* &state) = 0

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **solveLinearSystem**(*AmiVector* &rhs) = 0

Solves the linear system for the Newton step.

Parameters **rhs** – containing the RHS of the linear system, will be overwritten by solution to the linear system

virtual void **reinitialize**() = 0

Reinitialize the linear solver.

virtual bool **is_singular**(*Model* &model, const *SimulationState* &state) const = 0

Checks whether linear system is singular.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

Returns boolean indicating whether the linear system is singular (condition number < 1/machine precision)

virtual **~NewtonSolver**() = default

Public Static Functions

static std::unique_ptr<*NewtonSolver*> **getSolver**(const *Solver* &simulationSolver, const *Model* &model)

Factory method to create a *NewtonSolver* based on linsolType.

Parameters

- **simulationSolver** – solver with settings
- **model** – pointer to the model instance

Returns solver *NewtonSolver* according to the specified linsolType

Protected Attributes

AmiVector **xdot_**

dummy rhs, used as dummy argument when computing J and JB

AmiVector **x_**

dummy state, attached to linear solver

AmiVector **xB_**

dummy adjoint state, used as dummy argument when computing JB

AmiVector **dxB_**

dummy differential adjoint state, used as dummy argument when computing JB

Class NewtonSolverDense

- Defined in file_include_amici_newton_solver.h

Inheritance Relationships

Base Type

- `public amici::NewtonSolver` (*Class NewtonSolver*)

Class Documentation

class **NewtonSolverDense** : public amici::*NewtonSolver*

The *NewtonSolverDense* provides access to the dense linear solver for the Newton method.

Public Functions

explicit **NewtonSolverDense**(const *Model* &model)

constructor for sparse solver

Parameters model – model instance that provides problem dimensions

NewtonSolverDense(const *NewtonSolverDense*&) = delete

NewtonSolverDense &**operator=**(const *NewtonSolverDense* &other) = delete

~NewtonSolverDense() override

virtual void **solveLinearSystem**(*AmiVector* &rhs) override

Solves the linear system for the Newton step.

Parameters rhs – containing the RHS of the linear system, will be overwritten by solution to the linear system virtual void **prepareLinearSystem**(*Model* &model, const *SimulationState* &state) override

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **prepareLinearSystemB**(*Model* &model, const *SimulationState* &state) override

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **reinitialize**() override

Reinitialize the linear solver.

virtual bool **is_singular**(*Model* &model, const *SimulationState* &state) const override

Checks whether linear system is singular.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

Returns boolean indicating whether the linear system is singular (condition number < 1/machine precision)

Class NewtonSolverSparse

- Defined in file_include_amici_newton_solver.h

Inheritance Relationships

Base Type

- `public amici::NewtonSolver` (*Class NewtonSolver*)

Class Documentation

class **NewtonSolverSparse** : public amici::*NewtonSolver*

The *NewtonSolverSparse* provides access to the sparse linear solver for the Newton method.

Public Functions

explicit **NewtonSolverSparse**(const *Model* &model)

constructor for dense solver

Parameters **model** – model instance that provides problem dimensions

NewtonSolverSparse(const *NewtonSolverSparse*&) = delete

NewtonSolverSparse &**operator=**(const *NewtonSolverSparse* &other) = delete

~NewtonSolverSparse() override

virtual void **solveLinearSystem**(*AmiVector* &rhs) override

Solves the linear system for the Newton step.

Parameters **rhs** – containing the RHS of the linear system, will be overwritten by solution to the linear system

virtual void **prepareLinearSystem**(*Model* &model, const *SimulationState* &state) override

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual void **prepareLinearSystemB**(*Model* &model, const *SimulationState* &state) override

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

virtual bool **is_singular**(*Model* &model, const *SimulationState* &state) const override

Checks whether linear system is singular.

Parameters

- **model** – pointer to the model instance
- **state** – current simulation state

Returns boolean indicating whether the linear system is singular (condition number < 1/machine precision)

virtual void **reinitialize**() override

Reinitialize the linear solver.

Class ReturnData

- Defined in file_include_amici_rdata.h

Inheritance Relationships

Base Type

- `public amici::ModelDimensions` (*Struct ModelDimensions*)

Class Documentation

class **ReturnData** : public amici::*ModelDimensions*

Stores all data to be returned by amici::runAmiciSimulation.

NOTE: multi-dimensional arrays are stored in row-major order (C-style)

Public Functions

ReturnData() = default

Default constructor.

ReturnData(std::vector<*realtype*> ts, *ModelDimensions* const &model_dimensions, int nplist, int nmaxevent, int nt, int newton_maxsteps, std::vector<*ParameterScaling*> pscale, *SecondOrderMode* o2mode, *SensitivityOrder* sensi, *SensitivityMethod* sensi_meth, *RDataReporting* rdrm, bool quadratic_llh, bool sigma_res, *realtype* sigma_offset)

Constructor.

Parameters

- **ts** – see amici::SimulationParameters::ts
- **model_dimensions** – *Model* dimensions
- **nplist** – see amici::ModelDimensions::nplist
- **nmaxevent** – see amici::ModelDimensions::nmaxevent
- **nt** – see amici::ModelDimensions::nt
- **newton_maxsteps** – see amici::Solver::newton_maxsteps
- **pscale** – see *amici::SimulationParameters::pscale*
- **o2mode** – see amici::SimulationParameters::o2mode
- **sensi** – see amici::Solver::sensi
- **sensi_meth** – see amici::Solver::sensi_meth
- **rdrm** – see amici::Solver::rdata_reporting
- **quadratic_llh** – whether model defines a quadratic nllh and computing res, sres and FIM makes sense
- **sigma_res** – indicates whether additional residuals are to be added for each sigma
- **sigma_offset** – offset to ensure real-valuedness of sigma residuals

ReturnData(*Solver* const &solver, const *Model* &model)

constructor that uses information from model and solver to appropriately initialize fields

Parameters

- **solver** – solver instance
- **model** – model instance

~ReturnData() = default

void **processSimulationObjects**(*SteadystateProblem* const *preeq, *ForwardProblem* const *fwd, *BackwardProblem* const *bwd, *SteadystateProblem* const *posteq, *Model* &model, *Solver* const &solver, *ExpData* const *edata)

constructor that uses information from model and solver to appropriately initialize fields

Parameters

- **preeq** – simulated preequilibration problem, pass `nullptr` to ignore
- **fwd** – simulated forward problem, pass `nullptr` to ignore
- **bwd** – simulated backward problem, pass `nullptr` to ignore
- **posteq** – simulated postequilibration problem, pass `nullptr` to ignore
- **model** – matching model instance
- **solver** – matching solver instance
- **edata** – matching experimental data

Public Members

std::string **id**

Arbitrary (not necessarily unique) identifier.

std::vector<*realtype*> **ts**

timepoints (shape `nt`)

std::vector<*realtype*> **xdot**

time derivative (shape `nx`)

std::vector<*realtype*> **J**

Jacobian of differential equation right hand side (shape `nx` x `nx`, row-major)

std::vector<*realtype*> **w**

w data from the model (recurring terms in xdot, for imported SBML models from python, this contains the flux vector) (shape `nt` x `nw`, row major)

std::vector<*realtype*> **z**

event output (shape `nmaxevent` x `nz`, row-major) std::vector<*realtype*> **sigmaz**

event output sigma standard deviation (shape `nmaxevent` x `nz`, row-major)

std::vector<*realtype*> **sz**

parameter derivative of event output (shape `nmaxevent` x `nplist` x `nz`, row-major)

std::vector<*realtype*> **ssigmaz**

parameter derivative of event output standard deviation (shape `nmaxevent` x `nplist` x `nz`, row-major)

std::vector<*realtype*> **rz**

event trigger output (shape `nmaxevent` x `nz`, row-major)

std::vector<*realtype*> **srz**

parameter derivative of event trigger output (shape `nmaxevent` x `nplist` x `nz`, row-major)

std::vector<*realtype*> **s2rz**

second-order parameter derivative of event trigger output (shape `nmaxevent` x `nztrue` x `nplist` x `nplist`, row-major)

std::vector<*realtype*> **x**

state (shape `nt` x `nx`, row-major)

std::vector<*realtype*> **sx**

parameter derivative of state (shape `nt` x `nplist` x `nx`, row-major)

std::vector<*realtype*> **y**

observable (shape `nt` x `ny`, row-major)

std::vector<*realtype*> **sigmay**

observable standard deviation (shape `nt` x `ny`, row-major)

std::vector<*realtype*> **sy**

parameter derivative of observable (shape `nt` x `nplist` x `ny`, row-major)

std::vector<*realtype*> **ssigmay**

parameter derivative of observable standard deviation (shape `nt` x `nplist` x `ny`, row-major)

std::vector<*realtype*> **res**

observable (shape `nt*ny`, row-major)

std::vector<*realtype*> **sres**

parameter derivative of residual (shape `nt*ny` x `nplist`, row-major)

std::vector<*realtype*> **FIM**

fisher information matrix (shape `nplist` x `nplist`, row-major) std::vector<int> **numsteps**

number of integration steps forward problem (shape nt)

std::vector<int> **numstepsB**

number of integration steps backward problem (shape nt)

std::vector<int> **numrhsevals**

number of right hand side evaluations forward problem (shape nt)

std::vector<int> **numrhsevalsB**

number of right hand side evaluations backward problem (shape nt)

std::vector<int> **numerrtestfails**

number of error test failures forward problem (shape nt)

std::vector<int> **numerrtestfailsB**

number of error test failures backward problem (shape nt)

std::vector<int> **numnonlinsolvconvfails**

number of linear solver convergence failures forward problem (shape nt)

std::vector<int> **numnonlinsolvconvfailsB**

number of linear solver convergence failures backward problem (shape nt)

std::vector<int> **order**

employed order forward problem (shape nt)

double **cpu_time** = 0.0

computation time of forward solve [ms]

double **cpu_timeB** = 0.0

computation time of backward solve [ms]

double **cpu_time_total** = 0.0

total CPU time from entering runAmiciSimulation until exiting [ms]

std::vector<*SteadyStateStatus*> **preeq_status**

flags indicating success of steady state solver (preequilibration)

double **preeq_cpu_time** = 0.0

computation time of the steady state solver [ms] (preequilibration)

double **preeq_cpu_timeB** = 0.0

computation time of the steady state solver of the backward problem [ms] (preequilibration) std::vector<*SteadyStateStatus*> **posteq_status**

flags indicating success of steady state solver (postequilibration)

double **posteq_cpu_time** = 0.0

computation time of the steady state solver [ms] (postequilibration)

double **posteq_cpu_timeB** = 0.0

computation time of the steady state solver of the backward problem [ms] (postequilibration)

std::vector<int> **preeq_numsteps**

number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3)

int **preeq_numstepsB** = 0

number of simulation steps for adjoint steady state problem (preequilibration) [== 0 if analytical solution worked, > 0 otherwise]

std::vector<int> **posteq_numsteps**

number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (shape 3) (postequilibration)

int **posteq_numstepsB** = 0

number of simulation steps for adjoint steady state problem (postequilibration) [== 0 if analytical solution worked, > 0 otherwise]

realtype **preeq_t** = NAN

time when steadystate was reached via simulation (preequilibration)

realtype **preeq_wrms** = NAN

weighted root-mean-square of the rhs when steadystate was reached (preequilibration)

realtype **posteq_t** = NAN

time when steadystate was reached via simulation (postequilibration)

realtype **posteq_wrms** = NAN

weighted root-mean-square of the rhs when steadystate was reached (postequilibration)

std::vector<*realtype*> **x0**

initial state (shape `nx`)

std::vector<*realtype*> **x_ss**

preequilibration steady state found by Newton solver (shape `nx`)

std::vector<*realtype*> **sx0**

initial sensitivities (shape `nplist` x `nx`, row-major) std::vector<*realtype*> **sx_ss**

preequilibration sensitivities found by Newton solver (shape `nplist` x `nx`, row-major)

realtype **llh** = 0.0

log-likelihood value

realtype **chi2** = 0.0

χ^2 value

std::vector<*realtype*> **sllh**

parameter derivative of log-likelihood (shape `nplist`)

std::vector<*realtype*> **s2llh**

second-order parameter derivative of log-likelihood (shape `nJ-1` x `nplist`, row-major)

int **status** = 0

Simulation status code.

One of:

- AMICI_SUCCESS, indicating successful simulation
- AMICI_MAX_TIME_EXCEEDED, indicating that the simulation did not finish within the allowed time (see *Solver*.{set,get}MaxTime)
- AMICI_ERROR, indicating that some error occurred during simulation (a more detailed error message will have been printed).

int **nx** = {0}

number of states (alias `nx_rdata`, kept for backward compatibility)

int **nxtrue** = {0}

number of states in the unaugmented system (alias nxtrue_rdata, kept for backward compatibility)

int **nplist** = {0}

number of parameter for which sensitivities were requested

int **nmaxevent** = {0}

maximal number of occurring events (for every event type)

int **nt** = {0}

number of considered timepoints

int **newton_maxsteps** = {0}

maximal number of newton iterations for steady state calculation std::vector<*ParameterScaling*> **pscale**

scaling of parameterization

SecondOrderMode **o2mode** = {*SecondOrderMode*::*none*}

flag indicating whether second-order sensitivities were requested

SensitivityOrder **sensi** = {*SensitivityOrder*::*none*}

sensitivity order

SensitivityMethod **sensi_meth** = {*SensitivityMethod*::*none*}

sensitivity method

RDataReporting **rdata_reporting** = {*RDataReporting*::*full*}

reporting mode

bool **sigma_res**

boolean indicating whether residuals for standard deviations have been added

std::vector<*LogItem*> **messages**

log messages

Protected Functions

void **initializeLikelihoodReporting**(bool quadratic_llh)

initializes storage for likelihood reporting mode

Parameters **quadratic_llh** – whether model defines a quadratic nllh and computing res, sres and FIM makes sense.

void **initializeResidualReporting**(bool enable_res)

initializes storage for residual reporting mode

Parameters **enable_res** – whether residuals are to be computed

void **initializeFullReporting**(bool enable_fim)

initializes storage for full reporting mode

Parameters **enable_fim** – whether FIM Hessian approximation is to be computed

void **initializeObjectiveFunction**(bool enable_chi2)

initialize values for chi2 and llh and derivatives

Parameters **enable_chi2** – whether chi2 values are to be computed

void **processPreEquilibration**(*SteadystateProblem* const &preeq, *Model* &model)

extracts data from a preequilibration *SteadystateProblem*

Parameters

- **preeq** – *SteadystateProblem* for preequilibration
- **model** – *Model* instance to compute return values

void **processPostEquilibration**(*SteadystateProblem* const &posteq, *Model* &model, *ExpData* const *edata)

extracts data from a preequilibration *SteadystateProblem*

Parameters

- **posteq** – *SteadystateProblem* for postequilibration
- **model** – *Model* instance to compute return values
- **edata** – *ExpData* instance containing observable data

void **processForwardProblem**(*ForwardProblem* const &fwd, *Model* &model, *ExpData* const *edata)

extracts results from forward problem

Parameters

- **fwd** – forward problem
- **model** – model that was used for forward simulation
- **edata** – *ExpData* instance containing observable data

void **processBackwardProblem**(*ForwardProblem* const &fwd, *BackwardProblem* const &bwd, *SteadystateProblem* const *preeq, *Model* &model)

extracts results from backward problem

Parameters

- **fwd** – forward problem
- **bwd** – backward problem
- **preeq** – *SteadystateProblem* for preequilibration
- **model** – model that was used for forward/backward simulation

void **processSolver**(*Solver* const &solver)

extracts results from solver

Parameters **solver** – solver that was used for forward/backward simulation

template<class T>
inline void **storeJacobianAndDerivativeInReturnData**(*T* const &problem, *Model* &model)

Evaluates and stores the Jacobian and right hand side at final timepoint.

Parameters

- **problem** – forward problem or steadystate problem
- **model** – model that was used for forward/backward simulation

void **readSimulationState**(*SimulationState* const &state, *Model* &model)

sets member variables and model state according to provided simulation state

Parameters

- **state** – simulation state provided by Problem
- **model** – model that was used for forward/backward simulation

void **fres**(int it, *Model* &model, const *ExpData* &edata)

Residual function.

Parameters

- **it** – time index
- **model** – model that was used for forward/backward simulation
- **edata** – *ExpData* instance containing observable data

void **fchi2**(int it, const *ExpData* &edata)

Chi-squared function.

Parameters

- **it** – time index
- **edata** – *ExpData* instance containing observable data

void **fsres**(int it, *Model* &model, const *ExpData* &edata)

Residual sensitivity function.

Parameters

- **it** – time index
- **model** – model that was used for forward/backward simulation
- **edata** – *ExpData* instance containing observable data

void **fFIM**(int it, *Model* &model, const *ExpData* &edata)

Fisher information matrix function.

Parameters

- **it** – time index
- **model** – model that was used for forward/backward simulation
- **edata** – *ExpData* instance containing observable data

void **invalidate**(int it_start)

Set likelihood, state variables, outputs and respective sensitivities to NaN (typically after integration failure)

Parameters **it_start** – time index at which to start invalidating

void **invalidateLLH**()

Set likelihood and chi2 to NaN (typically after integration failure)

void **invalidateSLLH**()

Set likelihood sensitivities to NaN (typically after integration failure)

void **applyChainRuleFactorToSimulationResults**(const *Model* &model)

applies the chain rule to account for parameter transformation in the sensitivities of simulation results

Parameters **model** – *Model* from which the *ReturnData* was obtained

inline bool **computingFSA**() const

Checks whether forward sensitivity analysis is performed.

Returns boolean indicator

void **getDataOutput**(int it, *Model* &model, *ExpData* const *edata)

Extracts output information for data-points, expects that x_solver_ and sx_solver_ were set appropriately.

Parameters

- **it** – timepoint index
- **model** – model that was used in forward solve

- **edata** – *ExpData* instance carrying experimental data

void **getDataSensisFSA**(int it, *Model* &model, *ExpData* const *edata)

Extracts data information for forward sensitivity analysis, expects that x_solver_ and sx_solver_ were set appropriately.

Parameters

- **it** – index of current timepoint
- **model** – model that was used in forward solve
- **edata** – *ExpData* instance carrying experimental data

void **getEventOutput**(*realtype* t, const std::vector<int> rootidx, *Model* &model, *ExpData* const *edata)

Extracts output information for events, expects that x_solver_ and sx_solver_ were set appropriately.

Parameters

- **t** – event timepoint
- **rootidx** – information about which roots fired (1 indicating fired, 0/-1 for not)
- **model** – model that was used in forward solve
- **edata** – *ExpData* instance carrying experimental data

void **getEventSensisFSA**(int ie, *realtype* t, *Model* &model, *ExpData* const *edata)

Extracts event information for forward sensitivity analysis, expects that x_solver_ and sx_solver_ were set appropriately.

Parameters

- **ie** – index of event type
- **t** – event timepoint
- **model** – model that was used in forward solve
- **edata** – *ExpData* instance carrying experimental data

void **handleSx0Backward**(const *Model* &model, *SteadystateProblem* const &preeq, std::vector<*realtype*> &llhS0, *AmiVector* &xQB) const

Updates contribution to likelihood from quadratures (xQB), if preequilibration was run in adjoint mode.

Parameters

- **model** – model that was used for forward/backward simulation
- **preeq** – *SteadystateProblem* for preequilibration
- **llhS0** – contribution to likelihood for initial state sensitivities of preequilibration
- **xQB** – vector with quadratures from adjoint computation

void **handleSx0Forward**(const *Model* &model, std::vector<*realtype*> &llhS0, *AmiVector* &xB) const

Updates contribution to likelihood for initial state sensitivities (llhS0), if no preequilibration was run or if forward sensitivities were used.

Parameters

- **model** – model that was used for forward/backward simulation
- **llhS0** – contribution to likelihood for initial state sensitivities
- **xB** – vector with final adjoint state (excluding conservation laws)

Protected Attributes

realtype **sigma_offset**

offset for sigma_residuals

realtype **t_**

timepoint for model evaluation

AmiVector **x_solver_**

partial state vector, excluding states eliminated from conservation laws

AmiVector **dx_solver_**

partial time derivative of state vector, excluding states eliminated from conservation laws

AmiVectorArray **sx_solver_**

partial sensitivity state vector array, excluding states eliminated from conservation laws

AmiVector **x_rdata_**

full state vector, including states eliminated from conservation laws

AmiVectorArray **sx_rdata_**

full sensitivity state vector array, including states eliminated from conservation laws

std::vector<int> **nroots_**

array of number of found roots for a certain event type (shape `ne`)

Friends

template<class **Archive**>
friend void **serialize**(*Archive* &ar, *ReturnData* &r, unsigned int version)

Serialize *ReturnData* (see boost::serialization::serialize)

Parameters

- **ar** – Archive to serialize to
- **r** – Data to serialize
- **version** – Version number

Class SetupFailure

- Defined in file_include_amici_exception.h

Inheritance Relationships

Base Type

- `public amici::AmiException` (*Class AmiException*)

Class Documentation

class **SetupFailure** : public amici::*AmiException*

Setup failure exception.

This exception should be thrown when the solver setup failed for this exception we can assume that we cannot recover from the exception and an error will be thrown

Public Functions

explicit **SetupFailure**(char const *fmt, ...)

Constructor with printf style interface.

Parameters

- **fmt** – error message with printf format
- **...** – printf formatting variables

Class SimulationParameters

- Defined in file_include_amici_simulation_parameters.h

Inheritance Relationships

Derived Type

- `public amici::ExpData` (*Class ExpData*)

Class Documentation

class **SimulationParameters**

Container for various simulation parameters.

Subclassed by *amici::ExpData*

Public Functions

SimulationParameters() = default

inline explicit **SimulationParameters**(std::vector<*realtype*> timepoints)

Constructor.

Parameters **timepoints** – Timepoints for which simulation results are requested

inline **SimulationParameters**(std::vector<*realtype*> fixedParameters, std::vector<*realtype*> parameters)

Constructor.

Parameters

- **fixedParameters** – *Model* constants
- **parameters** – *Model* parameters

inline **SimulationParameters**(std::vector<*realtype*> fixedParameters, std::vector<*realtype*> parameters, std::vector<int> plist)

Constructor.

Parameters

- **fixedParameters** – *Model* constants
- **parameters** – *Model* parameters
- **plist** – *Model* parameter indices w.r.t. which sensitivities are to be computed

inline **SimulationParameters**(std::vector<*realtype*> timepoints, std::vector<*realtype*> fixedParameters, std::vector<*realtype*> parameters)

Constructor.

Parameters

- **timepoints** – Timepoints for which simulation results are requested
- **fixedParameters** – *Model* constants
- **parameters** – *Model* parameters

void **reinitializeAllFixedParameterDependentInitialStatesForPresimulation**(int nx_rdata)

Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).

Convenience function to populate `reinitialization_state_idxs_presim` and `reinitialization_state_idxs_sim`

Parameters **nx_rdata** – Number of states (*Model::nx_rdata*)

void **reinitializeAllFixedParameterDependentInitialStatesForSimulation**(int nx_rdata)

Set reinitialization of all states based on model constants for the 'main' simulation (only meaningful if presimulation or preequilibration is performed).

Convenience function to populate `reinitialization_state_idxs_presim` and `reinitialization_state_idxs_sim`

Parameters **nx_rdata** – Number of states (*Model::nx_rdata*) void **reinitializeAllFixedParameterDependentInitialStates**(int nx_rdata)

Set reinitialization of all states based on model constants for all simulation phases.

Convenience function to populate `reinitialization_state_idxs_presim` and `reinitialization_state_idxs_sim`

Parameters **nx_rdata** – Number of states (*Model::nx_rdata*)

Public Members

std::vector<*realtype*> **fixedParameters**

Model constants.

Vector of size *Model::nk()* or empty

std::vector<*realtype*> **fixedParametersPreequilibration**

Model constants for pre-equilibration.

Vector of size *Model::nk()* or empty.

std::vector<*realtype*> **fixedParametersPresimulation**

Model constants for pre-simulation.

Vector of size *Model::nk()* or empty.

std::vector<*realtype*> **parameters**

Model parameters.

Vector of size *Model::np()* or empty with parameter scaled according to SimulationParameter::pscale.

std::vector<*realtype*> **x0**

Initial state.

Vector of size Model::nx() or empty

std::vector<*realtype*> **sx0**

Initial state sensitivities.

Dimensions: Model::nx() * *Model::nplist()*, Model::nx() * ExpData::plist.size(), if *ExpData::plist* is not empty, or empty

std::vector<*ParameterScaling*> **pscale**

Parameter scales.

Vector of parameter scale of size *Model::np()*, indicating how/if each parameter is to be scaled.

std::vector<int> **plist**

Parameter indices w.r.t. which to compute sensitivities.

realtype **tstart_** = {0.0}

starting time

realtype **t_presim** = {0.0}

Duration of pre-simulation.

If this is > 0, presimulation will be performed from (model->t0 - t_presim) to model->t0 using the fixedParameters in fixedParametersPresimulation

std::vector<*realtype*> **ts_**

Timepoints for which model state/outputs/… are requested.

Vector of timepoints.

bool **reinitializeFixedParameterInitialStates** = {false}

Flag indicating whether reinitialization of states depending on fixed parameters is activated.

std::vector<int> **reinitialization_state_idxs_presim**

Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.

std::vector<int> **reinitialization_state_idxs_sim**

Indices of states to be reinitialized based on provided constants / fixed parameters.

Class Solver

- Defined in file_include_amici_solver.h

Inheritance Relationships

Derived Types

- `public amici::CVodeSolver` (*Class CVodeSolver*)
- `public amici::IDASolver` (*Class IDASolver*)

Class Documentation

class **Solver**

The *Solver* class provides a generic interface to CVODES and IDAS solvers, individual realizations are realized in the *CVodeSolver* and the *IDASolver* class. All transient private/protected members (CVODES/IDAS memory, interface variables and status flags) are specified as mutable and not included in serialization or equality checks. No solver setting parameter should be marked mutable.

NOTE: Any changes in data members here must be propagated to copy ctor, equality operator, serialization functions in serialization.h, and amici::hdf5::(read/write)SolverSettings(From/To)HDF5 in hdf5.cpp.

Subclassed by *amici::CVodeSolver*, *amici::IDASolver*

Public Types

using **user_data_type** = std::pair<*Model**, *Solver* const*>

Type of what is passed to Sundials solvers as user_data

Public Functions

Solver() = default

Default constructor.

Solver(const *Solver* &other)

Solver copy constructor.

Parameters **other** –

virtual **~Solver**() = default

virtual *Solver* ***clone**() const = 0

Clone this instance.

Returns The clone

int **run**(*realtype* tout) const

runs a forward simulation until the specified timepoint

Parameters **tout** – next timepoint

Returns status flag

int **step**(*realtype* tout) const

makes a single step in the simulation

Parameters **tout** – next timepoint

Returns status flag

void **runB**(*realtype* tout) const

runs a backward simulation until the specified timepoint

Parameters **tout** – next timepoint

void **setup**(*realtype* t0, *Model* *model, const *AmiVector* &x0, const *AmiVector* &dx0, const *AmiVectorArray* &sx0, const *AmiVectorArray* &sdx0) const

Initializes the ami memory object and applies specified options.

Parameters

- **t0** – initial timepoint
- **model** – pointer to the model instance
- **x0** – initial states
- **dx0** – initial derivative states
- **sx0** – initial state sensitivities
- **sdx0** – initial derivative state sensitivities

void **setupB**(int *which, *realtype* tf, *Model* *model, const *AmiVector* &xB0, const *AmiVector* &dxB0, const *AmiVector* &xQB0) const

Initializes the AMI memory object for the backwards problem.

Parameters

- **which** – index of the backward problem, will be set by this routine
- **tf** – final timepoint (initial timepoint for the bwd problem)
- **model** – pointer to the model instance
- **xB0** – initial adjoint states
- **dxB0** – initial adjoint derivative states
- **xQB0** – initial adjoint quadratures

void **setupSteadystate**(const *realtype* t0, *Model* *model, const *AmiVector* &x0, const *AmiVector* &dx0, const *AmiVector* &xB0, const *AmiVector* &dxB0, const *AmiVector* &xQ0) const

Initializes the ami memory for quadrature computation.

Parameters

- **t0** – initial timepoint
- **model** – pointer to the model instance
- **x0** – initial states
- **dx0** – initial derivative states
- **xB0** – initial adjoint states
- **dxB0** – initial derivative adjoint states
- **xQ0** – initial quadrature vector

void **updateAndReinitStatesAndSensitivities**(*Model* *model) const

Reinitializes state and respective sensitivities (if necessary) according to changes in fixedParameters.

Parameters **model** – pointer to the model instance

virtual void **getRootInfo**(int *rootsfound) const = 0

getRootInfo extracts information which event occurred

Parameters **rootsfound** – array with flags indicating whether the respective event occurred

virtual void **calcIC**(*realtype* tout1) const = 0

Calculates consistent initial conditions, assumes initial states to be correct (DAE only)

Parameters **tout1** – next timepoint to be computed (sets timescale)

virtual void **calcICB**(int which, *realtype* tout1) const = 0

Calculates consistent initial conditions for the backwards problem, assumes initial states to be correct (DAE only)

Parameters

- **which** – identifier of the backwards problem
- **tout1** – next timepoint to be computed (sets timescale)

virtual void **solveB**(*realtype* tBout, int itaskB) const = 0

Solves the backward problem until a predefined timepoint (adjoint only)

Parameters

- **tBout** – timepoint until which simulation should be performed
- **itaskB** – task identifier, can be CV_NORMAL or CV_ONE_STEP

virtual void **turnOffRootFinding**() const = 0

Disable rootfinding.

SensitivityMethod **getSensitivityMethod**() const

Return current sensitivity method.

Returns method enum

void **setSensitivityMethod**(*SensitivityMethod* sensi_meth)

Set sensitivity method.

Parameters **sensi_meth** –

SensitivityMethod **getSensitivityMethodPreequilibration**() const

Return current sensitivity method during preequilibration.

Returns method enum

void **setSensitivityMethodPreequilibration**(*SensitivityMethod* sensi_meth_preeq)

Set sensitivity method for preequilibration.

Parameters **sensi_meth_preeq** –

void **switchForwardSensisOff**() const

Disable forward sensitivity integration (used in steady state sim)

int **getNewtonMaxSteps**() const

Get maximum number of allowed Newton steps for steady state computation.

Returns

void **setNewtonMaxSteps**(int newton_maxsteps)

Set maximum number of allowed Newton steps for steady state computation.

Parameters **newton_maxsteps** –

NewtonDampingFactorMode **getNewtonDampingFactorMode**() const

Get a state of the damping factor used in the Newton solver.

Returns

void **setNewtonDampingFactorMode**(*NewtonDampingFactorMode* dampingFactorMode)

Turn on/off a damping factor in the Newton method.

Parameters **dampingFactorMode** –

double **getNewtonDampingFactorLowerBound**() const

Get a lower bound of the damping factor used in the Newton solver.

Returns

void **setNewtonDampingFactorLowerBound**(double dampingFactorLowerBound)

Set a lower bound of the damping factor in the Newton solver.

Parameters **dampingFactorLowerBound** –

SensitivityOrder **getSensitivityOrder**() const

Get sensitivity order.

Returns sensitivity order

void **setSensitivityOrder**(*SensitivityOrder* sensi)

Set the sensitivity order.

Parameters **sensi** – sensitivity order

double **getRelativeTolerance**() const

Get the relative tolerances for the forward problem.

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Returns relative tolerances

void **setRelativeTolerance**(double rtol)

Sets the relative tolerances for the forward problem.

Same tolerance is used for the backward problem if not specified differently via setRelativeToleranceASA.

Parameters **rtol** – relative tolerance (non-negative number)

double **getAbsoluteTolerance**() const

Get the absolute tolerances for the forward problem.

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Returns absolute tolerances

void **setAbsoluteTolerance**(double atol)

Sets the absolute tolerances for the forward problem.

Same tolerance is used for the backward problem if not specified differently via setAbsoluteToleranceASA.

Parameters **atol** – absolute tolerance (non-negative number)

double **getRelativeToleranceFSA**() const

Returns the relative tolerances for the forward sensitivity problem.

Returns relative tolerances

void **setRelativeToleranceFSA**(double rtol)

Sets the relative tolerances for the forward sensitivity problem.

Parameters **rtol** – relative tolerance (non-negative number)

double **getAbsoluteToleranceFSA**() const

Returns the absolute tolerances for the forward sensitivity problem.

Returns absolute tolerances

void **setAbsoluteToleranceFSA**(double atol)

Sets the absolute tolerances for the forward sensitivity problem.

Parameters **atol** – absolute tolerance (non-negative number) double **getRelativeToleranceB**() const

Returns the relative tolerances for the adjoint sensitivity problem.

Returns relative tolerances

void **setRelativeToleranceB**(double rtol)

Sets the relative tolerances for the adjoint sensitivity problem.

Parameters `rtol` – relative tolerance (non-negative number)

double **getAbsoluteToleranceB**() const

Returns the absolute tolerances for the backward problem for adjoint sensitivity analysis.

Returns absolute tolerances

void **setAbsoluteToleranceB**(double atol)

Sets the absolute tolerances for the backward problem for adjoint sensitivity analysis.

Parameters `atol` – absolute tolerance (non-negative number)

double **getRelativeToleranceQuadratures**() const

Returns the relative tolerance for the quadrature problem.

Returns relative tolerance

void **setRelativeToleranceQuadratures**(double rtol)

sets the relative tolerance for the quadrature problem

Parameters `rtol` – relative tolerance (non-negative number)

double **getAbsoluteToleranceQuadratures**() const

returns the absolute tolerance for the quadrature problem

Returns absolute tolerance

void **setAbsoluteToleranceQuadratures**(double atol)

sets the absolute tolerance for the quadrature problem

Parameters `atol` – absolute tolerance (non-negative number)

double **getSteadyStateToleranceFactor**() const

returns the steady state simulation tolerance factor.

Steady state simulation tolerances are the product of the simulation tolerances and this factor, unless manually set with `set(Absolute/Relative)ToleranceSteadyState()`.

Returns steady state simulation tolerance factor

void **setSteadyStateToleranceFactor**(double factor)

set the steady state simulation tolerance factor.

Steady state simulation tolerances are the product of the simulation tolerances and this factor, unless manually set with `set(Absolute/Relative)ToleranceSteadyState()`.

Parameters `factor` – tolerance factor (non-negative number)

double **getRelativeToleranceSteadyState**() const

returns the relative tolerance for the steady state problem

Returns relative tolerance

void **setRelativeToleranceSteadyState**(double rtol)

sets the relative tolerance for the steady state problem

Parameters **rtol** – relative tolerance (non-negative number)

double **getAbsoluteToleranceSteadyState**() const

returns the absolute tolerance for the steady state problem

Returns absolute tolerance

void **setAbsoluteToleranceSteadyState**(double atol)

sets the absolute tolerance for the steady state problem

Parameters **atol** – absolute tolerance (non-negative number)

double **getSteadyStateSensiToleranceFactor**() const

returns the steady state sensitivity simulation tolerance factor.

Steady state sensitivity simulation tolerances are the product of the sensitivity simulation tolerances and this factor, unless manually set with `set(Absolute/Relative)ToleranceSteadyStateSensi()`.

Returns steady state simulation tolerance factor

void **setSteadyStateSensiToleranceFactor**(double factor)

set the steady state sensitivity simulation tolerance factor.

Steady state sensitivity simulation tolerances are the product of the sensitivity simulation tolerances and this factor, unless manually set with `set(Absolute/Relative)ToleranceSteadyStateSensi()`.

Parameters **factor** – tolerance factor (non-negative number)

double **getRelativeToleranceSteadyStateSensi**() const

returns the relative tolerance for the sensitivities of the steady state problem

Returns relative tolerance

void **setRelativeToleranceSteadyStateSensi**(double rtol)

sets the relative tolerance for the sensitivities of the steady state problem

Parameters **rtol** – relative tolerance (non-negative number)

double **getAbsoluteToleranceSteadyStateSensi**() const

returns the absolute tolerance for the sensitivities of the steady state problem

Returns absolute tolerance

void **setAbsoluteToleranceSteadyStateSensi**(double atol)

sets the absolute tolerance for the sensitivities of the steady state problem

Parameters **atol** – absolute tolerance (non-negative number)

long int **getMaxSteps**() const

returns the maximum number of solver steps for the forward problem

Returns maximum number of solver steps

void **setMaxSteps**(long int maxsteps)

sets the maximum number of solver steps for the forward problem

Parameters **maxsteps** – maximum number of solver steps (positive number) double **getMaxTime**() const

Returns the maximum time allowed for integration.

Returns Time in seconds

void **setMaxTime**(double maxtime)

Set the maximum time allowed for integration.

Parameters **maxtime** – Time in seconds

void **startTimer**() const

Start timer for tracking integration time.

bool **timeExceeded**() const

Check whether maximum integration time was exceeded.

Returns True if the maximum integration time was exceeded, false otherwise.

long int **getMaxStepsBackwardProblem**() const

returns the maximum number of solver steps for the backward problem

Returns maximum number of solver steps

void **setMaxStepsBackwardProblem**(long int maxsteps)

sets the maximum number of solver steps for the backward problem

Note: default behaviour (100 times the value for the forward problem) can be restored by passing maxsteps=0

Parameters **maxsteps** – maximum number of solver steps (non-negative number)

LinearMultistepMethod **getLinearMultistepMethod**() const

returns the linear system multistep method

Returns linear system multistep method

void **setLinearMultistepMethod**(*LinearMultistepMethod* lmm)

sets the linear system multistep method

Parameters **lmm** – linear system multistep method

NonlinearSolverIteration **getNonlinearSolverIteration**() const

returns the nonlinear system solution method

Returns

void **setNonlinearSolverIteration**(*NonlinearSolverIteration* iter)

sets the nonlinear system solution method

Parameters **iter** – nonlinear system solution method

InterpolationType **getInterpolationType**() const

getInterpolationType

Returns

void **setInterpolationType**(*InterpolationType* interpType)

sets the interpolation of the forward solution that is used for the backwards problem

Parameters **interpType** – interpolation type

int **getStateOrdering**() const

Gets KLU / SuperLUMT state ordering mode.

Returns State-ordering as integer according to *SUNLinSolKLU::StateOrdering* or SUNLinSol-SuperLUMT::StateOrdering (which differ).

void **setStateOrdering**(int ordering)

Sets KLU / SuperLUMT state ordering mode.

This only applies when linsol is set to LinearSolver::KLU or LinearSolver::SuperLUMT. Mind the difference between *SUNLinSolKLU::StateOrdering* and SUNLinSolSuperLUMT::StateOrdering.

Parameters **ordering** – state ordering

bool **getStabilityLimitFlag**() const

returns stability limit detection mode

Returns stldet can be false (deactivated) or true (activated)

void **setStabilityLimitFlag**(bool stldet)

set stability limit detection mode

Parameters **stldet** – can be false (deactivated) or true (activated)

LinearSolver **getLinearSolver**() const

getLinearSolver

Returns

void **setLinearSolver**(*LinearSolver* linsol)

setLinearSolver

Parameters **linsol** –

InternalSensitivityMethod **getInternalSensitivityMethod**() const

returns the internal sensitivity method

Returns internal sensitivity method

void **setInternalSensitivityMethod**(*InternalSensitivityMethod* ism)

sets the internal sensitivity method

Parameters **ism** – internal sensitivity method

RDataReporting **getReturnDataReportingMode**() const

returns the *ReturnData* reporting mode

Returns *ReturnData* reporting mode

void **setReturnDataReportingMode**(*RDataReporting* rdrm)

sets the *ReturnData* reporting mode

Parameters **rdrm** – *ReturnData* reporting mode

void **writeSolution**(*realtype* *t, *AmiVector* &x, *AmiVector* &dx, *AmiVectorArray* &sx, *AmiVector* &xQ) const

write solution from forward simulation

Parameters

- **t** – time
- **x** – state
- **dx** – derivative state
- **sx** – state sensitivity
- **xQ** – quadrature

void **writeSolutionB**(*realtype* *t, *AmiVector* &xB, *AmiVector* &dxB, *AmiVector* &xQB, int which) const

write solution from backward simulation

Parameters

- **t** – time
- **xB** – adjoint state
- **dxB** – adjoint derivative state
- **xQB** – adjoint quadrature
- **which** – index of adjoint problem

const *AmiVector* &**getState**(*realtype* t) const

Access state solution at time t.

Parameters **t** – time

Returns x or interpolated solution dky

const *AmiVector* &**getDerivativeState**(*realtype* t) const

Access derivative state solution at time t.

Parameters **t** – time

Returns dx or interpolated solution dky

const *AmiVectorArray* &**getStateSensitivity**(*realtype* t) const

Access state sensitivity solution at time t.

Parameters **t** – time

Returns (interpolated) solution sx

const *AmiVector* &**getAdjointState**(int which, *realtype* t) const

Access adjoint solution at time t.

Parameters

- **which** – adjoint problem index
- **t** – time

Returns (interpolated) solution xB

const *AmiVector* &**getAdjointDerivativeState**(int which, *realtype* t) const

Access adjoint derivative solution at time t.

Parameters

- **which** – adjoint problem index
- **t** – time

Returns (interpolated) solution dxB

const *AmiVector* &**getAdjointQuadrature**(int which, *realtype* t) const

Access adjoint quadrature solution at time t.

Parameters

- **which** – adjoint problem index
- **t** – time

Returns (interpolated) solution xQB

const *AmiVector* &**getQuadrature**(*realtype* t) const

Access quadrature solution at time t.

Parameters **t** – time

Returns (interpolated) solution xQ

virtual void **reInit**(*realtype* t0, const *AmiVector* &yy0, const *AmiVector* &yp0) const = 0

Reinitializes the states in the solver after an event occurrence.

Parameters

- **t0** – reinitialization timepoint
- **yy0** – initial state variables
- **yp0** – initial derivative state variables (DAE only)

virtual void **sensReInit**(const *AmiVectorArray* &yyS0, const *AmiVectorArray* &ypS0) const = 0

Reinitializes the state sensitivities in the solver after an event occurrence.

Parameters

- **yyS0** – new state sensitivity
- **ypS0** – new derivative state sensitivities (DAE only)

virtual void **sensToggleOff**() const = 0

Switches off computation of state sensitivities without deallocating the memory for sensitivities.

virtual void **reInitB**(int which, *realtype* tB0, const *AmiVector* &yyB0, const *AmiVector* &ypB0) const = 0

Reinitializes the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **tB0** – reinitialization timepoint
- **yyB0** – new adjoint state
- **ypB0** – new adjoint derivative state

virtual void **quadReInitB**(int which, const *AmiVector* &yQB0) const = 0

Reinitialize the adjoint states after an event occurrence.

Parameters

- **which** – identifier of the backwards problem
- **yQB0** – new adjoint quadrature state

realtype **gett**() const

current solver timepoint

Returns t

realtype **getCpuTime**() const

Reads out the CPU time needed for forward solve.

Returns cpu_time

realtype **getCpuTimeB**() const

Reads out the CPU time needed for backward solve.

Returns cpu_timeB

int **nx**() const

number of states with which the solver was initialized

Returns x.getLength()

int **nplist**() const

number of parameters with which the solver was initialized

Returns sx.getLength()

int **nquad**() const

number of quadratures with which the solver was initialized

Returns xQB.getLength()

inline bool **computingFSA**() const

check if FSA is being computed

Returns flag

inline bool **computingASA**() const

check if ASA is being computed

Returns flag

void **resetDiagnosis**() const

Resets vectors containing diagnosis information.

void **storeDiagnosis**() const

Stores diagnosis information from solver memory block for forward problem.

void **storeDiagnosisB**(int which) const

Stores diagnosis information from solver memory block for backward problem.

Parameters **which** – identifier of the backwards problem

inline std::vector<int> const &**getNumSteps**() const

Accessor ns.

Returns ns

inline std::vector<int> const &**getNumStepsB**() const

Accessor nsB.

Returns nsB

inline std::vector<int> const &**getNumRhsEvals**() const

Accessor nrhs.

Returns nrhs

inline std::vector<int> const &**getNumRhsEvalsB**() const

Accessor nrhsB.

Returns nrhsB

inline std::vector<int> const &**getNumErrTestFails**() const

Accessor netf.

Returns netf

inline std::vector<int> const &**getNumErrTestFailsB**() const

Accessor netfB.

Returns netfB

inline std::vector<int> const &**getNumNonlinSolvConvFails**() const

Accessor nnlscf.

Returns nnlscf

inline std::vector<int> const &**getNumNonlinSolvConvFailsB**() const

Accessor nnlscfB.

Returns nnlscfB

inline std::vector<int> const &**getLastOrder**() const

Accessor order.

Returns order

inline bool **getNewtonStepSteadyStateCheck**() const

Returns how convergence checks for steadystate computation are performed. If activated, convergence checks are limited to every 25 steps in the simulation solver to limit performance impact.

Returns boolean flag indicating newton step (true) or the right hand side (false)

inline bool **getSensiSteadyStateCheck**() const

Returns how convergence checks for steadystate computation are performed.

Returns boolean flag indicating state and sensitivity equations (true) or only state variables (false).

inline void **setNewtonStepSteadyStateCheck**(bool flag)

Sets how convergence checks for steadystate computation are performed.

Parameters **flag** – boolean flag to pick newton step (true) or the right hand side (false, default)

inline void **setSensiSteadyStateCheck**(bool flag)

Sets for which variables convergence checks for steadystate computation are performed.

Parameters **flag** – boolean flag to pick state and sensitivity equations (true, default) or only state variables (false).

Public Members

Logger ***logger** = nullptr
logger

Protected Functions

virtual void **setStopTime**(*realtype* tstop) const = 0
Sets a timepoint at which the simulation will be stopped.

Parameters **tstop** – timepoint until which simulation should be performed

virtual int **solve**(*realtype* tout, int itask) const = 0
Solves the forward problem until a predefined timepoint.

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP

Returns status flag indicating success of execution

virtual int **solveF**(*realtype* tout, int itask, int *ncheckPtr) const = 0
Solves the forward problem until a predefined timepoint (adjoint only)

Parameters

- **tout** – timepoint until which simulation should be performed
- **itask** – task identifier, can be CV_NORMAL or CV_ONE_STEP
- **ncheckPtr** – pointer to a number that counts the internal checkpoints

Returns status flag indicating success of execution

virtual void **reInitPostProcessF**(*realtype* tnext) const = 0
reInitPostProcessF postprocessing of the solver memory after a discontinuity in the forward problem

Parameters **tnext** – next timepoint (defines integration direction)

virtual void **reInitPostProcessB**(*realtype* tnext) const = 0
reInitPostProcessB postprocessing of the solver memory after a discontinuity in the backward problem

Parameters **tnext** – next timepoint (defines integration direction)

virtual void **getSens**() const = 0
extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

virtual void **getB**(int which) const = 0
extracts the adjoint state at the current timepoint from solver memory and writes it to the xB member variable

Parameters **which** – index of the backwards problem

virtual void **getQuadB**(int which) const = 0
extracts the adjoint quadrature state at the current timepoint from solver memory and writes it to the xQB member variable

Parameters **which** – index of the backwards problem virtual void **getQuad**(*realtype* &t) const = 0

extracts the quadrature at the current timepoint from solver memory and writes it to the xQ member variable

Parameters **t** – timepoint for quadrature extraction

virtual void **init**(*realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const = 0

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **initSteadystate**(*realtype* t0, const *AmiVector* &x0, const *AmiVector* &dx0) const = 0

Initializes the states at the specified initial timepoint.

Parameters

- **t0** – initial timepoint
- **x0** – initial states
- **dx0** – initial derivative states

virtual void **sensInit1**(const *AmiVectorArray* &sx0, const *AmiVectorArray* &sdx0) const = 0

Initializes the forward sensitivities.

Parameters

- **sx0** – initial states sensitivities
- **sdx0** – initial derivative states sensitivities

virtual void **binit**(int which, *realtype* tf, const *AmiVector* &xB0, const *AmiVector* &dxB0) const = 0

Initialize the adjoint states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **tf** – final timepoint
- **xB0** – initial adjoint state
- **dxB0** – initial adjoint derivative state

virtual void **qbinit**(int which, const *AmiVector* &xQB0) const = 0

Initialize the quadrature states at the specified final timepoint.

Parameters

- **which** – identifier of the backwards problem
- **xQB0** – initial adjoint quadrature state

virtual void **rootInit**(int ne) const = 0

Initializes the rootfinding for events.

Parameters **ne** – number of different events void **initializeNonLinearSolverSens**(const *Model* *model) const

Initalize non-linear solver for sensitivities.

Parameters **model** – *Model* instance

virtual void **setDenseJacFn**() const = 0

Set the dense Jacobian function.

virtual void **setSparseJacFn**() const = 0

sets the sparse Jacobian function

virtual void **setBandJacFn**() const = 0

sets the banded Jacobian function

virtual void **setJacTimesVecFn**() const = 0

sets the Jacobian vector multiplication function

virtual void **setDenseJacFnB**(int which) const = 0

sets the dense Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setSparseJacFnB**(int which) const = 0

sets the sparse Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setBandJacFnB**(int which) const = 0

sets the banded Jacobian function

Parameters **which** – identifier of the backwards problem

virtual void **setJacTimesVecFnB**(int which) const = 0

sets the Jacobian vector multiplication function

Parameters **which** – identifier of the backwards problem

virtual void **setSparseJacFn_ss**() const = 0

sets the sparse Jacobian function for backward steady state case

virtual void **allocateSolver**() const = 0

Create specifies solver method and initializes solver memory for the forward problem.

virtual void **setSStolerances**(double rtol, double atol) const = 0

sets scalar relative and absolute tolerances for the forward problem

Parameters

- **rtol** – relative tolerances
- **atol** – absolute tolerances

virtual void **setSensSStolerances**(double rtol, const double *atol) const = 0

activates sets scalar relative and absolute tolerances for the sensitivity variables

Parameters

- **rtol** – relative tolerances
- **atol** – array of absolute tolerances for every sensitivity variable

virtual void **setSensErrCon**(bool error_corr) const = 0

SetSensErrCon specifies whether error control is also enforced for sensitivities for the forward problem

Parameters **error_corr** – activation flag

virtual void **setQuadErrConB**(int which, bool flag) const = 0

Specifies whether error control is also enforced for the backward quadrature problem.

Parameters

- **which** – identifier of the backwards problem
- **flag** – activation flag

virtual void **setQuadErrCon**(bool flag) const = 0

Specifies whether error control is also enforced for the forward quadrature problem.

Parameters **flag** – activation flag

virtual void **setErrHandlerFn**() const = 0

Attaches the error handler function (errMsgIdAndTxt) to the solver.

virtual void **setUserData**() const = 0

Attaches the user data to the forward problem.

virtual void **setUserDataB**(int which) const = 0

attaches the user data to the backward problem

Parameters **which** – identifier of the backwards problem

virtual void **setMaxNumSteps**(long int mxsteps) const = 0

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters **mxsteps** – number of steps

virtual void **setMaxNumStepsB**(int which, long int mxstepsB) const = 0

specifies the maximum number of steps for the forward problem

Note: in contrast to the SUNDIALS method, this sets the overall maximum, not the maximum between output times.

Parameters

- **which** – identifier of the backwards problem
- **mxstepsB** – number of steps

virtual void **setStabLimDet**(int stldet) const = 0

activates stability limit detection for the forward problem

Parameters **stldet** – flag for stability limit detection (TRUE or FALSE) virtual void **setStabLimDetB**(int which, int stldet) const = 0

activates stability limit detection for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **stldet** – flag for stability limit detection (TRUE or FALSE)

virtual void **setId**(const *Model* *model) const = 0

specify algebraic/differential components (DAE only)

Parameters **model** – model specification

virtual void **setSuppressAlg**(bool flag) const = 0

deactivates error control for algebraic components (DAE only)

Parameters **flag** – deactivation flag

virtual void **setSensParams**(const *realtype* *p, const *realtype* *pbar, const int *plist) const = 0

specifies the scaling and indexes for sensitivity computation

Parameters

- **p** – parameters
- **pbar** – parameter scaling constants
- **plist** – parameter index list

virtual void **getDky**(*realtype* t, int k) const = 0

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getDkyB**(*realtype* t, int k, int which) const = 0

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order
- **which** – index of backward problem

virtual void **getSensDky**(*realtype* t, int k) const = 0

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **getQuadDkyB**(*realtype* t, int k, int which) const = 0

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

- **which** – index of backward problem

virtual void **getQuadDky**(*realtype* t, int k) const = 0

interpolates the (derivative of the) solution at the requested timepoint

Parameters

- **t** – timepoint
- **k** – derivative order

virtual void **adjInit**() const = 0

initializes the adjoint problem

virtual void **quadInit**(const *AmiVector* &xQ0) const = 0

initializes the quadratures

Parameters **xQ0** – vector with initial values for xQ

virtual void **allocateSolverB**(int *which) const = 0

Specifies solver method and initializes solver memory for the backward problem.

Parameters **which** – identifier of the backwards problem

virtual void **setSStolerancesB**(int which, *realtype* relTolB, *realtype* absTolB) const = 0

sets relative and absolute tolerances for the backward problem

Parameters

- **which** – identifier of the backwards problem
- **relTolB** – relative tolerances
- **absTolB** – absolute tolerances

virtual void **quadSStolerancesB**(int which, *realtype* reltolQB, *realtype* abstolQB) const = 0

sets relative and absolute tolerances for the quadrature backward problem

Parameters

- **which** – identifier of the backwards problem
- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual void **quadSStolerances**(*realtype* reltolQB, *realtype* abstolQB) const = 0

sets relative and absolute tolerances for the quadrature problem

Parameters

- **reltolQB** – relative tolerances
- **abstolQB** – absolute tolerances

virtual void **getNumSteps**(const void *ami_mem, long int *numsteps) const = 0

reports the number of solver steps

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numsteps** – output array

virtual void **getNumRhsEvals**(const void *ami_mem, long int *numrhsevals) const = 0

reports the number of right hand evaluations

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numrhsevals** – output array

virtual void **getNumErrTestFails**(const void *ami_mem, long int *numerrtestfails) const = 0

reports the number of local error test failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numerrtestfails** – output array

virtual void **getNumNonlinSolvConvFails**(const void *ami_mem, long int *numnonlinsolvconvfails) const = 0

reports the number of nonlinear convergence failures

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **numnonlinsolvconvfails** – output array

virtual void **getLastOrder**(const void *ami_mem, int *order) const = 0

Reports the order of the integration method during the last internal step.

Parameters

- **ami_mem** – pointer to the solver memory instance (can be from forward or backward problem)
- **order** – output array

void **initializeLinearSolver**(const *Model* *model) const

Initializes and sets the linear solver for the forward problem.

Parameters **model** – pointer to the model object

void **initializeNonLinearSolver**() const

Sets the non-linear solver.

virtual void **setLinearSolver**() const = 0

Sets the linear solver for the forward problem.

virtual void **setLinearSolverB**(int which) const = 0

Sets the linear solver for the backward problem.

Parameters **which** – index of the backward problem

virtual void **setNonLinearSolver**() const = 0

Set the non-linear solver for the forward problem.

virtual void **setNonLinearSolverB**(int which) const = 0

Set the non-linear solver for the backward problem.

Parameters **which** – index of the backward problem

virtual void **setNonLinearSolverSens**() const = 0

Set the non-linear solver for sensitivities.

void **initializeLinearSolverB**(const *Model* *model, int which) const

Initializes the linear solver for the backward problem.

Parameters

- **model** – pointer to the model object
- **which** – index of the backward problem

void **initializeNonLinearSolverB**(int which) const

Initializes the non-linear solver for the backward problem.

Parameters **which** – index of the backward problem

virtual const *Model* ***getModel**() const = 0

Accessor function to the model stored in the user data

Returns user data model

bool **getInitDone**() const

checks whether memory for the forward problem has been allocated

Returns proxy for solverMemory->(cv|ida)_MallocDone

bool **getSensInitDone**() const

checks whether memory for forward sensitivities has been allocated

Returns proxy for solverMemory->(cv|ida)_SensMallocDone

bool **getAdjInitDone**() const

checks whether memory for forward interpolation has been allocated

Returns proxy for solverMemory->(cv|ida)_adjMallocDone

bool **getInitDoneB**(int which) const

checks whether memory for the backward problem has been allocated

Parameters **which** – adjoint problem index

Returns proxy for solverMemoryB->(cv|ida)_MallocDone

bool **getQuadInitDoneB**(int which) const

checks whether memory for backward quadratures has been allocated

Parameters **which** – adjoint problem index

Returns proxy for solverMemoryB->(cv|ida)_QuadMallocDone

bool **getQuadInitDone**() const

checks whether memory for quadratures has been allocated

Returns proxy for solverMemory->(cv|ida)_QuadMallocDone

virtual void **diag**() const = 0

attaches a diagonal linear solver to the forward problem virtual void **diagB**(int which) const = 0

attaches a diagonal linear solver to the backward problem

Parameters `which` – identifier of the backwards problem

void **resetMutableMemory**(int nx, int nplist, int nquad) const

resets solverMemory and solverMemoryB

Parameters

- `nx` – new number of state variables
- `nplist` – new number of sensitivity parameters
- `nquad` – new number of quadratures (only differs from nplist for higher order sensitivity computation)

virtual void ***getAdjBmem**(void *ami_mem, int which) const = 0

Retrieves the solver memory instance for the backward problem.

Parameters

- `which` – identifier of the backwards problem
- `ami_mem` – pointer to the forward solver memory instance

Returns A (void *) pointer to the CVODES memory allocated for the backward problem.

void **applyTolerances**() const

updates solver tolerances according to the currently specified member variables

void **applyTolerancesFSA**() const

updates FSA solver tolerances according to the currently specified member variables

void **applyTolerancesASA**(int which) const

updates ASA solver tolerances according to the currently specified member variables

Parameters `which` – identifier of the backwards problem

void **applyQuadTolerancesASA**(int which) const

updates ASA quadrature solver tolerances according to the currently specified member variables

Parameters `which` – identifier of the backwards problem

void **applyQuadTolerances**() const

updates quadrature solver tolerances according to the currently specified member variables

void **applySensitivityTolerances**() const

updates all sensitivity solver tolerances according to the currently specified member variables

void **setInitDone**() const

sets that memory for the forward problem has been allocated

void **setSensInitDone**() const

sets that memory for forward sensitivities has been allocated

void **setSensInitOff**() const

sets that memory for forward sensitivities has not been allocated

void **setAdjInitDone**() const

sets that memory for forward interpolation has been allocated void **setInitDoneB**(int which) const

sets that memory for the backward problem has been allocated

Parameters **which** – adjoint problem index

void **setQuadInitDoneB**(int which) const

sets that memory for backward quadratures has been allocated

Parameters **which** – adjoint problem index

void **setQuadInitDone**() const

sets that memory for quadratures has been allocated

void **checkSensitivityMethod**(const *SensitivityMethod* sensi_meth, bool preequilibration) const

Sets sensitivity method (for simulation or preequilibration)

Parameters

- **sensi_meth** – new value for sensi_meth[_preq]
- **preequilibration** – flag indicating preequilibration or simulation

Protected Attributes

mutable std::unique_ptr<void, std::function<void(void*)>> **solver_memory_**

pointer to solver memory block

mutable std::vector<std::unique_ptr<void, std::function<void(void*)>>> **solver_memory_B_**

pointer to solver memory block

mutable *user_data_type* **user_data**

Sundials user_data

InternalSensitivityMethod **ism_** = {*InternalSensitivityMethod*::*simultaneous*}

internal sensitivity method flag used to select the sensitivity solution method. Only applies for Forward Sensitivities.

LinearMultistepMethod **lmm_** = {*LinearMultistepMethod*::*BDF*}

specifies the linear multistep method.

NonlinearSolverIteration **iter_** = {*NonlinearSolverIteration*::*newton*}

specifies the type of nonlinear solver iteration

InterpolationType **interp_type_** = {*InterpolationType*::*hermite*}

interpolation type for the forward problem solution which is then used for the backwards problem.

long int **maxsteps_** = {10000}

maximum number of allowed integration steps

std::chrono::duration<double, std::ratio<1>> **maxtime_** = {std::chrono::duration<double>::max()}

Maximum wall-time for integration in seconds mutable std::chrono::time_point<std::chrono::system_clock> **starttime_**

Time at which solver timer was started

mutable std::unique_ptr<*SUNLinSolWrapper*> **linear_solver_**

linear solver for the forward problem

mutable std::unique_ptr<*SUNLinSolWrapper*> **linear_solver_B_**

linear solver for the backward problem

mutable std::unique_ptr<*SUNNonLinSolWrapper*> **non_linear_solver_**

non-linear solver for the forward problem

mutable std::unique_ptr<*SUNNonLinSolWrapper*> **non_linear_solver_B_**

non-linear solver for the backward problem

mutable std::unique_ptr<*SUNNonLinSolWrapper*> **non_linear_solver_sens_**

non-linear solver for the sensitivities

mutable bool **solver_was_called_F_** = {false}

flag indicating whether the forward solver has been called

mutable bool **solver_was_called_B_** = {false}

flag indicating whether the backward solver has been called

mutable *AmiVector* **x_** = {0}

state (dimension: nx_solver)

mutable *AmiVector* **dky_** = {0}

state interface variable (dimension: nx_solver)

mutable *AmiVector* **dx_** = {0}

state derivative dummy (dimension: nx_solver)

mutable *AmiVectorArray* **sx_** = {0, 0}

state sensitivities interface variable (dimension: nx_solver x nplist)

mutable *AmiVectorArray* **sdx_** = {0, 0}

state derivative sensitivities dummy (dimension: nx_solver x nplist)

mutable *AmiVector* **xB_** = {0}

adjoint state interface variable (dimension: nx_solver)

mutable *AmiVector* **dxB_** = {0}

adjoint derivative dummy variable (dimension: nx_solver) mutable *AmiVector* **xQB_** = {0}

adjoint quadrature interface variable (dimension: nJ x nplist)

mutable *AmiVector* **xQ_** = {0}

forward quadrature interface variable (dimension: nx_solver)

mutable *realtype* **t_** = {std::nan("")}

integration time of the forward problem

mutable bool **force_reinit_postprocess_F_** = {false}

flag to force reInitPostProcessF before next call to solve

mutable bool **force_reinit_postprocess_B_** = {false}

flag to force reInitPostProcessB before next call to solveB

mutable bool **sens_initialized_** = {false}

flag indicating whether sensInit1 was called

Friends

template<class **Archive**>
friend void **serialize**(*Archive* &ar, *Solver* &s, unsigned int version)

Serialize *Solver* (see boost::serialization::serialize)

Parameters

- **ar** – Archive to serialize to
- **s** – Data to serialize
- **version** – Version number

friend bool **operator==**(const *Solver* &a, const *Solver* &b)

Check equality of data members excluding solver memory.

Parameters

- **a** –
- **b** –

Returns

Class SteadystateProblem

- Defined in file_include_amici_steadystateproblem.h

Class Documentation

class **SteadystateProblem**

The *SteadystateProblem* class solves a steady-state problem using Newton's method and falls back to integration on failure.

Public Functions

explicit **SteadystateProblem**(const *Solver* &solver, const *Model* &model)

constructor

Parameters

- **solver** – *Solver* instance
- **model** – *Model* instance

void **workSteadyStateProblem**(const *Solver* &solver, *Model* &model, int it)

Handles steady state computation in the forward case: tries to determine the steady state of the ODE system and computes steady state sensitivities if requested.

Parameters

- **solver** – pointer to the solver object
- **model** – pointer to the model object
- **it** – integer with the index of the current time step

void **workSteadyStateBackwardProblem**(const *Solver* &solver, *Model* &model, const *BackwardProblem* *bwd)

Integrates over the adjoint state backward in time by solving a linear system of equations, which gives the analytical solution. Computes the gradient via adjoint steady state sensitivities

Parameters

- **solver** – pointer to the solver object
- **model** – pointer to the model object
- **bwd** – backward problem

inline const *SimulationState* &**getFinalSimulationState**() const

Returns the stored *SimulationState*.

Returns stored *SimulationState*

inline const *AmiVector* &**getEquilibrationQuadratures**() const

Returns the quadratures from pre- or postequilibration.

Returns xQB Vector with quadratures

inline const *AmiVector* &**getState**() const

Returns state at steadystate.

Returns x

inline const *AmiVectorArray* &**getStateSensitivity**() const

Returns state sensitivity at steadystate.

Returns sx

inline std::vector<*realtype*> const &**getDJydx**() const

Accessor for dJydx.

Returns dJydx

inline double **getCPUTime**() const

Accessor for run_time of the forward problem.

Returns run_time

inline double **getCPUTimeB**() const

Accessor for run_time of the backward problem.

Returns run_time

inline std::vector<*SteadyStateStatus*> const &**getSteadyStateStatus**() const

Accessor for steady_state_status.

Returns steady_state_status

inline *realtype* **getSteadyStateTime**() const

Get model time at which steadystate was found through simulation.

Returns t

inline *realtype* **getResidualNorm**() const

Accessor for wrms.

Returns wrms

inline const std::vector<int> &**getNumSteps**() const

Accessor for numsteps.

Returns numsteps

inline int **getNumStepsB**() const

Accessor for numstepsB.

Returns numstepsB

void **getAdjointUpdates**(*Model* &model, const *ExpData* &edata)

computes adjoint updates dJydx according to provided model and expdata

Parameters

- **model** – *Model* instance
- **edata** – experimental data

inline *AmiVector* const &**getAdjointState**() const

Return the adjoint state.

Returns xB adjoint state

inline *AmiVector* const &**getAdjointQuadrature**() const

Accessor for xQB.

Returns xQB

inline bool **hasQuadrature**() const

Accessor for hasQuadrature_.

Returns hasQuadrature_

bool **checkSteadyStateSuccess**() const

computes adjoint updates dJydx according to provided model and expdata

Returns convergence of steady state solver

Class SUNLinSolBand

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolBand** : public amici::*SUNLinSolWrapper*

SUNDIALS band direct solver.

Public Functions

SUNLinSolBand(N_Vector x, SUNMatrix A)

Create solver using existing matrix A without taking ownership of A.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **A** – square matrix

SUNLinSolBand(*AmiVector* const &x, int ubw, int lbw)

Create new band solver and matrix A.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **ubw** – upper bandwidth of band matrix A
- **lbw** – lower bandwidth of band matrix A

virtual SUNMatrix **getMatrix**() const override

Get the matrix A (matrix solvers only).

Returns A

Class SUNLinSolDense

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolDense** : public amici::*SUNLinSolWrapper*

SUNDIALS dense direct solver.

Public Functions

explicit **SUNLinSolDense**(*AmiVector* const &x)

Create dense solver.

Parameters **x** – A template for cloning vectors needed within the solver.

virtual SUNMatrix **getMatrix**() const override

Get the matrix A (matrix solvers only).

Returns A

Class SUNLinSolKLU

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolKLU** : public amici::*SUNLinSolWrapper*

SUNDIALS KLU sparse direct solver.

Public Types

enum class **StateOrdering**

KLU state reordering (different from SuperLUMT ordering!)

Values:

enumerator **AMD**

enumerator **COLAMD**

enumerator **natural**

Public Functions

SUNLinSolKLU(N_Vector x, SUNMatrix A)

Create KLU solver with given matrix.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **A** – sparse matrix

SUNLinSolKLU(*AmiVector* const &x, int nnz, int sparsetype, *StateOrdering* ordering)

Create KLU solver and matrix to operate on.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **nnz** – Number of non-zeros in matrix A
- **sparsetype** – Sparse matrix type (CSC_MAT, CSR_MAT)
- **ordering** –

virtual SUNMatrix **getMatrix**() const override

Get the matrix A (matrix solvers only).

Returns A

void **reInit**(int nnz, int reinit_type)

Reinitializes memory and flags for a new factorization (symbolic and numeric) to be conducted at the next solver setup call.

For more details see sunlinsol/sunlinsol_klu.h

Parameters

- **nnz** – Number of non-zeros
- **reinit_type** – SUNKLU_REINIT_FULL or SUNKLU_REINIT_PARTIAL

void **setOrdering**(*StateOrdering* ordering)

Sets the ordering used by KLU for reducing fill in the linear solve.

Parameters **ordering** –

Class SUNLinSolPCG

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolPCG** : public amici::*SUNLinSolWrapper*

SUNDIALS scaled preconditioned CG (Conjugate Gradient method) (PCG) solver.

Public Functions

SUNLinSolPCG(N_Vector y, int pretype, int maxl)

Create PCG solver.

Parameters

- **y** –
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

int **setATimes**(void *A_data, ATimesFn ATimes)

Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).

Parameters

- **A_data** –
- **ATimes** –

Returns

int **setPreconditioner**(void *P_data, PSetupFn Pset, PSolveFn Psol)

Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials_linearsolver.h).

Parameters

- **P_data** –
- **Pset** –
- **Psol** –

Returns

int **setScalingVectors**(N_Vector s, N_Vector nul)

Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).

Parameters

- **s** –
- **nul** –

Returns

int **getNumIters**() const

Returns the number of linear iterations performed in the last 'Solve' call.

Returns Number of iterations

realtype **getResNorm**() const

Returns the final residual norm from the last 'Solve' call.

Returns residual norm

N_Vector **getResid**() const

Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Returns

Class SUNLinSolSPBCGS

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolSPBCGS** : public amici::*SUNLinSolWrapper*

SUNDIALS scaled preconditioned Bi-CGStab (Bi-Conjugate Gradient Stable method) (SPBCGS) solver.

Public Functions

explicit **SUNLinSolSPBCGS**(N_Vector x, int pretype = PREC_NONE, int maxl = SUNSPBCGS_MAXL_DEFAULT)

SUNLinSolSPBCGS.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

explicit **SUNLinSolSPBCGS**(*AmiVector* const &x, int pretype = PREC_NONE, int maxl = SUNSPBCGS_MAXL_DEFAULT)

SUNLinSolSPBCGS.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

int **setATimes**(void *A_data, ATimesFn ATimes)

Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).

Parameters

- **A_data** –
- **ATimes** –

Returns

int **setPreconditioner**(void *P_data, PSetupFn Pset, PSolveFn Psol)

Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials_linearsolver.h).

Parameters

- **P_data** –
- **Pset** –
- **Psol** –

Returns

int **setScalingVectors**(N_Vector s, N_Vector nul)

Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).

Parameters

- **s** –
- **nul** –

Returns

int **getNumIters**() const

Returns the number of linear iterations performed in the last 'Solve' call.

Returns Number of iterations

realtype **getResNorm**() const

Returns the final residual norm from the last 'Solve' call.

Returns residual norm

N_Vector **getResid**() const

Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Returns

Class SUNLinSolSPFGMR

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolSPFGMR** : public amici::*SUNLinSolWrapper*

SUNDIALS scaled preconditioned FGMRES (Flexible Generalized Minimal Residual method) (SPFGMR) solver.

Public Functions

SUNLinSolSPFGMR(*AmiVector* const &x, int pretype, int maxl)

SUNLinSolSPFGMR.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

int **setATimes**(void *A_data, ATimesFn ATimes)

Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).

Parameters

- **A_data** –
- **ATimes** –

Returns

int **setPreconditioner**(void *P_data, PSetupFn Pset, PSolveFn Psol)

Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials_linearsolver.h).

Parameters

- **P_data** –
- **Pset** –
- **Psol** –

Returns

int **setScalingVectors**(N_Vector s, N_Vector nul)

Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).

Parameters

- **s** –
- **nul** –

Returns

int **getNumIters**() const

Returns the number of linear iterations performed in the last 'Solve' call.

Returns Number of iterations

realtype **getResNorm**() const

Returns the final residual norm from the last 'Solve' call.

Returns residual norm

N_Vector **getResid**() const

Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Returns

Class SUNLinSolSPGMR

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolSPGMR** : public amici::*SUNLinSolWrapper*

SUNDIALS scaled preconditioned GMRES (Generalized Minimal Residual method) solver (SPGMR).

Public Functions

explicit **SUNLinSolSPGMR**(*AmiVector* const &x, int pretype = PREC_NONE, int maxl = SUNSPGMR_MAXL_DEFAULT)

Create SPGMR solver.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

int **setATimes**(void *A_data, ATimesFn ATimes)

Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).

Parameters

- **A_data** –
- **ATimes** –

Returns

int **setPreconditioner**(void *P_data, PSetupFn Pset, PSolveFn Psol)

Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials_linearsolver.h).

Parameters

- **P_data** –
- **Pset** –
- **Psol** –

Returns

int **setScalingVectors**(N_Vector s, N_Vector nul)

Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).

Parameters

- **s** –
- **nul** –

Returns

int **getNumIters**() const

Returns the number of linear iterations performed in the last 'Solve' call.

Returns Number of iterations

realtype **getResNorm**() const

Returns the final residual norm from the last 'Solve' call.

Returns residual norm

N_Vector **getResid**() const

Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Returns

Class SUNLinSolSPTFQMR

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- public amici::SUNLinSolWrapper (*Class SUNLinSolWrapper*)

Class Documentation

class **SUNLinSolSPTFQMR** : public amici::*SUNLinSolWrapper*

SUNDIALS scaled preconditioned TFQMR (Transpose-Free Quasi-Minimal Residual method) (SPTFQMR) solver.

Public Functions

explicit **SUNLinSolSPTFQMR**(N_Vector x, int pretype = PREC_NONE, int maxl = SUNSPTFQMR_MAXL_DEFAULT)

Create SPTFQMR solver.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

explicit **SUNLinSolSPTFQMR**(*AmiVector* const &x, int pretype = PREC_NONE, int maxl = SUNSPTFQMR_MAXL_DEFAULT)

Create SPTFQMR solver.

Parameters

- **x** – A template for cloning vectors needed within the solver.
- **pretype** – Preconditioner type (PREC_NONE, PREC_LEFT, PREC_RIGHT, PREC_BOTH)
- **maxl** – Maximum number of solver iterations

int **setATimes**(void *A_data, ATimesFn ATimes)

Sets the function pointer for ATimes (see sundials/sundials_linearsolver.h).

Parameters

- **A_data** –
- **ATimes** –

Returns

int **setPreconditioner**(void *P_data, PSetupFn Pset, PSolveFn Psol)

Sets function pointers for PSetup and PSolve routines inside of iterative linear solver objects (see sundials/sundials_linearsolver.h).

Parameters

- **P_data** –

- **Pset** –
- **Psol** –

Returns

int **setScalingVectors**(N_Vector s, N_Vector nul)

Sets pointers to left/right scaling vectors for the linear system solve (see sundials/sundials_linearsolver.h).

Parameters

- **s** –
- **nul** –

Returns

int **getNumIters**() const

Returns the number of linear iterations performed in the last 'Solve' call.

Returns Number of iterations

realtype **getResNorm**() const

Returns the final residual norm from the last 'Solve' call.

Returns residual norm

N_Vector **getResid**() const

Get preconditioned initial residual (see sundials/sundials_linearsolver.h).

Returns

Class SUNLinSolWrapper

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Derived Types

- `public amici::SUNLinSolBand` (*Class SUNLinSolBand*)
- `public amici::SUNLinSolDense` (*Class SUNLinSolDense*)
- `public amici::SUNLinSolKLU` (*Class SUNLinSolKLU*)
- `public amici::SUNLinSolPCG` (*Class SUNLinSolPCG*)
- `public amici::SUNLinSolSPBCGS` (*Class SUNLinSolSPBCGS*)
- `public amici::SUNLinSolSPFGMR` (*Class SUNLinSolSPFGMR*)
- `public amici::SUNLinSolSPGMR` (*Class SUNLinSolSPGMR*)
- `public amici::SUNLinSolSPTFQMR` (*Class SUNLinSolSPTFQMR*)

Class Documentation

class **SUNLinSolWrapper**

A RAII wrapper for SUNLinearSolver structs.

For details on member functions see documentation in sunlinsol/sundials_linearsolver.h.

Subclassed by *amici::SUNLinSolBand*, *amici::SUNLinSolDense*, *amici::SUNLinSolKLU*, *amici::SUNLinSolPCG*, *amici::SUNLinSolSPBCGS*, *amici::SUNLinSolSPFGMR*, *amici::SUNLinSolSPGMR*, *amici::SUNLinSolSPTFQMR*

Public Functions

SUNLinSolWrapper() = default

explicit **SUNLinSolWrapper**(SUNLinearSolver linsol)

Wrap existing SUNLinearSolver.

Parameters **linsol** –

virtual **~SUNLinSolWrapper**()

SUNLinSolWrapper(const *SUNLinSolWrapper* &other) = delete

Copy constructor.

Parameters **other** –

SUNLinSolWrapper(*SUNLinSolWrapper* &&other) noexcept

Move constructor.

Parameters **other** –

SUNLinSolWrapper &**operator=**(const *SUNLinSolWrapper* &other) = delete

Copy assignment.

Parameters **other** –

Returns

SUNLinSolWrapper &**operator=**(*SUNLinSolWrapper* &&other) noexcept

Move assignment.

Parameters **other** –

Returns

SUNLinearSolver **get**() const

Returns the wrapped SUNLinSol.

Returns SUNLinearSolver

SUNLinearSolver_Type **getType**() const

Returns an identifier for the linear solver type.

Returns

void **setup**(SUNMatrix A) const

Performs any linear solver setup needed, based on an updated system matrix A.

Parameters **A** –

void **setup**(const *SUNMatrixWrapper* &A) const

Performs any linear solver setup needed, based on an updated system matrix A.

Parameters **A** –

int **Solve**(SUNMatrix A, N_Vector x, N_Vector b, *realtype* tol) const

Solves a linear system A*x = b.

Parameters

- **A** –
- **x** – A template for cloning vectors needed within the solver.
- **b** –
- **tol** – Tolerance (weighted 2-norm), iterative solvers only

Returns error flag

long int **getLastFlag**() const

Returns the last error flag encountered within the linear solver.

Returns error flag

int **space**(long int *lenrwLS, long int *leniwLS) const

Returns the integer and real workspace sizes for the linear solver.

Parameters

- **lenrwLS** – output argument for size of real workspace
- **leniwLS** – output argument for size of integer workspace

Returns workspace size

virtual SUNMatrix **getMatrix**() const

Get the matrix A (matrix solvers only).

Returns A

Protected Functions

int **initialize**()

Performs linear solver initialization (assumes that all solver-specific options have been set).

Returns error code

Protected Attributes

SUNLinearSolver **solver_** = {nullptr}

Wrapped solver

Class SUNMatrixWrapper

- Defined in file_include_amici_sundials_matrix_wrapper.h

Class Documentation

class **SUNMatrixWrapper**

A RAII wrapper for SUNMatrix structs.

This can create dense, sparse, or banded matrices using the respective constructor.

Public Functions

SUNMatrixWrapper() = default

SUNMatrixWrapper(sunindextype M, sunindextype N, sunindextype NNZ, int sparsetype)

Create sparse matrix. See SUNSparseMatrix in sunmatrix_sparse.h.

Parameters

- **M** – Number of rows
- **N** – Number of columns
- **NNZ** – Number of nonzeros
- **sparsetype** – Sparse type

SUNMatrixWrapper(sunindextype M, sunindextype N)

Create dense matrix. See SUNDenseMatrix in sunmatrix_dense.h.

Parameters

- **M** – Number of rows
- **N** – Number of columns

SUNMatrixWrapper(sunindextype M, sunindextype ubw, sunindextype lbw)

Create banded matrix. See SUNBandMatrix in sunmatrix_band.h.

Parameters

- **M** – Number of rows and columns
- **ubw** – Upper bandwidth
- **lbw** – Lower bandwidth

SUNMatrixWrapper(const *SUNMatrixWrapper* &A, *realtype* droptol, int sparsetype)

Create sparse matrix from dense or banded matrix. See SUNSparseFromDenseMatrix and SUNSparseFromBandMatrix in sunmatrix_sparse.h.

Parameters

- **A** – Wrapper for dense matrix
- **droptol** – tolerance for dropping entries
- **sparsetype** – Sparse type

explicit **`SUNMatrixWrapper`**(SUNMatrix mat)

Wrap existing SUNMatrix.

Parameters **`mat`** –

`~SUNMatrixWrapper`()

`SUNMatrixWrapper`(const *SUNMatrixWrapper* &other)

Copy constructor.

Parameters **`other`** –

`SUNMatrixWrapper`(*SUNMatrixWrapper* &&other)

Move constructor.

Parameters **`other`** –

SUNMatrixWrapper &**`operator=`**(const *SUNMatrixWrapper* &other)

Copy assignment.

Parameters **`other`** –

Returns

SUNMatrixWrapper &**`operator=`**(*SUNMatrixWrapper* &&other)

Move assignment.

Parameters **`other`** –

Returns

void **`reallocate`**(sunindextype nnz)

Reallocate space for sparse matrix according to specified nnz.

Parameters **`nnz`** – new number of nonzero entries

void **`realloc`**()

Reallocate space for sparse matrix to used space according to last entry in indexptrs.

SUNMatrix **`get`**() const

Get the wrapped SUNMatrix.

Note: Even though the returned matrix_ pointer is const qualified, matrix_->content will not be const. This is a shortcoming in the underlying C library, which we cannot address and it is not intended that any of those values are modified externally. If matrix_->content is manipulated, cpp:meth:SUNMatrixWrapper:`refresh` needs to be called.

Returns raw SunMatrix object

inline sunindextype **`rows`**() const

Get the number of rows.

Returns number of rows

inline sunindextype **`columns`**() const

Get the number of columns.

Returns number of columns

sunindextype **num_nonzeros**() const

Get the number of specified non-zero elements (sparse matrices only)

Note: value will be 0 before indexptrs are set.

Returns number of nonzero entries

sunindextype **num_indexptrs**() const

Get the number of indexptrs that can be specified (sparse matrices only)

Returns number of indexptrs

sunindextype **capacity**() const

Get the number of allocated data elements.

Returns number of allocated entries

realtype ***data**()

Get raw data of a sparse matrix.

Returns pointer to first data entry

const *realtype* ***data**() const

Get const raw data of a sparse matrix.

Returns pointer to first data entry

inline *realtype* **get_data**(sunindextype idx) const

Get data of a sparse matrix.

Parameters **idx** – data index

Returns idx-th data entry

inline *realtype* **get_data**(sunindextype irow, sunindextype icol) const

Get data entry for a dense matrix.

Parameters

- **irow** – row
- **icol** – col

Returns A(irow,icol)

inline void **set_data**(sunindextype idx, *realtype* data)

Set data entry for a sparse matrix.

Parameters

- **idx** – data index
- **data** – data for idx-th entry

inline void **set_data**(sunindextype irow, sunindextype icol, *realtype* data)

Set data entry for a dense matrix.

Parameters

- **irow** – row
- **icol** – col

- **data** – data for idx-th entry

inline sunindextype **get_indexval**(sunindextype idx) const

Get the index value of a sparse matrix.

Parameters **idx** – data index

Returns row (CSC) or column (CSR) for idx-th data entry

inline void **set_indexval**(sunindextype idx, sunindextype val)

Set the index value of a sparse matrix.

Parameters

- **idx** – data index
- **val** – row (CSC) or column (CSR) for idx-th data entry

inline void **set_indexvals**(const gsl::span<const sunindextype> vals)

Set the index values of a sparse matrix.

Parameters **vals** – rows (CSC) or columns (CSR) for data entries

inline sunindextype **get_indexptr**(sunindextype ptr_idx) const

Get the index pointer of a sparse matrix.

Parameters **ptr_idx** – pointer index

Returns index where the ptr_idx-th column (CSC) or row (CSR) starts

inline void **set_indexptr**(sunindextype ptr_idx, sunindextype ptr)

Set the index pointer of a sparse matrix.

Parameters

- **ptr_idx** – pointer index
- **ptr** – data-index where the ptr_idx-th column (CSC) or row (CSR) starts

inline void **set_indexptrs**(const gsl::span<const sunindextype> ptrs)

Set the index pointers of a sparse matrix.

Parameters **ptrs** – starting data-indices where the columns (CSC) or rows (CSR) start

int **sparsetype**() const

Get the type of sparse matrix.

Returns matrix type

void **scale**(*realtype* a)

multiply with a scalar (in-place)

Parameters **a** – scalar value to multiply matrix

void **multiply**(N_Vector c, *const_N_Vector* b, *realtype* alpha = 1.0) const

N_Vector interface for multiply.

Parameters

- **c** – output vector, may already contain values
- **b** – multiplication vector
- **alpha** – scalar coefficient for matrix

inline void **multiply**(*AmiVector* &c, *AmiVector* const &b, *realtype* alpha = 1.0) const

AmiVector interface for multiply.

Parameters

- **c** – output vector, may already contain values
- **b** – multiplication vector
- **alpha** – scalar coefficient for matrix

void **multiply**(gsl::span<*realtype*> c, gsl::span<const *realtype*> b, const *realtype* alpha = 1.0) const

Perform matrix vector multiplication c += alpha * A*b.

Parameters

- **c** – output vector, may already contain values
- **b** – multiplication vector
- **alpha** – scalar coefficient

void **multiply**(N_Vector c, *const_N_Vector* b, gsl::span<const int> cols, bool transpose) const

Perform reordered matrix vector multiplication c += A[:,cols]*b.

Parameters

- **c** – output vector, may already contain values
- **b** – multiplication vector
- **cols** – int vector for column reordering
- **transpose** – bool transpose A before multiplication

void **multiply**(gsl::span<*realtype*> c, gsl::span<const *realtype*> b, gsl::span<const int> cols, bool transpose) const

Perform reordered matrix vector multiplication c += A[:,cols]*b.

Parameters

- **c** – output vector, may already contain values
- **b** – multiplication vector
- **cols** – int vector for column reordering
- **transpose** – bool transpose A before multiplication

void **sparse_multiply**(*SUNMatrixWrapper* &C, const *SUNMatrixWrapper* &B) const

Perform matrix matrix multiplication C = A * B for sparse A, B, C.

Note: will overwrite existing data, indexptrs, indexvals for C, but will use preallocated space for these vars

Parameters

- **C** – output matrix,
- **B** – multiplication matrix

void **sparse_add**(const *SUNMatrixWrapper* &A, *realtype* alpha, const *SUNMatrixWrapper* &B, *realtype* beta)

Perform sparse matrix matrix addition C = alpha * A + beta * B.

Note: will overwrite existing data, indexptrs, indexvals for C, but will use preallocated space for these vars

Parameters

- **A** – addition matrix
- **alpha** – scalar A
- **B** – addition matrix
- **beta** – scalar B

void **sparse_sum**(const std::vector<*SUNMatrixWrapper*> &mats)

Perform matrix-matrix addition A = sum(mats(0)…mats(len(mats)))

Note: will overwrite existing data, indexptrs, indexvals for A, but will use preallocated space for these vars

Parameters **mats** – vector of sparse matrices

sunindextype **scatter**(const sunindextype k, const *realtype* beta, sunindextype *w, gsl::span<*realtype*> x, const sunindextype mark, *SUNMatrixWrapper* *C, sunindextype nnz) const

Compute x = x + beta * A(:,k), where x is a dense vector and A(:,k) is sparse, and update the sparsity pattern for C(:,j) if applicable.

This function currently has two purposes:

- perform parts of sparse matrix-matrix multiplication C(:,j)=A(:,k)*B(k,j) enabled by passing beta=B(k,j), x=C(:,j), C=C, w=sparsity of C(:,j) from B(k,0…j-1), nnz=nnz(C(:,0…j-1)
- add the k-th column of the sparse matrix A multiplied by beta to the dense vector x. enabled by passing beta=*, x=x, C=nullptr, w=nullptr, nnz=*

Parameters

- **k** – column index
- **beta** – scaling factor
- **w** – index workspace, (w[i]<mark) indicates non-zeroness of C(i,j) (dimension: m), if this is a nullptr, sparsity pattern of C will not be updated (if applicable).
- **x** – dense output vector (dimension: m)
- **mark** – marker for w to indicate nonzero pattern
- **C** – sparse output matrix, if this is a nullptr, sparsity pattern of C will not be updated
- **nnz** – number of nonzeros that were already written to C

Returns updated number of nonzeros in C void **transpose**(*SUNMatrixWrapper* &C, const *realtype* alpha, sunindextype blocksize) const

Compute transpose A' of sparse matrix A and writes it to the matrix C = alpha * A'.

Parameters

- **C** – output matrix (sparse or dense)
- **alpha** – scalar multiplier
- **blocksize** – blocksize for transposition. For full matrix transpose set to ncols/nrows

void **to_dense**(*SUNMatrixWrapper* &D) const

Writes a sparse matrix A to a dense matrix D.

Parameters **D** – dense output matrix

void **to_diag**(N_Vector v) const

Writes the diagonal of sparse matrix A to a dense vector v.

Parameters **v** – dense outut vector

void **zero**()

Set to 0.0, for sparse matrices also resets indexptr/indexvals.

inline SUNMatrix_ID **matrix_id**() const

Get matrix id.

Returns SUNMatrix_ID

void **refresh**()

Update internal cache, needs to be called after external manipulation of matrix_->content.

Class SUNNonLinSolFixedPoint

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNNonLinSolWrapper` (*Class SUNNonLinSolWrapper*)

Class Documentation

class **SUNNonLinSolFixedPoint** : public amici::*SUNNonLinSolWrapper*

SUNDIALS Fixed point non-linear solver to solve G(y) = y.

Public Functions

explicit **SUNNonLinSolFixedPoint**(*const_N_Vector* x, int m = 0)

Create fixed-point solver.

Parameters

- **x** – template for cloning vectors needed within the solver.
- **m** – number of acceleration vectors to use

SUNNonLinSolFixedPoint(int count, *const_N_Vector* x, int m = 0)

Create fixed-point solver for use with sensitivity analysis.

Parameters

- **count** – Number of vectors in the nonlinear solve. When integrating a system containing Ns sensitivities the value of count is:
  - Ns+1 if using a simultaneous corrector approach.
  - Ns if using a staggered corrector approach.
- **x** – template for cloning vectors needed within the solver.
- **m** – number of acceleration vectors to use

int **getSysFn**(SUNNonlinSolSysFn *SysFn) const

Get function to evaluate the fixed point function G(y) = y.

Parameters **SysFn** –

Returns

Class SUNNonLinSolNewton

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Base Type

- `public amici::SUNNonLinSolWrapper` (*Class SUNNonLinSolWrapper*)

Class Documentation

class **SUNNonLinSolNewton** : public amici::*SUNNonLinSolWrapper*

SUNDIALS Newton non-linear solver to solve F (y) = 0.

Public Functions

explicit **SUNNonLinSolNewton**(N_Vector x)

Create Newton solver.

Parameters **x** – A template for cloning vectors needed within the solver.

SUNNonLinSolNewton(int count, N_Vector x)

Create Newton solver for enabled sensitivity analysis.

Parameters

- **count** – Number of vectors in the nonlinear solve. When integrating a system containing Ns sensitivities the value of count is:
  - Ns+1 if using a simultaneous corrector approach.
  - Ns if using a staggered corrector approach.
- **x** – A template for cloning vectors needed within the solver.

int **getSysFn**(SUNNonlinSolSysFn *SysFn) const

Get function to evaluate the nonlinear residual function F(y) = 0.

Parameters **SysFn** –

Returns

Class SUNNonLinSolWrapper

- Defined in file_include_amici_sundials_linsol_wrapper.h

Inheritance Relationships

Derived Types

- `public amici::SUNNonLinSolFixedPoint` (*Class SUNNonLinSolFixedPoint*)
- `public amici::SUNNonLinSolNewton` (*Class SUNNonLinSolNewton*)

Class Documentation

class **SUNNonLinSolWrapper**

A RAII wrapper for SUNNonLinearSolver structs which solve the nonlinear system F (y) = 0 or G(y) = y.

Subclassed by *amici::SUNNonLinSolFixedPoint*, *amici::SUNNonLinSolNewton*

Public Functions

explicit **SUNNonLinSolWrapper**(SUNNonlinearSolver sol)

SUNNonLinSolWrapper from existing SUNNonlinearSolver.

Parameters **sol** –

virtual **~SUNNonLinSolWrapper**()

SUNNonLinSolWrapper(const *SUNNonLinSolWrapper* &other) = delete

Copy constructor.

Parameters **other** –

SUNNonLinSolWrapper(*SUNNonLinSolWrapper* &&other) noexcept

Move constructor.

Parameters **other** –

SUNNonLinSolWrapper &**operator=**(const *SUNNonLinSolWrapper* &other) = delete

Copy assignment.

Parameters **other** –

Returns

SUNNonLinSolWrapper &**operator=**(*SUNNonLinSolWrapper* &&other) noexcept

Move assignment.

Parameters **other** –

Returns

SUNNonlinearSolver **get**() const

Get the wrapped SUNNonlinearSolver.

Returns SUNNonlinearSolver

SUNNonlinearSolver_Type **getType**() const

Get type ID of the solver.

Returns

int **setup**(N_Vector y, void *mem)

Setup solver.

Parameters

- **y** – the initial iteration passed to the nonlinear solver.
- **mem** – the sundials integrator memory structure.

Returns

int **Solve**(N_Vector y0, N_Vector y, N_Vector w, *realtype* tol, bool callLSetup, void *mem)

Solve the nonlinear system F (y) = 0 or G(y) = y.

Parameters

- **y0** – the initial iterate for the nonlinear solve. This must remain unchanged throughout the solution process.
- **y** – the solution to the nonlinear system
- **w** – the solution error weight vector used for computing weighted error norms.

- **tol** – the requested solution tolerance in the weighted root-mean- squared norm.
- **callLSetup** – a flag indicating that the integrator recommends for the linear solver setup function to be called.
- **mem** – the sundials integrator memory structure.

Returns

int **setSysFn**(SUNNonlinSolSysFn SysFn)

Set function to evaluate the nonlinear residual function F(y) = 0 or the fixed point function G(y) = y.

Parameters **SysFn** –

Returns

int **setLSetupFn**(SUNNonlinSolLSetupFn SetupFn)

Set linear solver setup function.

Parameters **SetupFn** –

Returns

int **setLSolveFn**(SUNNonlinSolLSolveFn SolveFn)

Set linear solver solve function.

Parameters **SolveFn** –

Returns

int **setConvTestFn**(SUNNonlinSolConvTestFn CTestFn, void *ctest_data)

Set function to test for convergence.

Parameters

- **CTestFn** –
- **ctest_data** –

Returns

int **setMaxIters**(int maxiters)

Set maximum number of non-linear iterations.

Parameters **maxiters** –

Returns

long int **getNumIters**() const

getNumIters

Returns

int **getCurIter**() const

getCurIter

Returns

long int **getNumConvFails**() const

getNumConvFails

Returns

Protected Functions

void **initialize**()

initialize

Protected Attributes

SUNNonlinearSolver **solver** = nullptr

the wrapper solver

Enums

Enum BLASLayout

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**BLASLayout**

BLAS Matrix Layout, affects dgemm and gemv calls

Values:

enumerator **rowMajor**

enumerator **colMajor**

Enum BLASTranspose

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**BLASTranspose**

BLAS Matrix Transposition, affects dgemm and gemv calls

Values:

enumerator **noTrans**

enumerator **trans**

enumerator **conjTrans**

Enum FixedParameterContext

- Defined in file_include_amici_defines.h

Enum Documentation

enum class `amici::`**`FixedParameterContext`**

fixedParameter to be used in condition context

Values:

enumerator **`simulation`**

enumerator **`preequilibration`**

enumerator **`presimulation`**

Enum InternalSensitivityMethod

- Defined in file_include_amici_defines.h

Enum Documentation

enum class `amici::`**`InternalSensitivityMethod`**

CVODES/IDAS forward sensitivity computation method

Values:

enumerator **`simultaneous`**

enumerator **`staggered`**

enumerator **`staggered1`**

Enum InterpolationType

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**InterpolationType**

CVODES/IDAS state interpolation for adjoint sensitivity analysis

Values:

enumerator **hermite**

enumerator **polynomial**

Enum LinearMultistepMethod

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**LinearMultistepMethod**

CVODES/IDAS linear multistep method

Values:

enumerator **adams**

enumerator **BDF**

Enum LinearSolver

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**LinearSolver**

linear solvers for CVODES/IDAS

Values:

enumerator **dense**

enumerator **band**

enumerator **LAPACKDense**

enumerator **LAPACKBand**

enumerator **diag**

enumerator **SPGMR**

enumerator **SPBCG**

enumerator **SPTFQMR**

enumerator **KLU**

enumerator **SuperLUMT**

Enum LogSeverity

- Defined in file_include_amici_logging.h

Enum Documentation

enum class amici::**LogSeverity**

Severity levels for logging.

Values:

enumerator **error**

enumerator **warning**

enumerator **debug**

Enum ModelQuantity

- Defined in file_include_amici_model.h

Enum Documentation

enum class amici::**ModelQuantity**

Describes the various model quantities.

Values:

enumerator **J**

enumerator **JB**

enumerator **Jv**

enumerator **JvB**

enumerator **JDiag**

enumerator **sx**

enumerator **sy**

enumerator **sz**

enumerator **srz**

enumerator **ssigmay**

enumerator **ssigmaz**

enumerator **xdot**

enumerator **sxdot**

enumerator **xBdot**

enumerator **x0_rdata**

enumerator **x0**

enumerator **x_rdata**

enumerator **x**

enumerator **dwdw**

enumerator **dwdx**

enumerator **dwdp**

enumerator **y**

enumerator **dydp**

enumerator **dydx**

enumerator **w**

enumerator **root**

enumerator **qBdot**

enumerator **qBdot_ss**

enumerator **xBdot_ss**

enumerator **JSparseB_ss**

enumerator **deltax**

enumerator **deltasx**

enumerator **deltaxB**

enumerator **k**

enumerator **p**

enumerator **ts**

enumerator **dJydy**

enumerator **dJydy_matlab**

enumerator **deltaqB**

enumerator **dsigmaydp**

enumerator **dsigmaydy**

enumerator **dsigmazdp**

enumerator **dJydsigma**

enumerator **dJydx**

enumerator **dzdx**

enumerator **dzdp**

enumerator **dJrzdsigma**

enumerator **dJrzdz**

enumerator **dJrzdx**

enumerator **dJzdsigma**

enumerator **dJzdz**

enumerator **dJzdx**

enumerator **drzdp**

enumerator **drzdx**

Enum NewtonDampingFactorMode

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**NewtonDampingFactorMode**

Damping factor flag for the Newton method

Values:

enumerator **off**

enumerator **on**

Enum NonlinearSolverIteration

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**NonlinearSolverIteration**

CVODES/IDAS Nonlinear Iteration method

Values:

enumerator **functional**

enumerator **fixedpoint**

deprecated

enumerator **newton**

Enum ObservableScaling

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**ObservableScaling**

modes for observable scaling

Values:

enumerator **lin**

enumerator **log**

enumerator **log10**

Enum ParameterScaling

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**ParameterScaling**

modes for parameter transformations

Values:

enumerator **none**

enumerator **ln**

enumerator **log10**

Enum RDataReporting

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**RDataReporting**

Values:

enumerator **full**

enumerator **residuals**

enumerator **likelihood**

Enum SecondOrderMode

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SecondOrderMode**

modes for second order sensitivity analysis

Values:

enumerator **none**

enumerator **full**

enumerator **directional**

Enum SensitivityMethod

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SensitivityMethod**

methods for sensitivity computation

Values:

enumerator **none**

enumerator **forward**

enumerator **adjoint**

Enum SensitivityOrder

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SensitivityOrder**

orders of sensitivity analysis

Values:

enumerator **none**

enumerator **first**

enumerator **second**

Enum SteadyStateContext

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SteadyStateContext**

Context for which the sensitivity flag should be computed

Values:

enumerator **newtonSensi**

enumerator **sensiStorage**

enumerator **solverCreation**

Enum SteadyStateSensitivityMode

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SteadyStateSensitivityMode**

Sensitivity computation mode in steadyStateProblem

Values:

enumerator **newtonOnly**

enumerator **integrationOnly**

enumerator **integrateIfNewtonFails**

Enum SteadyStateStatus

- Defined in file_include_amici_defines.h

Enum Documentation

enum class amici::**SteadyStateStatus**

State in which the steady state computation finished

Values:

enumerator **failed_too_long_simulation** enumerator **failed_damping**

enumerator **failed_factorization**

enumerator **failed_convergence**

enumerator **failed**

enumerator **not_run**

enumerator **success**

Functions

Template Function amici::addSlice(const gsl::span<T const>, gsl::span<T>)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**addSlice**(const gsl::span<*T* const> slice, gsl::span<*T*> buffer)

local helper function to add the computed slice to provided buffer (span)

Parameters

- **slice** – computed value
- **buffer** – buffer to which values are to be added

Template Function amici::addSlice(std::vector<T> const&, gsl::span<T>)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**addSlice**(std::vector<*T*> const &s, gsl::span<*T*> b)

local helper function to add the computed slice to provided buffer (vector/span)

Parameters

- **s** – computed value
- **b** – buffer to which values are to be written

Function amici::amici_daxpy

- Defined in file_include_amici_cblas.h

Function Documentation

void amici::**amici_daxpy**(int n, double alpha, const double *x, int incx, double *y, int incy)

Compute y = a*x + y.

Parameters

- **n** – number of elements in y
- **alpha** – scalar coefficient of x
- **x** – vector of length n*incx
- **incx** – x stride
- **y** – vector of length n*incy
- **incy** – y stride

Function amici::amici_dgemm

- Defined in file_include_amici_cblas.h

Function Documentation

void amici::**amici_dgemm**(*BLASLayout* layout, *BLASTranspose* TransA, *BLASTranspose* TransB, int M, int N, int K, double alpha, const double *A, int lda, const double *B, int ldb, double beta, double *C, int ldc)

CBLAS matrix matrix multiplication (dgemm)

This routines computes $C = alpha * A * B + beta * C$ with A: [MxK] B:[KxN] C:[MxN]

Parameters

- **layout** – memory layout.
- **TransA** – flag indicating whether A should be transposed before multiplication
- **TransB** – flag indicating whether B should be transposed before multiplication
- **M** – number of rows in A/C
- **N** – number of columns in B/C
- **K** – number of rows in B, number of columns in A
- **alpha** – coefficient alpha
- **A** – matrix A
- **lda** – leading dimension of A (>=M or >=K)
- **B** – matrix B
- **ldb** – leading dimension of B (>=K or >=N)
- **beta** – coefficient beta

- **C** – matrix C
- **ldc** – leading dimension of C (>=M or >= N)

Function amici::amici_dgemv

- Defined in file_include_amici_cblas.h

Function Documentation

void amici::**amici_dgemv**(*BLASLayout* layout, *BLASTranspose* TransA, int M, int N, double alpha, const double *A, int lda, const double *X, int incX, double beta, double *Y, int incY)

CBLAS matrix vector multiplication (dgemv).

Computes $y = alpha * A * x + beta * y$ with A: [MxN] x:[Nx1] y:[Mx1]

Parameters

- **layout** – Matrix layout, column major or row major.
- **TransA** – flag indicating whether A should be transposed before multiplication
- **M** – number of rows in A
- **N** – number of columns in A
- **alpha** – coefficient alpha
- **A** – matrix A
- **lda** – leading dimension / stride of A (>=N if row-major, >=M if col-major)
- **X** – vector X
- **incX** – increment for entries of X
- **beta** – coefficient beta
- **Y** – vector Y
- **incY** – increment for entries of Y

Function amici::backtraceString

- Defined in file_include_amici_misc.h

Function Documentation

std::string amici::**backtraceString**(int maxFrames, int const first_frame = 0)

Returns the current backtrace as std::string.

Parameters

- **maxFrames** – Number of frames to include
- **first_frame** – Index of first frame to include

Returns Backtrace

Template Function amici::checkBufferSize

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**checkBufferSize**(gsl::span<*T*> buffer, typename gsl::span<*T*>::index_type expected_size)

local helper to check whether the provided buffer has the expected size

Parameters

- **buffer** – buffer to which values are to be written
- **expected_size** – expected size of the buffer

Function amici::checkSigmaPositivity(std::vector<realtype> const&, char const *)

- Defined in file_include_amici_edata.h

Function Documentation

void amici::**checkSigmaPositivity**(std::vector<*realtype*> const &sigmaVector, char const *vectorName)

checks input vector of sigmas for not strictly positive values

Parameters

- **sigmaVector** – vector input to be checked
- **vectorName** – name of the input

Function amici::checkSigmaPositivity(realtype, char const *)

- Defined in file_include_amici_edata.h

Function Documentation

void amici::**checkSigmaPositivity**(*realtype* sigma, char const *sigmaName)

checks input scalar sigma for not strictly positive value

Parameters

- **sigma** – input to be checked
- **sigmaName** – name of the input

Template Function amici::deserializeFromChar

- Defined in file_include_amici_serialization.h

Function Documentation

template<typename **T**>
T amici::**deserializeFromChar**(const char *buffer, int size)

Deserialize object that has been serialized using serializeToChar.

Parameters

- **buffer** – serialized object
- **size** – length of buffer

Returns The deserialized object

Template Function amici::deserializeFromString

- Defined in file_include_amici_serialization.h

Function Documentation

template<typename **T**>
T amici::**deserializeFromString**(std::string const &serialized)

Deserialize object that has been serialized using serializeToString.

Parameters **serialized** – serialized object

Returns The deserialized object

Function amici::dotProd

- Defined in file_include_amici_vector.h

Function Documentation

inline *realtype* amici::**dotProd**(*AmiVector* const &x, *AmiVector* const &y)

Compute dot product of x and y.

Parameters

- **x** – vector
- **y** – vector

Returns dot product of x and y

Function amici::getScaledParameter

- Defined in file_include_amici_misc.h

Function Documentation

double amici::**getScaledParameter**(double unscaledParameter, *ParameterScaling* scaling)

Apply parameter scaling according to `scaling`

Parameters

- **unscaledParameter** –
- **scaling** – parameter scaling

Returns Scaled parameter

Function amici::getUnscaledParameter

- Defined in file_include_amici_misc.h

Function Documentation

double amici::**getUnscaledParameter**(double scaledParameter, *ParameterScaling* scaling)

Remove parameter scaling according to `scaling`

Parameters

- **scaledParameter** – scaled parameter
- **scaling** – parameter scaling

Returns Unscaled parameter

Function amici::hdf5::attributeExists(H5::H5File const&, const std::string&, const std::string&)

- Defined in file_include_amici_hdf5.h

Function Documentation

bool amici::hdf5::**attributeExists**(H5::H5File const &file, const std::string &optionsObject, const std::string &attributeName)

Check whether an attribute with the given name exists on the given dataset.

Parameters

- **file** – The HDF5 file object
- **optionsObject** – Dataset of which attributes should be checked
- **attributeName** – Name of the attribute of interest

Returns `true` if attribute exists, `false` otherwise

Function amici::hdf5::attributeExists(H5::H5Object const&, const std::string&)

- Defined in file_include_amici_hdf5.h

Function Documentation

bool amici::hdf5::**attributeExists**(H5::H5Object const &object, const std::string &attributeName)

Check whether an attribute with the given name exists on the given object.

Parameters

- **object** – An HDF5 object
- **attributeName** – Name of the attribute of interest

Returns `true` if attribute exists, `false` otherwise

Function amici::hdf5::createAndWriteDouble1DDataset

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**createAndWriteDouble1DDataset**(H5::H5File const &file, std::string const &datasetName, gsl::span<const double> buffer)

Create and write to 1-dimensional native double dataset.

Parameters

- **file** – HDF5 file object
- **datasetName** – Name of dataset to create
- **buffer** – Data to write to dataset

Function amici::hdf5::createAndWriteDouble2DDataset

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**createAndWriteDouble2DDataset**(H5::H5File const &file, std::string const &datasetName, gsl::span<const double> buffer, hsize_t m, hsize_t n)

Create and write to 2-dimensional native double dataset.

Parameters

- **file** – HDF5 file object
- **datasetName** – Name of dataset to create
- **buffer** – Flattened data to write to dataset (assuming row-major)
- **m** – Number of rows in buffer

- **n** – Number of columns buffer

Function amici::hdf5::createAndWriteDouble3DDataset

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**createAndWriteDouble3DDataset**(H5::H5File const &file, std::string const &datasetName, gsl::span<const double> buffer, hsize_t m, hsize_t n, hsize_t o)

Create and write to 3-dimensional native double dataset.

Parameters

- **file** – HDF5 file object
- **datasetName** – Name of dataset to create
- **buffer** – Flattened data to write to dataset (assuming row-major)
- **m** – Length of first dimension in buffer
- **n** – Length of first dimension in buffer
- **o** – Length of first dimension in buffer

Function amici::hdf5::createAndWriteInt1DDataset

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**createAndWriteInt1DDataset**(H5::H5File const &file, std::string const &datasetName, gsl::span<const int> buffer)

Create and write to 1-dimensional native integer dataset.

Parameters

- **file** – HDF5 file object
- **datasetName** – Name of dataset to create
- **buffer** – Data to write to dataset

Function amici::hdf5::createAndWriteInt2DDataset

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::createAndWriteInt2DDataset`(H5::H5File const &file, std::string const &datasetName, gsl::span<const int> buffer, hsize_t m, hsize_t n)

Create and write to 2-dimensional native integer dataset.

Parameters

- **file** – HDF5 file object
- **datasetName** – Name of dataset to create
- **buffer** – Flattened data to write to dataset (assuming row-major)
- **m** – Number of rows in buffer
- **n** – Number of columns buffer

Function amici::hdf5::createGroup

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::createGroup`(const H5::H5File &file, std::string const &groupPath, bool recursively = true)

Create the given group and possibly parents.

Parameters

- **file** – HDF5 file to write to
- **groupPath** – Path to the group to be created
- **recursively** – Create intermediary groups

Function amici::hdf5::createOrOpenForWriting

- Defined in file_include_amici_hdf5.h

Function Documentation

H5::H5File `amici::hdf5::createOrOpenForWriting`(std::string const &hdf5filename)

Open the given file for writing.

Append if exists, create if not.

Parameters **hdf5filename** – File to open

Returns File object

Function amici::hdf5::getDoubleDataset1D

- Defined in file_include_amici_hdf5.h

Function Documentation

std::vector<double> `amici::hdf5::getDoubleDataset1D`(const H5::H5File &file, std::string const &name)

Read 1-dimensional native double dataset from HDF5 file.

Parameters

- **file** – HDF5 file object
- **name** – Name of dataset to read

Returns Data read

Function amici::hdf5::getDoubleDataset2D

- Defined in file_include_amici_hdf5.h

Function Documentation

std::vector<double> `amici::hdf5::getDoubleDataset2D`(const H5::H5File &file, std::string const &name, hsize_t &m, hsize_t &n)

Read 2-dimensional native double dataset from HDF5 file.

Parameters

- **file** – HDF5 file object
- **name** – Name of dataset to read
- **m** – Number of rows in the dataset
- **n** – Number of columns in the dataset

Returns Flattened data (row-major)

Function amici::hdf5::getDoubleDataset3D

- Defined in file_include_amici_hdf5.h

Function Documentation

std::vector<double> `amici::hdf5::getDoubleDataset3D`(const H5::H5File &file, std::string const &name, hsize_t &m, hsize_t &n, hsize_t &o)

Read 3-dimensional native double dataset from HDF5 file.

Parameters

- **file** – HDF5 file object
- **name** – Name of dataset to read

- **m** – Length of first dimension in dataset
- **n** – Length of first dimension in dataset
- **o** – Length of first dimension in dataset

Returns Flattened data (row-major)

Function amici::hdf5::getDoubleScalarAttribute

- Defined in file_include_amici_hdf5.h

Function Documentation

double `amici::hdf5::getDoubleScalarAttribute`(const H5::H5File &file, const std::string &optionsObject, const std::string &attributeName)

Read scalar native double attribute from HDF5 object.

Parameters

- **file** – HDF5 file
- **optionsObject** – Object to read attribute from
- **attributeName** – Name of attribute to read

Returns Attribute value

Function amici::hdf5::getIntDataset1D

- Defined in file_include_amici_hdf5.h

Function Documentation

std::vector<int> `amici::hdf5::getIntDataset1D`(const H5::H5File &file, std::string const &name)

Read 1-dimensional native integer dataset from HDF5 file.

Parameters

- **file** – HDF5 file object
- **name** – Name of dataset to read

Returns Data read

Function amici::hdf5::getIntScalarAttribute

- Defined in file_include_amici_hdf5.h

Function Documentation

int `amici::hdf5::getIntScalarAttribute`(const H5::H5File &file, const std::string &optionsObject, const std::string &attributeName)

Read scalar native integer attribute from HDF5 object.

Parameters

- **`file`** – HDF5 file
- **`optionsObject`** – Object to read attribute from
- **`attributeName`** – Name of attribute to read

Returns Attribute value

Function amici::hdf5::getStringAttribute

- Defined in file_include_amici_hdf5.h

Function Documentation

std::string `amici::hdf5::getStringAttribute`(H5::H5File const &file, std::string const &optionsObject, std::string const &attributeName)

Read string attribute from HDF5 object.

Parameters

- **`file`** – HDF5 file
- **`optionsObject`** – Object to read attribute from
- **`attributeName`** – Name of attribute to read

Returns Attribute value

Function amici::hdf5::locationExists(std::string const&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

bool `amici::hdf5::locationExists`(std::string const &filename, std::string const &location)

Check if the given location (group, link or dataset) exists in the given file.

Parameters

- **`filename`** – HDF5 filename
- **`location`** – Location to test for

Returns `true` if exists, `false` otherwise

Function amici::hdf5::locationExists(H5::H5File const&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

bool `amici::hdf5::locationExists`(H5::H5File const &file, std::string const &location)

Check if the given location (group, link or dataset) exists in the given file.

Parameters

- **`file`** – HDF5 file object
- **`location`** – Location to test for

Returns `true` if exists, `false` otherwise

Function amici::hdf5::readModelDataFromHDF5(std::string const&, Model&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::readModelDataFromHDF5`(std::string const &hdffile, *Model* &model, std::string const &datasetPath)

Read model data from HDF5 file.

Parameters

- **`hdffile`** – Name of HDF5 file
- **`model`** – *Model* to set data on
- **`datasetPath`** – Path inside the HDF5 file

Function amici::hdf5::readModelDataFromHDF5(H5::H5File const&, Model&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::readModelDataFromHDF5`(H5::H5File const &file, *Model* &model, std::string const &datasetPath)

Read model data from HDF5 file.

Parameters

- **`file`** – HDF5 file handle to read from
- **`model`** – *Model* to set data on
- **`datasetPath`** – Path inside the HDF5 file

Function amici::hdf5::readSimulationExpData

- Defined in file_include_amici_hdf5.h

Function Documentation

std::unique_ptr<*ExpData*> `amici::hdf5::readSimulationExpData`(const std::string &hdf5Filename, const std::string &hdf5Root, const *Model* &model)

Read AMICI *ExpData* data from HDF5 file.

Parameters

- **hdf5Filename** – Name of HDF5 file
- **hdf5Root** – Path inside the HDF5 file to object having *ExpData*
- **model** – The model for which data is to be read

Returns *ExpData* created from data in the given location

Function amici::hdf5::readSolverSettingsFromHDF5(const H5::H5File&, Solver&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::readSolverSettingsFromHDF5`(const H5::H5File &file, *Solver* &solver, std::string const &datasetPath)

Read solver options from HDF5 file.

Parameters

- **file** – HDF5 file to read from
- **solver** – *Solver* to set options on
- **datasetPath** – Path inside the HDF5 file

Function amici::hdf5::readSolverSettingsFromHDF5(std::string const&, Solver&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**readSolverSettingsFromHDF5**(std::string const &hdffile, *Solver* &solver, std::string const &datasetPath)

Read solver options from HDF5 file.

Parameters

- **hdffile** – Name of HDF5 file
- **solver** – *Solver* to set options on
- **datasetPath** – Path inside the HDF5 file

Function amici::hdf5::writeReturnData(const ReturnData&, H5::H5File const&, const std::string&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**writeReturnData**(const *ReturnData* &rdata, H5::H5File const &file, const std::string &hdf5Location)

Write *ReturnData* to HDF5 file.

Parameters

- **rdata** – Data to write
- **file** – HDF5 file to write to
- **hdf5Location** – Full dataset path inside the HDF5 file (will be created)

Function amici::hdf5::writeReturnData(const ReturnData&, std::string const&, const std::string&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**writeReturnData**(const *ReturnData* &rdata, std::string const &hdf5Filename, const std::string &hdf5Location)

Write *ReturnData* to HDF5 file.

Parameters

- **rdata** – Data to write
- **hdf5Filename** – Filename of HDF5 file
- **hdf5Location** – Full dataset path inside the HDF5 file (will be created)

Function amici::hdf5::writeReturnDataDiagnosis

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::writeReturnDataDiagnosis`(const *ReturnData* &rdata, H5::H5File const &file, const std::string &hdf5Location)

Write *ReturnData* diagnosis data to HDF5 file.

Parameters

- **rdata** – Data to write
- **file** – HDF5 file to write to
- **hdf5Location** – Full dataset path inside the HDF5 file (will be created)

Function amici::hdf5::writeSimulationExpData

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::writeSimulationExpData`(const *ExpData* &edata, H5::H5File const &file, const std::string &hdf5Location)

Write AMICI experimental data to HDF5 file.

Parameters

- **edata** – The experimental data which is to be written
- **file** – Name of HDF5 file
- **hdf5Location** – Path inside the HDF5 file to object having *ExpData*

Function amici::hdf5::writeSolverSettingsToHDF5(Solver const&, std::string const&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void `amici::hdf5::writeSolverSettingsToHDF5`(*Solver* const &solver, std::string const &hdf5Filename, std::string const &hdf5Location)

Write solver options to HDF5 file.

Parameters

- **hdf5Filename** – Name of HDF5 file to write to
- **solver** – *Solver* to write options from
- **hdf5Location** – Path inside the HDF5 file

Function amici::hdf5::writeSolverSettingsToHDF5(Solver const&, H5::H5File const&, std::string const&)

- Defined in file_include_amici_hdf5.h

Function Documentation

void amici::hdf5::**writeSolverSettingsToHDF5**(*Solver* const &solver, H5::H5File const &file, std::string const &hdf5Location)

Write solver options to HDF5 file.

Parameters

- **file** – File to read from
- **solver** – *Solver* to write options from
- **hdf5Location** – Path inside the HDF5 file

Template Function amici::is_equal

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
bool amici::**is_equal**(*T* const &a, *T* const &b)

Check if two spans are equal, treating NaNs in the same position as equal.

Parameters

- **a** –
- **b** –

Returns Whether the contents of the two spans are equal.

Function amici::linearSum

- Defined in file_include_amici_vector.h

Function Documentation

inline void amici::**linearSum**(*realtype* a, *AmiVector* const &x, *realtype* b, *AmiVector* const &y, *AmiVector* &z)

Computes z = a*x + b*y.

Parameters

- **a** – coefficient for x
- **x** – a vector
- **b** – coefficient for y

- **y** – another vector with same size as x
- **z** – result vector of same size as x and y

Function amici::N_VGetArrayPointerConst

- Defined in file_include_amici_vector.h

Function Documentation

inline const *realtype* *amici::**N_VGetArrayPointerConst**(*const_N_Vector* x)

Function amici::operator==(ExpData const&, ExpData const&)

- Defined in file_include_amici_edata.h

Function Documentation

inline bool amici::**operator==**(*ExpData* const &lhs, *ExpData* const &rhs)

Equality operator.

Parameters

- **lhs** – some object
- **rhs** – another object

Returns `true`, if both arguments are equal; `false` otherwise.

Function amici::operator==(const Model&, const Model&)

- Defined in file_include_amici_model.h

Function Documentation

bool amici::**operator==**(const *Model* &a, const *Model* &b)

Parameters

- **a** – First model instance
- **b** – Second model instance

Returns Equality

Function amici::operator==(const ModelDimensions&, const ModelDimensions&)

- Defined in file_include_amici_model.h

Function Documentation

bool `amici::operator==`(const *ModelDimensions* &a, const *ModelDimensions* &b)

Function amici::operator==(const ModelState&, const ModelState&)

- Defined in file_include_amici_model_state.h

Function Documentation

inline bool `amici::operator==`(const *ModelState* &a, const *ModelState* &b)

Function amici::operator==(const SimulationParameters&, const SimulationParameters&)

- Defined in file_include_amici_simulation_parameters.h

Function Documentation

bool `amici::operator==`(const *SimulationParameters* &a, const *SimulationParameters* &b)

Function amici::operator==(const Solver&, const Solver&)

- Defined in file_include_amici_solver.h

Function Documentation

bool `amici::operator==`(const *Solver* &a, const *Solver* &b)

Parameters

- **a** –
- **b** –

Returns

Function amici::printfToString

- Defined in file_include_amici_misc.h

Function Documentation

std::string amici::**printfToString**(char const *fmt, va_list ap)

Format printf-style arguments to std::string.

Parameters

- **fmt** – Format string
- **ap** – Argument list pointer

Returns Formatted String

Function amici::regexErrorToString

- Defined in file_include_amici_misc.h

Function Documentation

std::string amici::**regexErrorToString**(std::regex_constants::error_type err_type)

Convert std::regex_constants::error_type to string.

Parameters **err_type** – error type

Returns Error type as string

Function amici::runAmiciSimulation

- Defined in file_include_amici_amici.h

Function Documentation

std::unique_ptr<*ReturnData*> amici::**runAmiciSimulation**(*Solver* &solver, *ExpData* const *edata, *Model* &model, bool rethrow = false)

Core integration routine. Initializes the solver and runs the forward and backward problem.

Parameters

- **solver** – *Solver* instance
- **edata** – pointer to experimental data object
- **model** – model specification object
- **rethrow** – rethrow integration exceptions?

Returns rdata pointer to return data object

Function amici::runAmiciSimulations

- Defined in file_include_amici_amici.h

Function Documentation

std::vector<std::unique_ptr<*ReturnData*>> `amici::runAmiciSimulations`(*Solver* const &solver, std::vector<*ExpData**> const &edatas, *Model* const &model, bool failfast, int num_threads)

Same as runAmiciSimulation, but for multiple *ExpData* instances. When compiled with OpenMP support, this function runs multi-threaded.

Parameters

- **solver** – *Solver* instance
- **edatas** – experimental data objects
- **model** – model specification object
- **failfast** – flag to allow early termination
- **num_threads** – number of threads for parallel execution

Returns vector of pointers to return data objects

Function amici::scaleParameters

- Defined in file_include_amici_misc.h

Function Documentation

void `amici::scaleParameters`(gsl::span<*realtype* const> bufferUnscaled, gsl::span<*ParameterScaling* const> pscale, gsl::span<*realtype*> bufferScaled)

Apply parameter scaling according to `scaling`

Parameters

- **bufferUnscaled** –
- **pscale** – parameter scaling
- **bufferScaled** – destination

Template Function amici::serializeToChar

- Defined in file_include_amici_serialization.h

Function Documentation

template<typename **T**>
char *`amici::`**`serializeToChar`**(*T* const &data, int *size)

Serialize object to char array.

Parameters

- **`data`** – input object
- **`size`** – maximum char length

Returns The object serialized as char

Template Function amici::serializeToStdVec

- Defined in file_include_amici_serialization.h

Function Documentation

template<typename **T**>
std::vector<char> `amici::`**`serializeToStdVec`**(*T* const &data)

Serialize object to std::vector<char>

Parameters **`data`** – input object

Returns The object serialized as std::vector<char>

Template Function amici::serializeToString

- Defined in file_include_amici_serialization.h

Function Documentation

template<typename **T**>
std::string `amici::`**`serializeToString`**(*T* const &data)

Serialize object to string.

Parameters **`data`** – input object

Returns The object serialized as string

Function amici::simulation_status_to_str

- Defined in file_include_amici_amici.h

Function Documentation

std::string amici::**simulation_status_to_str**(int status)

Get the string representation of the given simulation status code (see *ReturnData::status*).

Parameters **status** – Status code

Returns Name of the variable representing this status code.

Template Function amici::slice(std::vector<T>&, int, unsigned)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
gsl::span<*T*> amici::**slice**(std::vector<*T*> &data, int index, unsigned size)

creates a slice from existing data

Parameters

- **data** – to be sliced
- **index** – slice index
- **size** – slice size

Returns span of the slice

Template Function amici::slice(std::vector<T> const&, int, unsigned)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
gsl::span<*T* const> amici::**slice**(std::vector<*T*> const &data, int index, unsigned size)

creates a constant slice from existing constant data

Parameters

- **data** – to be sliced
- **index** – slice index
- **size** – slice size

Returns span of the slice

Function amici::unravel_index(size_t, size_t)

- Defined in file_include_amici_misc.h

Function Documentation

auto amici::**unravel_index**(size_t flat_idx, size_t num_cols) -> std::pair<size_t, size_t>

Convert a flat index to a pair of row/column indices, assuming row-major order.

Parameters

- **flat_idx** – flat index
- **num_cols** – number of columns of referred to matrix

Returns row index, column index

Function amici::unravel_index(sunindextype, SUNMatrix)

- Defined in file_include_amici_sundials_matrix_wrapper.h

Function Documentation

auto amici::**unravel_index**(sunindextype i, SUNMatrix m) -> std::pair<sunindextype, sunindextype>

Convert a flat index to a pair of row/column indices.

Parameters

- **i** – flat index
- **m** – referred to matrix

Returns row index, column index

Function amici::unscaleParameters

- Defined in file_include_amici_misc.h

Function Documentation

void amici::**unscaleParameters**(gsl::span<*realtype* const> bufferScaled, gsl::span<*ParameterScaling* const> pscale, gsl::span<*realtype*> bufferUnscaled)

Remove parameter scaling according to the parameter scaling in pscale.

All vectors must be of same length.

Parameters

- **bufferScaled** – scaled parameters
- **pscale** – parameter scaling
- **bufferUnscaled** – unscaled parameters are written to the array

Function amici::wrapErrHandlerFn

- Defined in file_include_amici_solver.h

Function Documentation

void amici::**wrapErrHandlerFn**(int error_code, const char *module, const char *function, char *msg, void *eh_data)

Extracts diagnosis information from solver memory block and passes them to the specified output function.

Parameters

- **error_code** – error identifier
- **module** – name of the module in which the error occurred
- **function** – name of the function in which the error occurred
- **msg** – error message
- **eh_data** – *amici::Solver* as void*

Template Function amici::writeSlice(const gsl::span<T const>, gsl::span<T>)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**writeSlice**(const gsl::span<*T* const> slice, gsl::span<*T*> buffer)

local helper function to write computed slice to provided buffer (span)

Parameters

- **slice** – computed value
- **buffer** – buffer to which values are to be written

Template Function amici::writeSlice(std::vector<T> const&, std::vector<T>&)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**writeSlice**(std::vector<*T*> const &s, std::vector<*T*> &b)

local helper function to write computed slice to provided buffer (vector)

Parameters

- **s** – computed value
- **b** – buffer to which values are to be written

Template Function amici::writeSlice(std::vector<T> const&, gsl::span<T>)

- Defined in file_include_amici_misc.h

Function Documentation

template<class **T**>
void amici::**writeSlice**(std::vector<*T*> const &s, gsl::span<*T*> b)

local helper function to write computed slice to provided buffer (vector/span)

Parameters

- **s** – computed value
- **b** – buffer to which values are to be written

Function amici::writeSlice(AmiVector const&, gsl::span<realtype>)

- Defined in file_include_amici_misc.h

Function Documentation

void amici::**writeSlice**(*AmiVector* const &s, gsl::span<*realtype*> b)

local helper function to write computed slice to provided buffer (AmiVector/span)

Parameters

- **s** – computed value
- **b** – buffer to which values are to be written

Template Function boost::serialization::archiveVector

- Defined in file_include_amici_serialization.h

Function Documentation

template<class **Archive**, typename **T**>
void boost::serialization::**archiveVector**(*Archive* &ar, *T* **p, int size)

Serialize a raw array to a boost archive.

Parameters

- **ar** – archive
- **p** – Pointer to array
- **size** – Size of p

Template Function boost::serialization::serialize(Archive&, amici::Model&, unsigned int)

- Defined in file_include_amici_model.h

Function Documentation

template<class **Archive**>
void `boost::serialization::`**`serialize`**(*Archive* &ar, amici::*Model* &m, unsigned int version)

Template Function boost::serialization::serialize(Archive&, amici::ReturnData&, unsigned int)

- Defined in file_include_amici_rdata.h

Function Documentation

template<class **Archive**>
void `boost::serialization::`**`serialize`**(*Archive* &ar, amici::*ReturnData* &r, unsigned int version)

Template Function boost::serialization::serialize(Archive&, amici::Solver&, unsigned int)

- Defined in file_include_amici_solver.h

Function Documentation

template<class **Archive**>
void `boost::serialization::`**`serialize`**(*Archive* &ar, amici::*Solver* &s, unsigned int version)

Template Function boost::serialization::serialize(Archive&, amici::CVodeSolver&, unsigned int)

- Defined in file_include_amici_solver_cvodes.h

Function Documentation

template<class **Archive**>
void `boost::serialization::`**`serialize`**(*Archive* &ar, amici::*CVodeSolver* &s, unsigned int version)

Template Function boost::serialization::serialize(Archive&, amici::IDASolver&, unsigned int)

- Defined in file_include_amici_solver_idas.h

Function Documentation

template<class **Archive**>
void boost::serialization::**serialize**(*Archive* &ar, amici::*IDASolver* &s, unsigned int version)

Function gsl::make_span(SUNMatrix)

- Defined in file_include_amici_sundials_matrix_wrapper.h

Function Documentation

inline span<realtype> gsl::**make_span**(SUNMatrix m)

Create span from SUNMatrix.

Parameters **m** – SUNMatrix

Returns Created span

Function gsl::make_span(N_Vector)

- Defined in file_include_amici_vector.h

Function Documentation

inline span<realtype> gsl::**make_span**(N_Vector nv)

Create span from N_Vector.

Parameters **nv** –

Returns

Variables

Variable amici::AMICI_CONV_FAILURE

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_CONV_FAILURE** = -4

Variable amici::AMICI_DAMPING_FACTOR_ERROR

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_DAMPING_FACTOR_ERROR** = -86

Variable amici::AMICI_DATA_RETURN

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_DATA_RETURN** = 1

Variable amici::AMICI_ERR_FAILURE

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_ERR_FAILURE** = -3

Variable amici::AMICI_ERROR

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_ERROR** = -99

Variable amici::AMICI_FIRST_RHSFUNC_ERR

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_FIRST_RHSFUNC_ERR` = -9

Variable amici::AMICI_ILL_INPUT

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_ILL_INPUT` = -22

Variable amici::AMICI_MAX_TIME_EXCEEDED

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_MAX_TIME_EXCEEDED` = -1000

Variable amici::AMICI_NO_STEADY_STATE

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_NO_STEADY_STATE` = -81

Variable amici::AMICI_NORMAL

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_NORMAL** = 1

Variable amici::AMICI_NOT_IMPLEMENTED

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_NOT_IMPLEMENTED** = -999

Variable amici::AMICI_ONE_STEP

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_ONE_STEP** = 2

Variable amici::AMICI_ONEOUTPUT

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_ONEOUTPUT** = 5

Variable amici::AMICI_PREEQUILIBRATE

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_PREEQUILIBRATE** = -1

Variable amici::AMICI_RECOVERABLE_ERROR

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_RECOVERABLE_ERROR` = 1

Variable amici::AMICI_RHSFUNC_FAIL

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_RHSFUNC_FAIL` = -8

Variable amici::AMICI_ROOT_RETURN

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_ROOT_RETURN` = 2

Variable amici::AMICI_SINGULAR_JACOBIAN

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int `amici::AMICI_SINGULAR_JACOBIAN` = -809

Variable amici::AMICI_SUCCESS

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_SUCCESS** = 0

Variable amici::AMICI_TOO_MUCH_ACC

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_TOO_MUCH_ACC** = -2

Variable amici::AMICI_TOO_MUCH_WORK

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_TOO_MUCH_WORK** = -1

Variable amici::AMICI_UNRECOVERABLE_ERROR

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr int amici::**AMICI_UNRECOVERABLE_ERROR** = -10

Variable amici::model_quantity_to_str

- Defined in file_include_amici_model.h

Variable Documentation

const std::map<*ModelQuantity*, std::string> amici::**model_quantity_to_str**

Variable amici::pi

- Defined in file_include_amici_defines.h

Variable Documentation

constexpr double `amici::pi` = 3.14159265358979323846

Defines

Define _USE_MATH_DEFINES

- Defined in file_include_amici_defines.h

Define Documentation

`_USE_MATH_DEFINES`

Define AMICI_H5_RESTORE_ERROR_HANDLER

- Defined in file_include_amici_hdf5.h

Define Documentation

`AMICI_H5_RESTORE_ERROR_HANDLER`

Define AMICI_H5_SAVE_ERROR_HANDLER

- Defined in file_include_amici_hdf5.h

Define Documentation

`AMICI_H5_SAVE_ERROR_HANDLER`

Define AMICI_VERSION

- Defined in file_include_amici_version.in.h

Define Documentation

`AMICI_VERSION`

Define M_1_PI

- Defined in file_include_amici_defines.h

Define Documentation

`M_1_PI`

Define M_2_PI

- Defined in file_include_amici_defines.h

Define Documentation

`M_2_PI`

Define M_2_SQRTPI

- Defined in file_include_amici_defines.h

Define Documentation

`M_2_SQRTPI`

Define M_E

- Defined in file_include_amici_defines.h

Define Documentation

M_E

Define M_LN10

- Defined in file_include_amici_defines.h

Define Documentation

M_LN10

Define M_LN2

- Defined in file_include_amici_defines.h

Define Documentation

M_LN2

Define M_LOG10E

- Defined in file_include_amici_defines.h

Define Documentation

M_LOG10E

Define M_LOG2E

- Defined in file_include_amici_defines.h

Define Documentation

M_LOG2E

Define M_PI

- Defined in file_include_amici_defines.h

Define Documentation

M_PI

Define M_PI_2

- Defined in file_include_amici_defines.h

Define Documentation

M_PI_2

Define M_PI_4

- Defined in file_include_amici_defines.h

Define Documentation

M_PI_4

Define M_SQRT1_2

- Defined in file_include_amici_defines.h

Define Documentation

M_SQRT1_2

Define M_SQRT2

- Defined in file_include_amici_defines.h

Define Documentation

M_SQRT2

Typedefs

Typedef amici::const_N_Vector

- Defined in file_include_amici_vector.h

Typedef Documentation

using `amici::`**`const_N_Vector`** = std::add_const_t<typename std::remove_pointer_t<N_Vector>>*

Since const N_Vector is not what we want

Typedef amici::realtype

- Defined in file_include_amici_defines.h

Typedef Documentation

using `amici::`**`realtype`** = double

defines variable type for simulation variables (determines numerical accuracy)

CHAPTER

TWELVE

MATLAB INTERFACE

12.1 Installing the AMICI MATLAB toolbox

To use AMICI from MATLAB, start MATLAB and add the `AMICI/matlab` directory to the MATLAB path. To add all toolbox directories to the MATLAB path, execute the matlab script:

```
installAMICI.m
```

To store the installation for further MATLAB session, the path can be saved via:

```
savepath
```

For the compilation of `.mex` files, MATLAB needs to be configured with a working C++ compiler. The C++ compiler needs to be installed and configured via:

```
mex -setup c++
```

For a list of supported compilers we refer to the respective MathWorks documentation.

12.2 Using AMICI's MATLAB interface

In the following we will give a detailed overview how to specify models in MATLAB and how to call the generated simulation files.

Note: The MATLAB interface requires the MathWorks Symbolic Math Toolbox for model import (but not for model simulation).

The Symbolic Math Toolbox requirement can be circumvented by performing model import using the Python interface. The resulting code can then be used from Matlab (see *Compiling a Python-generated model*).

Warning: Due to changes in the Symbolic Math Toolbox, the last MATLAB release with working AMICI model import is R2017b (see https://github.com/AMICI-dev/AMICI/issues/307).

12.2.1 Specifying models in Matlab

This guide will guide the user on how to specify models in MATLAB. For example implementations see the examples in the `matlab/examples` directory.

Header

The model definition needs to be defined as a function which returns a `struct` with all symbolic definitions and options.

```
function [model] = example_model_syms()
```

Options

Set the options by specifying the respective field of the model struct

```
model.(fieldname) = value
```

The options specify default options for simulation, parametrisation and compilation. All of these options are optional.

field	description	default
.param	default parametrisation 'log'/'log10'/'lin'	'lin'
.debug	flag to compile with debug symbols	false
.forward	flag to activate forward sensitivities	true
.adjoint	flag to activate adjoint sensitivities	true

When set to `false`, the fields `forward` and `adjoint` will speed up the time required to compile the model but also disable the respective sensitivity computation.

States

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily.

```
syms state1 state2 state3
```

Create the state vector containing all states:

```
model.sym.x = [ state1 state2 state3 ];
```

Parameters

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities will be derived for all *parameters*.

```
syms param1 param2 param3 param4 param5 param6
```

Create the parameters vector

```
model.sym.p = [ param1 param2 param3 param4 param5 param6 ];
```

Constants

Create the respective symbolic variables. The name of the symbolic variable can be chosen arbitrarily. Sensitivities with respect to *constants* will not be derived.

```
syms const1 const2
```

Create the constants vector

```
model.sym.k = [ const1 const2 ];
```

Differential equations

For time-dependent differential equations you can specify a symbolic variable for time. This **needs** to be denoted by `t`.

```
syms t
```

Specify the right hand side of the differential equation `f` or `xdot`

```
model.sym.xdot(1) = [ const1 - param1*state1 ];
model.sym.xdot(2) = [ +param2*state1 + dirac(t-param3) - const2*state2 ];
model.sym.xdot(3) = [ param4*state2 ];
```

or

```
model.sym.f(1) = [ const1 - param1*state1 ];
model.sym.f(2) = [ +param2*state1 + dirac(t-param3) - const2*state2 ];
model.sym.f(3) = [ param4*state2 ];
```

The specification of `f` or `xdot` may depend on states, parameters and constants.

For DAEs also specify the mass matrix.

```
model.sym.M = [1, 0, 0;...
               0, 1, 0;...
               0, 0, 0];
```

The specification of `M` may depend on parameters and constants.

For ODEs the integrator will solve the equation $\dot{x} = f$ and for DAEs the equations $M \cdot \dot{x} = f$. AMICI will decide whether to use CVODES (for ODEs) or IDAS (for DAEs) based on whether the mass matrix is defined or not.

In the definition of the differential equation you can use certain symbolic functions. For a full list of available functions see `src/symbolic_functions.cpp`.

Dirac functions can be used to cause a jump in the respective states at the specified time-point. This is typically used to model injections, or other external stimuli. Spline functions can be used to model time/state dependent response with unknown time/state dependence.

Initial Conditions

Specify the initial conditions. These may depend on parameters on constants and must have the same size as x.

```
model.sym.x0 = [ param4, 0, 0 ];
```

Observables

Specify the observables. These may depend on parameters and constants.

```
model.sym.y(1) = state1 + state2;
model.sym.y(2) = state3 - state2;
```

In the definition of the observable you can use certain symbolic functions. For a full list of available functions see src/symbolic_functions.cpp. Dirac functions in observables will have no effect.

Events

Specifying events is optional. Events are specified in terms of a trigger function, a bolus function and an output function. The roots of the trigger function defines the occurrences of the event. The bolus function defines the change in the state on event occurrences. The output function defines the expression which is evaluated and reported by the simulation routine on every event occurrence. The user can create events by constructing a vector of objects of the class amievent.

```
model.sym.event(1) = amievent(state1 - state2,0,[]);
```

Events may depend on states, parameters and constants but *not* on observables.

For more details about event support see:

> Fröhlich, F., Theis, F. J., Rädler, J. O., & Hasenauer, J. (2017). Parameter estimation for dynamical systems with discrete events and logical operations. Bioinformatics, 33(7), 1049-1056. doi:10.1093/bioinformatics/btw764.

Standard deviation

Specifying standard deviations is optional. It only has an effect when computing adjoint sensitivities. It allows the user to specify standard deviations of experimental data for observables and events.

Standard deviation for observable data is denoted by sigma_y

```
model.sym.sigma_y(1) = param5;
```

Standard deviation for event data is denoted by sigma_t

```
model.sym.sigma_t(1) = param6;
```

Both sigma_y and sigma_t can either be a scalar or of the same dimension as the observables / events function. They can depend on time and parameters but must not depend on the states or observables. The values provided in sigma_y and sigma_t will only be used if the value in D.Sigma_Y or D.Sigma_T in the user-provided data struct is NaN. See simulation for details.

Objective Function

By default, AMICI assumes a normal noise model and uses the corresponding negative log-likelihood

$$J = 1/2 * sum(((y_i(t) - my_t i)/sigma_{yi})^2 + log(2 * pi * sigma_y^2)$$

as objective function. A user provided objective function can be specified in

```
model.sym.Jy
```

As reference see the default specification of `this.sym.Jy` in `amimodel.makeSyms`.

12.2.2 SBML

AMICI can also import SBML models using the command `SBML2AMICI`. This will generate a model specification as described above, which may be edited by the user to apply further changes.

12.2.3 Model Compilation

The model can then be compiled by calling `amiwrap.m`:

```
amiwrap(modelname,'example_model_syms',dir,o2flag)
```

Here `modelname` should be a string defining the name of the model, `dir` should be a string containing the path to the directory in which simulation files should be placed and `o2flag` is a flag indicating whether second order sensitivities should also be compiled. The user should make sure that the previously defined function `example_model_syms` is in the user path. Alternatively, the user can also call the function `example_model_syms`

```
[model] = example_model_syms()
```

and subsequently provide the generated struct to `amiwrap(...)`, instead of providing the symbolic function:

```
amiwrap(modelname,model,dir,o2flag)
```

In a similar fashion, the user could also generate multiple models and pass them directly to `amiwrap(...)` without generating respective model definition scripts.

Compiling a Python-generated model

For better performance or to avoid the Symbolic Math Toolbox requirement, it might be desirable to import a model in Python and compile the resulting code into a mex file. For Python model import, consult the respective section of the Python documentation. Once the imported succeeded, there will be a `compileMexFile.m` script inside the newly created model directory which can be invoked to compile the mex file. This mex file and `simulate_*.m` can be used as if fully created by matlab.

Using Python-AMICI model import from Matlab

With recent matlab versions it is possible to use the AMICI python package from within Matlab. This not quite comfortable yet, but it is possible.

Here for proof of concept:

- Install the python package as described in the documentation
- Ensure `pyversion` shows the correct python version (3.6 or 3.7)
- Then, from within the AMICI `matlab/` directory:

```
sbml_importer = py.amici.SbmlImporter('../python/examples/example_steadystate/model_
↪steadystate_scaled.xml')
sbml_importer.sbml2amici('steadystate', 'steadystate_example_from_python')
model = py.steadystate.getModel()
solver = model.getSolver()
model.setTimepoints(linspace(0, 50, 51))
rdata = py.amici.runAmiciSimulation(model, solver)
result = struct(py.dict(rdata.items()))
t = double(py.array.array('d', result.ts))
x = double(py.array.array('d', result.x.flatten()))
x = reshape(x, flip(double(py.array.array('d', result.x.shape))))
plot(t, x)
```

12.2.4 Model simulation

After the call to `amiwrap(...)` two files will be placed in the specified directory. One is a `_modelname_.mex` and the other is `simulate_*modelname*.m`. The mex file should never be called directly. Instead the MATLAB script, which acts as a wrapper around the .mex simulation file should be used.

The `simulate_ _modelname_.m` itself carries extensive documentation on how to call the function, what it returns and what additional options can be specified. In the following we will give a short overview of possible function calls.

Integration

Define a time vector:

```
t = linspace(0,10,100)
```

Generate a parameter vector:

```
theta = ones(6,1);
```

Generate a constants vector:

```
kappa = ones(2,1);
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the `sol.status` flag. Negative values indicated failed integration. The states will then be available as `sol.x`. The observables will then be available as `sol.y`. The event outputs will then be available as sol.z. If no event occurred there will be an event at the end of the considered interval with the final value of the root function is stored in sol.rz.

Alternatively the integration can also be called via

```
[status,t,x,y] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the flag status . Negative values indicated failed integration. The states will then be available as x. The observables will then be available as y. No event output will be given.

Forward Sensitivities

Set the sensitivity computation to forward sensitivities and integrate:

```
options.sensi = 1;
options.sensi_meth = 'forward';
sol = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the sol.status flag. Negative values indicate failed integration. The states will be available as sol.x, with the derivative with respect to the parameters in sol.sx. The observables will be available as sol.y, with the derivative with respect to the parameters in sol.sy. The event outputs will be available as sol.z, with the derivative with respect to the parameters in sol.sz. If no event occured there will be an event at the end of the considered interval with the final value of the root function stored in sol.rz, with the derivative with respect to the parameters in sol.srz.

Alternatively the integration can also be called via

```
[status,t,x,y,sx,sy] = simulate_modelname(t,theta,kappa,[],options)
```

The integration status will be indicated by the status flag. Negative values indicate failed integration. The states will then be available as x, with derivative with respect to the parameters in sx. The observables will then be available as y, with derivative with respect to the parameters in sy. No event output will be given.

Adjoint sensitivities

Set the sensitivity computation to adjoint sensitivities:

```
options.sensi = 1;
options.sensi_meth = 'adjoint';
```

Define Experimental Data:

```
D.Y = [NaN(1,2)],ones(length(t)-1,2)];
D.Sigma_Y = [0.1*ones(length(t)-1,2),NaN(1,2)];
D.T = ones(1,1);
D.Sigma_T = NaN;
```

The NaN values in Sigma_Y and Sigma_T will be replaced by the specification in model.sym.sigma_y and model.sym.sigma_t. Data points with NaN value will be completely ignored.

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The integration status will be indicated by the sol.status flag. Negative values indicate failed integration. The log-likelihood will then be available as `sol.llh` and the derivative with respect to the parameters in `sol.sllh`. Note that for adjoint sensitivities no state, observable and event sensitivities will be available. Yet this approach can be expected to be significantly faster for systems with a large number of parameters.

Steady-state sensitivities

This will compute state sensitivities according to the formula

$$s_k^x = -\left(\frac{\partial f}{\partial x}\right)^{-1} \frac{\partial f}{\partial \theta_k}$$

In the current implementation this formulation does not allow for conservation laws as this would result in a singular Jacobian.

Set the final timepoint as infinity, this will indicate the solver to compute the steadystate:

```
t = Inf;
```

Set the sensitivity computation to steady state sensitivities:

```
options.sensi = 1;
```

Integrate:

```
sol = simulate_modelname(t,theta,kappa,D,options)
```

The states will be available as `sol.x`, with the derivative with respect to the parameters in `sol.sx`. The observables will be available as `sol.y`, with the derivative with respect to the parameters in `sol.sy`. Notice that for steady state sensitivities no event sensitivities will be available. For the accuracy of the computed derivatives it is essential that the system is sufficiently close to a steady state. This can be checked by examining the right hand side of the system at the final time-point via `sol.diagnosis.xdot`.

12.3 FAQ

Q: My model fails to build.

A: Remove the corresponding model directory located in AMICI/models/*yourmodelname* and compile again.

Q: It still does not compile.

A: Remove the directory AMICI/models/`mexext` and compile again.

Q: It still does not compile.

A: Make an issue and we will have a look.

Q: My Python-generated model does not compile from MATLAB.

A: Try building any of the available examples before. If this succeeds, retry building the original model. Some dependencies might not be built correctly when using only the `compileMexFile.m` script.

Q: I get an out of memory error while compiling my model on a Windows machine.

A: This may be due to an old compiler version. See issue #161 for instructions on how to install a new compiler.

Q: How are events interpreted in a DAE context?

A: Currently we only support impulse free events. Also sensitivities have never been tested. Proceed with care and create an issue if any problems arise!

Q: The simulation/sensitivities I get are incorrect.

A: There are some known issues, especially with adjoint sensitivities, events and DAEs. If your particular problem is not featured in the issues list, please add it!

12.4 AMICI Matlab API

AMICI Matlab library functions

12.4.1 Class Hierarchy

12.4.2 File Hierarchy

12.4.3 Full API

Namespaces

Namespace matlab

Contents

Namespaces

- *Namespace matlab::mixin*

Namespace matlab::mixin

Classes and Structs

Class amidata

- Defined in file_matlab_@amidata_amidata.m

Inheritance Relationships

Base Type

- `public handle`

Class Documentation

`amidata : public handle`

AMIDATA provides a data container to pass experimental data to the simulation routine for likelihood computation. when any of the properties are updated, the class automatically checks consistency of dimension and updates related properties and initialises them with NaNs.

Public Functions

`amidata::amidata(matlabtypesubstitute varargin)`

amidata creates an amidata container for experimental data with specified dimensions amidata.

AMIDATA(amidata) creates a copy of the input container

AMIDATA(struct) tries to creates an amidata container from the input struct. the struct should have the following

AMIDATA(nt,ny,nz,ne,nk) constructs an empty data container with in the provided dimensions intialised with NaNs

fields t [nt,1] Y [nt,ny] Sigma_Y [nt,ny] Z [ne,nz] Sigma_Z [ne,nz] condition [nk,1] conditionPreequilibration [nk,1] if some fields are missing the function will try to initialise them with NaNs with consistent dimensions

param varargin

Public Members

`matlabtypesubstitute nt = 0`

number of timepoints

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute ny = 0`

number of observables

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute nz = 0`

number of event observables

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute ne = 0`

number of events

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute nk = 0`

number of conditions/constants

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute t = double.empty("")`

timepoints of observations

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute Y = double.empty("")`

observations

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute Sigma_Y = double.empty("")`

standard deviation of observations

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute Z = double.empty("")`

event observations

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute Sigma_Z = double.empty("")`

standard deviation of event observations

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute condition = double.empty("")`

experimental condition

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute conditionPreequilibration = double.empty("")`

experimental condition for preequilibration

Default: double.empty("")

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute reinitializeStates = false`

reinitialize states based on fixed parameters after preeq.?

Default: false

Class amievent

- Defined in file_matlab_@amievent_amievent.m

Class Documentation

class amievent

AMIEVENT defines events which later on will be transformed into appropriate C code.

Public Functions

amievent::amievent(matlabtypesubstitute trigger, matlabtypesubstitute bolus, matlabtypesubstitute z)

amievent constructs an amievent object from the provided input.

param trigger trigger function, the event will be triggered on at all roots of this function

param bolus the bolus that will be added to all states on every occurence of the event

param z the event output that will be reported on every occurence of the event

mlhsInnerSubst<::amievent > amievent::setHflag(matlabtypesubstitute hflag)

setHflag sets the hflag property.

param hflag value for the hflag property, type double

retval this updated event definition object

Public Members

::symbolic trigger = sym.empty("")

the trigger function activates the event on every zero crossing

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** sym.empty("")

::symbolic bolus = sym.empty("")

the bolus function defines the change in states that is applied on every event occurence

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** sym.empty("")

::symbolic z = sym.empty("")

output function for the event

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** sym.empty("")

matlabtypesubstitute hflag = logical.empty("")

flag indicating that a heaviside function is present, this helps to speed up symbolic computations

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** logical.empty("")

Class amifun

- Defined in file_matlab_@amifun_amifun.m

Class Documentation

class amifun

AMIFUN defines functions which later on will be transformed into appropriate C code.

Public Functions

amifun::amifun(matlabtypesubstitute funstr, matlabtypesubstitute model)

amievent constructs an amifun object from the provided input.

param funstr name of the requested function

param model amimodel object which carries all symbolic definitions to construct the function

noret::substitute amifun::writeCcode_sensi(::amimodel model, ::fileid fid)

writeCcode_sensi is a wrapper for writeCcode which loops over parameters and reduces overhead by check nonzero values

param model model defintion object

param fid file id in which the final expression is written

retval fid void

noret::substitute amifun::writeCcode(::amimodel model, ::fileid fid)

writeCcode is a wrapper for gccode which initialises data and reduces overhead by check nonzero values

param model model defintion object

param fid file id in which the final expression is written

retval fid void

noret::substitute amifun::writeMcode(::amimodel model)

writeMcode generates matlab evaluable code for specific model functions

param model model defintion object

retval model void

mlhsInnerSubst<::amifun > amifun::gccode(::amimodel model, ::fileid fid)

gccode transforms symbolic expressions into c code and writes the respective expression into a specified file

param model model definition object

param fid file id in which the expression should be written

retval this function definition object

mlhsInnerSubst<::amifun > amifun::getDeps(::amimodel model)

getDeps populates the sensiflag for the requested function

param model model definition object

retval this updated function definition object

mlhsInnerSubst<::amifun > amifun::getArgs(::amimodel model)

getFArgs populates the fargstr property with the argument string of the respective model function (if applicable). model functions are not wrapped versions of functions which have a model specific name and for which the call is solver specific.

param model model definition object

retval this updated function definition object

mlhsInnerSubst<::amifun > amifun::getNVecs()

getfunargs populates the nvecs property with the names of the N_Vector elements which are required in the execution of the function (if applicable). the information is directly extracted from the argument string

retval this updated function definition object

mlhsInnerSubst<::amifun > amifun::getCVar()

getCVar populates the cvar property

retval this updated function definition object

mlhsInnerSubst<::amifun > amifun::getSensiFlag()

getSensiFlag populates the sensiflag property

retval this updated function definition object

mlhsSubst< mlhsInnerSubst<::amifun >,mlhsInnerSubst<::amimodel > > amifun::getSyms(::amimodel model)

getSyms computes the symbolic expression for the requested function

param model model definition object

retval this updated function definition object

retval model updated model definition object

Public Members

::symbolic sym = sym("[]")

symbolic definition struct

Default: sym("[]")

::symbolic sym_noopt = sym("[]")

symbolic definition which was not optimized (no dependencies on w)

Default: sym("[]")

::symbolic strsym = sym("[]")

short symbolic string which can be used for the reuse of precomputed values

Default: sym("[]")

::symbolic strsym_old = sym("[]")

short symbolic string which can be used for the reuse of old values

Default: sym("[]")

::char funstr = char.empty("")

name of the model

Default: char.empty("")

::char cvar = char.empty("")

name of the c variable

Default: char.empty("")

::char argstr = char.empty("")

argument string (solver specific)

Default: char.empty("")

::cell deps = cell.empty("")

dependencies on other functions

Default: cell.empty("")

matlabtypesubstitute nvecs = cell.empty("")

nvec dependencies

Default: cell.empty("")

matlabtypesubstitute sensiflag = logical.empty("")

indicates whether the function is a sensitivity or derivative with respect to parameters

Default: logical.empty("")

Class amimodel

- Defined in file_matlab_@amimodel_amimodel.m

Inheritance Relationships

Base Type

- `public handle`

Class Documentation

amimodel : public handle

AMIMODEL carries all model definitions including functions and events.

Public Functions

amimodel::amimodel(::string symfun, ::string modelname)

amimodel initializes the model object based on the provided symfun and modelname

param symfun this is the string to the function which generates the modelstruct. You can also directly pass the struct here

param modelname name of the model

noret::substitute amimodel::updateRHS(matlabtypesubstitute xdot)

updateRHS updates the private fun property .fun.xdot.sym (right hand side of the differential equation)

param xdot new right hand side of the differential equation

retval xdot void

noret::substitute amimodel::updateModelName(matlabtypesubstitute modelname)

updateModelName updates the modelname

param modelname new modelname

retval modelname void

`noret::substitute amimodel::updateWrapPath(matlabtypesubstitute wrap_path)`

updateModelName updates the modelname

param wrap_path new wrap_path

retval wrap_path void

`noret::substitute amimodel::parseModel()`

parseModel parses the model definition and computes all necessary symbolic expressions.

retval void

`noret::substitute amimodel::generateC()`

generateC generates the c files which will be used in the compilation.

retval void

`noret::substitute amimodel::generateRebuildM()`

generateRebuildM generates a Matlab script for recompilation of this model

retval void

`noret::substitute amimodel::compileC()`

compileC compiles the mex simulation file

retval void

`noret::substitute amimodel::generateM(::amimodel amimodelo2)`

generateM generates the matlab wrapper for the compiled C files.

param amimodelo2 this struct must contain all necessary symbolic definitions for second order sensivities

retval amimodelo2 void

`noret::substitute amimodel::getFun(::struct HTable, ::string funstr)`

getFun generates symbolic expressions for the requested function.

param HTable struct with hashes of symbolic definition from the previous compilation

param funstr function for which symbolic expressions should be computed

retval funstr void

`noret::substitute amimodel::makeEvents()`

makeEvents extracts discontiniuties from the model right hand side and converts them into events

retval void

`noret::substitute amimodel::makeSyms()`

makeSyms extracts symbolic definition from the user provided model and checks them for consistency

retval void

`mlhsInnerSubst<::bool > amimodel::checkDeps(::struct HTable, ::cell deps)`

checkDeps checks the dependencies of functions and populates sym fields if necessary

param HTable struct with reference hashes of functions in its fields

param deps cell array with containing a list of dependencies

retval cflag boolean indicating whether any of the dependencies have changed with respect to the hashes stored in HTable

`mlhsInnerSubst<::struct > amimodel::loadOldHashes()`

loadOldHashes loads information from a previous compilation of the model.

retval HTable struct with hashes of symbolic definition from the previous compilation

`mlhsInnerSubst< matlabtypesubstitute > amimodel::augmento2()`

augmento2 augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward apporach later on.

retval this augmented system which contains symbolic definition of the original system and its sensitivities

`mlhsInnerSubst<::amimodel > amimodel::augmento2vec()`

augmento2 augments the system equation to also include equations for sensitivity equation. This will enable us to compute second order sensitivities in a forward-adjoint or forward-forward apporach later on.

retval modelo2vec augmented system which contains symbolic definition of the original system and its sensitivities

Public Members

`::struct sym = struct.empty("")`

symbolic definition struct

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct.empty(“”)

`::struct fun = struct.empty("")`

struct which stores information for which functions c code needs to be generated

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct.empty(“”)

::amievent event = amievent.empty("")

struct which stores information for which functions c code needs to be generated

---

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** amievent.empty("")

::string modelname = char.empty("")

name of the model

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** char.empty("")

::struct HTable = struct.empty("")

struct that contains hash values for the symbolic model definitions

---

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct.empty("")

::bool debug = false

flag indicating whether debugging symbols should be compiled

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** false

---

::bool adjoint = true

flag indicating whether adjoint sensitivities should be enabled

---

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** true

::bool forward = true

flag indicating whether forward sensitivities should be enabled

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** true

::double t0 = 0

default initial time

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** 0

::string wtype = char.empty("")

type of wrapper (cvodes/idas)

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** char.empty("")

::int nx = double.empty("")

number of states

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::int nxtrue = double.empty("")

number of original states for second order sensitivities

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

`::int ny = double.empty("")`

number of observables

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

`::int nytrue = double.empty("")`

number of original observables for second order sensitivities

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

`::int np = double.empty("")`

number of parameters

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

`::int nk = double.empty("")`

number of constants

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::int ng = double.empty("")

number of objective functions

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::int nevent = double.empty("")

number of events

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::int nz = double.empty("")

number of event outputs

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::int nztrue = double.empty("")

number of original event outputs for second order sensitivities

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*int id = double.empty("")

flag for DAEs

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty(“”)

::int ubw = double.empty("")

upper Jacobian bandwidth

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty(“”)

::int lbw = double.empty("")

lower Jacobian bandwidth

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty(“”)

::int nnz = double.empty("")

number of nonzero entries in Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty(“”)

::*int sparseidx = double.empty("")

dataindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty(“”)

::*int rowvals = double.empty("")

rowindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*int colptrs = double.empty("")

columnindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*int sparseidxB = double.empty("")

dataindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*int rowvalsB = double.empty("")

rowindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*int colptrsB = double.empty("")

columnindexes of sparse Jacobian

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** double.empty("")

::*cell funs = cell.empty("")

cell array of functions to be compiled

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** cell.empty("")

::*cell mfuns = cell.empty("")

cell array of matlab functions to be compiled

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** cell.empty("")

::string coptim = "-O3"

optimisation flag for compilation

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** "-O3"

::string param = "lin"

default parametrisation

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** "lin"

matlabtypesubstitute wrap_path = char.empty("")

path to wrapper

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** char.empty("")

matlabtypesubstitute recompile = false

flag to enforce recompilation of the model

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** false

matlabtypesubstitute cfun = struct.empty("")

storage for flags determining recompilation of individual functions

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct.empty("")

matlabtypesubstitute o2flag = 0

flag which identifies augmented models 0 indicates no augmentation 1 indicates augmentation by first order sensitivities (yields second order sensitivities) 2 indicates augmentation by one linear combination of first order sensitivities (yields hessian-vector product)

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** 0

matlabtypesubstitute z2event = double.empty("")

vector that maps outputs to events

Default: double.empty("")

matlabtypesubstitute splineflag = false

flag indicating whether the model contains spline functions

Default: false

matlabtypesubstitute minflag = false

flag indicating whether the model contains min functions

Default: false

`matlabtypesubstitute maxflag = false`

flag indicating whether the model contains max functions

Default: false

`::int nw = 0`

number of derived variables w, w is used for code optimization to reduce the number of frequently occuring expressions

Default: 0

`::int ndwdx = 0`

number of derivatives of derived variables w, dwdx

Default: 0

`::int ndwdp = 0`

number of derivatives of derived variables w, dwdp

Default: 0

Public Static Functions

`noret::substitute amimodel::compileAndLinkModel(matlabtypesubstitute modelname, matlabtypesubstitute modelSourceFolder, matlabtypesubstitute coptim, matlabtypesubstitute debug, matlabtypesubstitute funs, matlabtypesubstitute cfun)`

compileAndLinkModel compiles the mex simulation file. It does not check if the model files have changed since generating C++ code or whether all files are still present. Use only if you know what you are doing. The safer alternative is rerunning *amiwrap()*.

param modelname name of the model as specified for *amiwrap()*

param modelSourceFolder path to model source directory

param coptim optimization flags

param debug enable debugging

param funs array with names of the model functions, will be guessed from source files if left empty

param cfun struct indicating which files should be recompiled

retval cfun void

`noret::substitute amimodel::generateMatlabWrapper(matlabtypesubstitute nx, matlabtypesubstitute ny, matlabtypesubstitute np, matlabtypesubstitute nk, matlabtypesubstitute nz, matlabtypesubstitute o2flag, ::amimodel amimodelo2, matlabtypesubstitute wrapperFilename, matlabtypesubstitute modelname, matlabtypesubstitute pscale, matlabtypesubstitute forward, matlabtypesubstitute adjoint)`

generateMatlabWrapper generates the matlab wrapper for the compiled C files.

param nx number of states

param ny number of observables

param np number of parameters

param nk number of fixed parameters

param nz number of events

param o2flag o2flag

param amimodelo2 this struct must contain all necessary symbolic definitions for second order sensivities

param wrapperFilename output filename

param modelname name of the model

param pscale default parameter scaling

param forward has forward sensitivity equations

param adjoint has adjoint sensitivity equations

retval adjoint void

Class amioption

- Defined in file_matlab_@amioption_amioption.m

Inheritance Relationships

Base Type

- `public matlab::mixin::CustomDisplay`

Class Documentation

`amioption : public matlab::mixin::CustomDisplay`

AMIOPTION provides an option container to pass simulation parameters to the simulation routine.

Public Functions

`amioption::amioption(matlabtypesubstitute varargin)`

amioptions Construct a new amioptions object OPTS = *amioption()* creates a set of options with each option set to its default value.

OPTS = *amioption(PARAM, VAL, . . .)* creates a set of options with the named parameters altered with the specified values.

OPTS = *amioption(OLDOPTS, PARAM, VAL, . . .)* creates a copy of OLDOPTS with the named parameters altered with the specified value

Note: to see the parameters, check the documentation page for amioption

param varargin input to construct amioption object, see function function description

Public Members

`matlabtypesubstitute atol = 1e-16`

absolute integration tolerace

Default: 1e-16

`matlabtypesubstitute rtol = 1e-8`

relative integration tolerace

Default: 1e-8

`matlabtypesubstitute maxsteps = 1e4`

maximum number of integration steps

Default: 1e4

`matlabtypesubstitute quad_atol = 1e-12`

absolute quadrature tolerace

Default: 1e-12

`matlabtypesubstitute quad_rtol = 1e-8`

relative quadrature tolerace

Default: 1e-8

`matlabtypesubstitute maxstepsB = 0`

maximum number of integration steps

Default: 0

`matlabtypesubstitute ss_atol = 1e-16`

absolute steady state tolerace

Default: 1e-16

`matlabtypesubstitute ss_rtol = 1e-8`

relative steady state tolerace

Default: 1e-8

`matlabtypesubstitute sens_ind = double.empty("")`

index of parameters for which the sensitivities are computed

Default: double.empty("")

`matlabtypesubstitute tstart = 0`

starting time of the simulation

Default: 0

matlabtypesubstitute lmm = 2

linear multistep method.

Default: 2

matlabtypesubstitute iter = 2

iteration method for linear multistep.

Default: 2

matlabtypesubstitute linsol = 9

linear solver

Default: 9

matlabtypesubstitute stldet = true

stability detection flag

Default: true

matlabtypesubstitute interpType = 1

interpolation type

Default: 1

matlabtypesubstitute ism = 1

forward sensitivity mode

Default: 1

matlabtypesubstitute sensi_meth = 1

sensitivity method

Default: 1

Note: This property has custom functionality when its value is changed.

matlabtypesubstitute sensi_meth_preeq = 1

sensitivity method for preequilibration

Default: 1

matlabtypesubstitute sensi = 0

sensitivity order

Default: 0

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute nmaxevent = 10`

number of reported events

Default: 10

`matlabtypesubstitute ordering = 0`

reordering of states

Default: 0

`matlabtypesubstitute ss = 0`

steady state sensitivity flag

Default: 0

`matlabtypesubstitute x0 = double.empty("")`

custom initial state

Default: double.empty(“”)

`matlabtypesubstitute sx0 = double.empty("")`

custom initial sensitivity

Default: double.empty(“”)

`matlabtypesubstitute newton_maxsteps = 40`

newton solver: maximum newton steps

Default: 40

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute z2event = double.empty("")`

mapping of event ouputs to events

Default: double.empty(“”)

`matlabtypesubstitute pscale = "[]"`

parameter scaling Single value or vector matching sens_ind. Valid options are “log”,”log10” and “lin” for log, log10 or unscaled parameters p. Use [] for default as specified in the model (fallback: `lin`).

Default: “[]”

Note: This property has custom functionality when its value is changed.

`matlabtypesubstitute steadyStateSensitivityMode = 0`

Mode for for computing sensitivities ({0: Newton}, 1: Simulation)

Default: 0

Public Static Functions

mlhsInnerSubst< matlabtypesubstitute > amioption::getIntegerPScale(matlabtypesubstitute pscaleString)

pscaleInt converts a parameter scaling string into the corresponding integer representation

param pscaleString parameter scaling string

retval pscaleString int

Class amised

- Defined in file_matlab_@amised_amised.m

Inheritance Relationships

Base Type

- `public handle`

Class Documentation

amised : public handle

AMISED is a container for SED-ML objects.

Public Functions

amised::amised(matlabtypesubstitute sedname)

amised reads in an SEDML document using the JAVA binding of of libSEDML

param sedname name/path of the SEDML document

Public Members

matlabtypesubstitute model = struct("'event',[],'sym',[]")

amimodel from the specified model

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct("'event',[],'sym',[]")

matlabtypesubstitute modelname = {""}

cell array of model identifiers

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** {""}

matlabtypesubstitute sedml = struct.empty("")

stores the struct tree from the xml definition

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** struct.empty("")

matlabtypesubstitute outputcount = "[]"

count the number of outputs per model

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** "[]"

matlabtypesubstitute varidx = "[]"

indexes for dataGenerators

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** "[]"

matlabtypesubstitute varsym = sym("[]")

symbolic expressions for variables

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** sym("[]")

matlabtypesubstitute datasym = sym("[]")

symbolic expressions for data

Note: This property has non-standard access specifiers: `SetAccess = Private, GetAccess = Public`

Note: Matlab documentation of property attributes. **Default:** sym("[]")

Class optsym

- Defined in file_matlab_@optsym_optsym.m

Inheritance Relationships

Base Type

- `public sym`

Class Documentation

optsym : public sym

OPTSYM is an auxiliary class to gain access to the private symbolic property `s` which is necessary to be able to call symobj::optimize on it.

Public Functions

optsym::optsym(::sym symbol)

optsym converts the symbolic object into a optsym object

param symbol symbolic object

mlhsInnerSubst<::sym > optsym::getoptimized()

getoptimized calls symobj::optimize on the optsym object

retval out optimized symbolic object

Functions

Function am_and

- Defined in file_matlab_symbolic_am_and.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_and(::sym a, ::sym b)

am_and is the amici implementation of the symbolic and function

param a first input parameter

param b second input parameter

retval fun logical value, negative for false, positive for true

Function am_eq

- Defined in file_matlab_symbolic_am_eq.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_eq(matlabtypesubstitute varargin)

am_eq is currently a placeholder that simply produces an error message

param varargin elements for chain of equalities

retval fun logical value, negative for false, positive for true

Function am_ge

- Defined in file_matlab_symbolic_am_ge.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_ge(::sym varargin)

am_ge is the amici implementation of the n-ary mathml greaterorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} >= varargin{2},varargin{2} >= varargin{3},. . .)

param varargin chain of input parameters

retval fun a >= b logical value, negative for false, positive for true

Function am_gt

- Defined in file_matlab_symbolic_am_gt.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_gt(::sym varargin)`

am_gt is the amici implementation of the n-ary mathml greaterthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} > varargin{2},varargin{2} > varargin{3},...)

param varargin chain of input parameters

retval fun a > b logical value, negative for false, positive for true

Function am_if

- Defined in file_matlab_symbolic_am_if.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_if(::sym condition, ::sym truepart, ::sym falsepart)`

am_if is the amici implementation of the symbolic if function

param condition logical value

param truepart value if condition is true

param falsepart value if condition is false

retval fun if condition is true truepart, else falsepart

Function am_le

- Defined in file_matlab_symbolic_am_le.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_le(::sym varargin)`

am_le is the amici implementation of the n-ary mathml lessorequal function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} <= varargin{2},varargin{2} <= varargin{3},...)

param varargin chain of input parameters

retval fun a <= b logical value, negative for false, positive for true

Function am_lt

- Defined in file_matlab_symbolic_am_lt.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_lt(::sym varargin)`

am_lt is the amici implementation of the n-ary mathml lessthan function this is an n-ary function, for more than 2 input parameters it will check whether and(varargin{1} < varargin{2},varargin{2} < varargin{3},. . .)

param varargin chain of input parameters

retval fun a < b logical value, negative for false, positive for true

Function am_max

- Defined in file_matlab_symbolic_am_max.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_max(::sym a, ::sym b)`

am_max is the amici implementation of the symbolic max function

param a first input parameter

param b second input parameter

retval fun maximum of a and b

Function am_min

- Defined in file_matlab_symbolic_am_min.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_min(::sym a, ::sym b)`

am_min is the amici implementation of the symbolic min function

param a first input parameter

param b second input parameter

retval fun minimum of a and b

Function am_or

- Defined in file_matlab_symbolic_am_or.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_or(::sym a, ::sym b)

am_or is the amici implementation of the symbolic or function

param a first input parameter

param b second input parameter

retval fun logical value, negative for false, positive for true

Function am_piecewise

- Defined in file_matlab_symbolic_am_piecewise.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_piecewise(matlabtypesubstitute piece, matlabtypesubstitute condition, matlabtypesubstitute default)

am_piecewise is the amici implementation of the mathml piecewise function

param piece value if condition is true

param condition logical value

param default value if condition is false

retval fun return value, piece if condition is true, default if not

Function am_spline

- Defined in file_matlab_symbolic_am_spline.m

Function Documentation

mlhsInnerSubst< matlabtypesubstitute > am_spline(matlabtypesubstitute varargin)

Function am_spline_pos

- Defined in file_matlab_symbolic_am_spline_pos.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_spline_pos(matlabtypesubstitute varargin)`

Function am_stepfun

- Defined in file_matlab_symbolic_am_stepfun.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_stepfun(::sym t, matlabtypesubstitute tstart, matlabtypesubstitute vstart, matlabtypesubstitute tend, matlabtypesubstitute vend)`

am_stepfun is the amici implementation of the step function

param t input variable

param tstart input variable value at which the step starts

param vstart value during the step

param tend input variable value at which the step end

param vend value after the step

retval fun 0 before tstart, vstart between tstart and tend and vend after tend

Function am_xor

- Defined in file_matlab_symbolic_am_xor.m

Function Documentation

`mlhsInnerSubst< matlabtypesubstitute > am_xor(::sym a, ::sym b)`

am_xor is the amici implementation of the symbolic exclusive or function

param a first input parameter

param b second input parameter

retval fun logical value, negative for false, positive for true

Function AMICI2D2D

- Defined in file_matlab_AMICI2D2D.m

Function Documentation

noret::substitute AMICI2D2D(matlabtypesubstitute filename, matlabtypesubstitute modelname)

Function amiwrap

- Defined in file_matlab_amiwrap.m

Function Documentation

noret::substitute amiwrap(matlabtypesubstitute varargin)

AMIWRAP generates c++ mex files for the simulation of systems of differential equations via CVODES and IDAS.

param varargin

```
amiwrap ( modelname, symfun, tdir, o2flag )
```

Required Parameters for varargin:

- modelname specifies the name of the model which will be later used for the naming of the simulation file
- symfun specifies a function which executes model definition
- tdir target directory where the simulation file should be placed **Default:** $AMICIDIR/models/modelname
- o2flag boolean whether second order sensitivities should be enabled **Default:** false

retval o2flag void

Function installAMICI

- Defined in file_matlab_installAMICI.m

Function Documentation

```
noret::substitute installAMICI()
```

Function SBML2AMICI

- Defined in file_matlab_SBML2AMICI.m

Function Documentation

```
noret::substitute SBML2AMICI(matlabtypesubstitute filename,
matlabtypesubstitute modelname)
```

SBML2AMICI generates AMICI model definition files from SBML.

param filename name of the SBML file (withouth extension)

param modelname name of the model, this will define the name of the output file (default: input filename)

retval modelname void

CHAPTER

THIRTEEN

AMICI DEVELOPER'S GUIDE

This document contains information for AMICI developers, not too relevant to regular users.

13.1 Branches / releases

AMICI roughly follows the GitFlow. All new contributions are merged into `develop`. These changes are regularly merged into `master` as new releases. For release versioning we are trying to follow semantic versioning. New releases are created on Github and are automatically deployed to Zenodo for archiving and to obtain a digital object identifier (DOI) to make them citable. Furthermore, our CI pipeline will automatically create and deploy a new release on PyPI.

We try to keep a clean git history. Therefore, feature pull requests are squash-merged to `develop`. Merging of release branches to master is done via merge commits.

13.2 When starting to work on some issue

When starting to work on some Github issue, please assign yourself to let other developers know that you are working on it to avoid duplicate work. If the respective issue is not completely clear, it is generally a good idea to ask for clarification before starting to work on it.

If you want to work on something new, please create a Github issue first.

13.3 Code contributions

When making code contributions, please follow our style guide and the process described below:

- Check if you agree to release your contribution under the conditions provided in `LICENSE`. By opening a pull requests you confirm us that you do agree.
- Start a new branch from `develop` (on your fork, or at the main repository if you have access)
- Implement your changes
- Submit a pull request to the `develop` branch
- Make sure your code is documented appropriately
  - Run `scripts/run-doxygen.sh` to check completeness of your documentation
- Make sure your code is compatible with C++14, `gcc` and `clang` (our CI pipeline will do this for you)
- When adding new functionality, please also provide test cases (see `tests/cpp/` and documentation/CI.md)

- Write meaningful commit messages
- Run all tests to ensure nothing was broken (more details)
  - Run `scripts/buildAll.sh && scripts/run-cpp-tests.sh`.
  - If you made changes to the Matlab or C++ code and have a Matlab license, please also run `tests/cpp/wrapTestModels.m` and `tests/testModels.m`
  - If you made changes to the Python or C++ code, run `make python-tests` in `build`
- When all tests are passing and you think your code is ready to merge, request a code review (see also our code review guideline)
- Wait for feedback. If you do not receive feedback to your pull request within a week, please give us a friendly reminder.

13.3.1 Style guide

General

- All files and functions should come with file-level and function-level documentation.
- All new functionality should be covered by unit or integration tests. Runtime of those tests should be kept as short as possible.

Python

- We want to be compatible with Python 3.8
- For the Python code we want to follow PEP8. Although this is not the case for all existing code, any new contributions should do so.
- We use Python type hints for all functions (but not for class attributes, since they are not supported by the current Python doxygen filter). In Python code type hints should be used instead of doxygen `@type`.

  For function docstrings, follow this format:

```
"""One-line description.

Possible a more detailed description

Arguments:
    Argument1: This needs to start on the same line, otherwise the current
        doxygen filter will fail.

    Returns:
        Return value

    Raises:
        SomeError in case of some error.
"""
```

C++

- We use C++14
- We want to maintain compatibility with g++, clang and the Intel C++ compiler
- For code formatting, we use the settings from `.clang-format` in the root directory
- *Details to be defined*

Matlab

To be defined

13.4 Further topics

13.4.1 AMICI documentation

This file describes how the AMICI documentation is organized and compiled.

Building documentation

Multi-interface documentation

AMICI documentation hosted at Read the Docs (RTD) is generated using Sphinx and related packages. The legacy GitHub Pages URL https://amici-dev.github.io/AMICI/ is set up as a redirect to RTD.

The main configuration file is `documentation/conf.py` and the documentation is generated using `scripts/run-sphinx.sh`. The documentation is written to `documentation/_build/`.

The documentation comprises:

- reStructuredText / Markdown files from `documentation/`
- Python API documentation of native Python modules
- Python API documentation of Python generated via SWIG (doxygen-style comments translated to docstrings by SWIG)
- C++ API documentation (doxygen -> exhale -> breathe -> sphinx)
- Matlab API documentation (mtocpp -> doxygen -> exhale -> breathe -> sphinx)

Doxygen-only (legacy)

(Parts of the) AMICI documentation can also be directly created using doxygen directly. It combines Markdown files from the root directory, from `documentation/` and in-source documentation from the C++ and Matlab source files.

The documentation is generated by running

```
scripts/run-doxygen.sh
```

The resulting HTML and PDF documentation will be created in `doc/`. `scripts/run-doxygen.sh` also checks for any missing in-source documentation.

Doxygen configuration

The main doxygen configuration file is located in `matlab/mtoc/config/Doxyfile.template`. Edit this file for inclusion or exclusion of additional files.

Matlab documentation

Matlab documentation is processed by mtoc++. This is configured in `matlab/mtoc/config`.

Python documentation

Python documentation is processed by doxygen and doxypypy using the script and filters in `scripts/`.

Writing documentation

Out-of-source documentation

Out-of-source documentation files should be written in reStructuredText if intended for Read the Docs or in Markdown if intended for rendering on GitHub. Files to be included in the Sphinx/RTD documentation live in `documentation/`. Graphics for documentation are kept in `documentation/gfx/`.

When using Markdown

- Note that there are some incompatibilities of GitHub Markdown and Doxygen Markdown. Ideally documentation should be written in a format compatible with both. This affects for example images links which currently cause trouble in Doxygen.
- Where possible, relative links are preferred over absolute links. However, they should work with both Github and Doxygen and ideally with local files for offline use.
- Please stick to the limit of 80 characters per line for readability of raw Markdown files where possible.

  However, note that some Markdown interpreters can handle line breaks within links and headings, whereas others cannot. Here, compatibility is preferred over linebreaks.
- Avoid trailing whitespace

Maintaining the list of publications

We want to maintain a list of publications / projects using AMICI. This is located at `documentation/references.md`. This file is created by `documentation/recreate_reference_list.py` based on the bibtex file `documentation/amici_refs.bib`.

After any changes to `documentation/amici_refs.bib`, please run

```
documentation/recreate_reference_list.py
```

(requires biblib)

13.4.2 Code review guide

A guide for reviewing code and having your code reviewed by others.

Everyone

- Don't be too protective of your code
- Accept that, to a large extent, coding decisions are a matter of personal preference
- Don't get personal
- Ask for clarification
- Avoid strong language
- Try to understand your counterpart's perspective
- Clarify how strong you feel about each discussion point

Reviewing code

- If there are no objective advantages, don't force your style on others
- Ask questions instead of making demands
- Assume the author gave his best
- Mind the scope (many things are nice to have, but might be out of scope of the current change - open a new issue)
- The goal is "good enough", not "perfect"
- Be constructive
- You do not always have to request changes

Having your code reviewed

- Don't take it personal - the review is on the code, not on you
- Code reviews take time, appreciate the reviewer's comments
- Assume the reviewer did his best (but might still be wrong)
- Keep code changes small (e.g. separate wide reformatting from actual code changes to facility review)
- If the reviewer does not understand your code, probably many others won't either

Checklist

- [ ] Adherence to project-specific style guide
- [ ] The code is self-explanatory
- [ ] The code is concise / expressive
- [ ] Meaningful identifiers are used
- [ ] Corner-cases are covered, cases not covered fail loudly
- [ ] The code can be expected to scale well (enough)

- [ ] The code is well documented (e.g., input, operation, output), but without trivial comments
- [ ] The code is SOLID
- [ ] New code is added in the most meaningful place (i.e. matches the current architecture)
- [ ] No magic numbers
- [ ] No hard-coded values that should be user inputs
- [ ] No dead code left
- [ ] The changes make sense
- [ ] The changes are not obviously degrading performance
- [ ] There is no duplicated code
- [ ] The API is convenient
- [ ] Code block length and complexity is adequate
- [ ] Spelling okay
- [ ] The code is tested

13.4.3 Continuous integration (CI) and tests

AMICI uses a continuous integration pipeline running via https://github.com/features/actions. This includes the following steps:

- Checking existence and format of documentation
- Static code analysis (http://cppcheck.sourceforge.net/)
- Unit and integration tests
- Memory leak detection

More details are provided in the sections below.

The CI scripts and tests can be found in `tests/` and `scripts/`. Some of the tests are integrated with CMake, see `make help` in the build directory.

C++ unit and integration tests

To run C++ tests, build AMICI with `make` or `scripts/buildAll.sh`, then run `scripts/run-cpp-tests.sh`.

Python unit and integration tests

To run Python tests, run `../scripts/run-python-tests.sh` from anywhere (assumes build directory is `build/`) or run `make python-tests` in your build directory.

SBML Test Suite

We test Python-AMICI SBML support using the test cases from the semantic SBML Test Suite. When making changes to the model import functions, make sure to run these tests.

To run the SBML Test Suite test cases, the easiest way is:

1. Running `scripts/installAmiciSource.sh` which creates a virtual Python environment and performs a development installation of AMICI from the current repository. (This needs to be run only once or after AMICI model generation or C++ changes).
2. Running `scripts/run-SBMLTestsuite.sh`. This will download the test cases if necessary and run them all. A subset of test cases can be selected with an optional argument (e.g. `scripts/run-SBMLTestsuite.sh 1,3-6,8`, to run cases 1, 3, 4, 5, 6 and 8).

   Once the test cases are available locally, for debugging it might be easier to directly use `pytest` with `tests/testSBMLSuite.py`.

Matlab tests (not included in CI pipeline)

To execute the Matlab test suite, run `tests/testModels.m`.

Model simulation integration tests

Many of our integration tests are model simulations. The simulation results obtained from the Python and C++ are compared to results saved in an HDF5 file (`tests/cpp/expectedResults.h5`). Settings and data for the test simulations are also specified in this file.

Note: The C++ code for the models is included in the repository under `models/`. This code is to be updated whenever `amici::Model` changes.

Regenerating C++ code of the test models

Regeneration of the model code has to be done whenever `amici::Model` or the Matlab model import routines change.

This is done with

```
tests/cpp/wrapTestModels.m
```

Note: This is currently only possible from Matlab < R2018a. This should change as soon as 1) all second-order sensitivity code is ported to C++/Python, 2) a non-SBML import exists for Python and 3) support for events has been added for Python.

Regenerating expected results

To update test results, run `make test` in the build directory, replace `tests/cpp/expectedResults.h5` by `tests/cpp/writeResults.h5.bak` [ONLY DO THIS AFTER TRIPLE CHECKING CORRECTNESS OF RESULTS] Before replacing the test results, confirm that only expected datasets have changed, e.g. using

```
h5diff -v --relative 1e-8 tests/cpp/expectedResults.h5 tests/cpp/writeResults.h5.bak |␣
↪less
```

Adding/Updating tests

To add new tests add a new corresponding python script (see, e.g., `./tests/generateTestConfig/example_dirac.py`) and add it to and run `tests/generateTestConfigurationForExamples.sh`. Then regenerate the expected test results (see above).

13.4.4 Debugging AMICI

This document contains some information on how to debug any issues in AMICI, in particular for C++ Python extensions.

Caveman debugging / printf-debugging

The simplest approach may often be adding some print-statements to the code, as this does not require any special tools.

Note that after each change of the C++ files, the AMICI extension *as well as the model extension* (if any model functions are called), need to be recompiled. The simplest and safest approach would be re-installation of the amici package and re-import of the model. As this can be very time-consuming, the following shortcut is possible, assuming you are using a development installation (`pip install -e ...`):

```
# rebuild the amici base extension, from within the amici root directory
# (note that this only recompiles the amici source files, NOT third-party
# dependencies such as sundials):
cd python/sdist/
python setup.py build_ext --build-lib .

# rebuild the model, from within the model package directory:
python setup.py build_ext --force --build-lib .
```

Note: Be careful when working interactively, Python may not pick up any changes in already imported modules. The safest is to start a new Python process after any changes.

Using a proper debugger

Debugging with with [gdb](https://www.sourceware.org/gdb/) is most convenient with a minimal reproducible example that is directly invoked from gdb. For example:

```
# start gdb
gdb --args python -m pytest ../tests/test_sbml_import.py::test_nosensi

# inside gdb, set a meaningful breakpoint and launch:
break amici::runAmiciSimulation
run
# ... (see one of the many gdb tutorials)
```

Alternative, gdb can attach to a running process by passing the `--pid` argument.

For many users, it may be more convenient to use gdb via some graphical user interface as provided by various C++ IDEs.

Note: For better debugging experience, but at the cost of runtime performance, consider building the amici and model extension with environment variable `ENABLE_AMICI_DEBUGGING=TRUE`. This will include debugging symbols and disable compiler optimizations.

CHAPTER

FOURTEEN

HANDLING OF DISCONTINUITIES

This document provides guidance and rationale on the implementation of events in AMICI. Events include any discontinuities in the right hand side of the differential equation. There are three types of discontinuities:

- **Solution Jump Discontinuities** can be created by SBML events or delta functions in the right hand side.
- **Right-Hand-Side Jump Discontinuities** result in removable discontinuities in the solution and can be created by Piecewise, Heaviside functions and other logical operators in the right hand side.
- **Right-Hand-Side Removable Discontinuities** do not lead to discontinuities in the solution, but may lead to discontinuous higher order temporal derivatives and can be created by functions such as max or min in the right hand side.

14.1 Mathematical Considerations

A detailed mathematical description of the required sensitivity formulas is provided in

- Fröhlich, F., Theis, F. J., Rädler, J. O., & Hasenauer, J. (2017). Parameter estimation for dynamical systems with discrete events and logical operations. Bioinformatics, 33(7), 1049-1056. doi:10.1093/bioinformatics/btw764.

14.2 Algorithmic Considerations

14.2.1 Solution Jump Discontinuities

SUNDIALS by itself does not support solution jump discontinuities. We implement support by accessing private SUNDIALS API in `amici::Solver::resetState()`, `amici::Solver::reInitPostProcess()` and *`amici::Solver::reInitPostProcessB()`*. These functions reset interval variables to initial values to simulate a fresh integration start, but keep/update the solution history, which is important for adjoint solutions.

14.2.2 Right-Hand-Side Jump Discontinuities

In principle these discontinuities do not need any special treatment, but empirically, the solver may overstep or completely ignore the discontinuity, leading to poor solution quality. This is particularly problematic when step size is large and changes in step size, which can be caused by parameter changes, inclusion of forward sensitivities or during backward solves, may alter solutions in unexpected ways. Accordingly, finite difference approximations, forward sensitivities as well as adjoint sensitivities will yield poor derivative approximations.

To address these issues, we use the built-in rootfinding functionality in SUNDIALS, which pauses the solver at the locations of discontinuities and avoids overstepping or ignoring of discontinuities.

Another difficulty comes with the evaluation of Heaviside functions. After or during processing of discontinuities, Heaviside functions need to be evaluated at the left and right hand limit of discontinuities. This is challenging as the solver may slightly over- or understep the discontinuity timepoint by a small epsilon and limits have to be correctly computed in both forward and backward passes.

To address this issue, AMICI uses a vector of Heaviside helper variables h that keeps track of the values of the Heaviside functions that have the respective root function as argument. These will be automatically updated during events and take either 0 or 1 values as appropriate pre/post event limits.

In order to fully support SBML events and Piecewise functions, AMICI uses the SUNDIALS functionality to only track zero crossings from negative to positive. Accordingly, two root functions are necessary to keep track of Heaviside functions and two Heaviside function helper variables will be created, where one corresponds to the value of *Heaviside(…)* and one to the value of *1-Heaviside(…)*. To ensure that Heaviside functions are correctly evaluated at the beginning of the simulation, Heaviside functions are implement as unit steps that evaluate to *1* at *0*. The arguments of Heaviside functions are normalized such that respective properties of Piecewise functions are conserved for the first Heaviside function variable. Accordingly, the value of of the second helper variable is incorrect when simulation starts when the respective Heaviside function evaluates to zero at initialization and should generally not be used.

14.2.3 Right-Hand-Side Removable Discontinuities

Removable discontinuities do not require any special treatment. Numerically, this may be advantageous, but is currently not implemented.

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V