

---

# **AMICI Documentation**

***Release 0.11.14***

**The AMICI developers**

**Mar 16, 2021**



# ABOUT

<b>1</b>	<b>About AMICI</b>	<b>3</b>
1.1	Features . . . . .	3
1.2	Interfaces & workflow . . . . .	4
<b>2</b>	<b>Availability</b>	<b>5</b>
2.1	Source code . . . . .	5
2.2	Python package . . . . .	5
2.3	Installation instructions . . . . .	5
<b>3</b>	<b>License conditions</b>	<b>7</b>
3.1	AMICI . . . . .	7
3.2	AMICI Logo . . . . .	7
3.3	Dependencies . . . . .	8
<b>4</b>	<b>How to cite AMICI</b>	<b>9</b>
<b>5</b>	<b>References</b>	<b>11</b>
<b>6</b>	<b>Background</b>	<b>13</b>
6.1	Publications on various features of AMICI . . . . .	13
6.2	Third-Party numerical algorithms used by AMICI . . . . .	13
<b>7</b>	<b>Changelog</b>	<b>15</b>
7.1	v0.11.14 (2021-03-16) . . . . .	15
7.2	v0.11.13 (2021-02-20) . . . . .	16
7.3	v0.11.12 (2021-01-26) . . . . .	17
7.4	v0.11.11 (2020-12-15) . . . . .	17
7.5	v0.11.10 (2020-11-30) . . . . .	18
7.6	v0.11.9 (2020-11-29) . . . . .	18
7.7	v0.11.8 (2020-10-21) . . . . .	18
7.8	v0.11.7 (2020-09-22) . . . . .	19
7.9	v0.11.6 (2020-08-20) . . . . .	20
7.10	v0.11.5 (2020-08-07) . . . . .	20
7.11	v0.11.4 (2020-08-06) . . . . .	21
7.12	v0.11.3 (2020-08-06) . . . . .	21
7.13	v0.11.2 (2020-07-17) . . . . .	22
7.14	v0.11.1 (2020-06-05) . . . . .	22
7.15	v0.11.0 (2020-05-10) . . . . .	23
7.16	v0.10.21 (2020-04-04) . . . . .	24
7.17	v0.10.20 (2020-03-18) . . . . .	24
7.18	v0.10.19 (2020-02-13) . . . . .	25

7.19	v0.10.18 (2020-02-11)	25
7.20	v0.10.17 (2020-01-15)	26
7.21	v0.10.16 (2019-12-11)	26
7.22	v0.10.15 (2019-12-03)	26
7.23	v0.10.14 (2019-12-02)	26
7.24	v0.10.13 (2019-10-09)	27
7.25	v0.10.12 (2019-09-28)	27
7.26	v0.10.11 (2019-08-31)	28
7.27	v0.10.10 (2019-08-07)	28
7.28	v0.10.9 (2019-07-24)	28
7.29	v0.10.8 (2019-07-24)	29
7.30	v0.10.7 (2019-05-01)	29
7.31	v0.10.6 (2019-04-19)	29
7.32	v0.10.5 (2019-04-08)	30
7.33	v0.10.4 (2019-03-21)	30
7.34	v0.10.3 (2019-03-13)	30
7.35	v0.10.2 (2019-03-07)	31
7.36	v0.10.1 (2019-03-04)	31
7.37	v0.10.0 (2019-03-01)	31
7.38	v0.9.5 (2019-02-26)	31
7.39	v0.9.4 (2019-02-11)	32
7.40	v0.9.3 (2019-02-07)	32
7.41	v0.9.2 (2019-01-30)	32
7.42	v0.9.1 (2019-01-21)	33
7.43	v0.9.0 (2019-01-18)	33
7.44	v0.8.2 (2019-01-07)	33
7.45	v0.8.1 (2018-11-25)	34
7.46	v0.8.0 (2018-11-25)	34
7.47	v0.7.13 (2018-11-18)	34
7.48	v0.7.12 (2018-11-17)	34
7.49	v0.7.11 (2018-10-15)	35
7.50	v0.7.10 (2018-08-29)	35
7.51	v0.7.9 (2018-08-24)	35
7.52	v0.7.8 (2018-08-19)	35
7.53	v0.7.7 (2018-08-17)	35
7.54	v0.7.6 (2018-08-13)	35
7.55	v0.7.5 (2018-07-30)	36
7.56	v0.7.4 (2018-07-27)	36
7.57	v0.7.3 (2018-07-13)	36
7.58	v0.7.2 (2018-07-03)	37
7.59	v0.7.1 (2018-06-12)	37
7.60	v0.7.0 (2018-06-09)	37
7.61	v0.6.0 (2018-05-22)	37
7.62	v0.5.0 (2018-03-15)	38
7.63	v0.4.0 (2017-05-15)	38
7.64	v0.3.0 (2016-09-05)	38
7.65	0.2.1 (2016-05-09)	38
7.66	v0.2 (2016-03-17)	39
7.67	v0.1 (2015-11-05)	39
<b>8</b>	<b>Glossary</b>	<b>41</b>
<b>9</b>	<b>Contributing</b>	<b>43</b>
9.1	Contributing to AMICI	43

9.2	Code of Conduct . . . . .	43
<b>10</b>	<b>Python interface</b>	<b>45</b>
10.1	Installing the AMICI Python package . . . . .	45
10.2	Using AMICI's Python interface . . . . .	51
10.3	FAQ . . . . .	125
10.4	AMICI Python API . . . . .	125
<b>11</b>	<b>C++ interface</b>	<b>255</b>
11.1	Building the C++ library . . . . .	255
11.2	Using AMICI's C++ interface . . . . .	256
11.3	AMICI C++ API . . . . .	258
<b>12</b>	<b>Matlab interface</b>	<b>519</b>
12.1	Installing the AMICI MATLAB toolbox . . . . .	519
12.2	Using AMICI's MATLAB interface . . . . .	519
12.3	FAQ . . . . .	526
12.4	AMICI Matlab API . . . . .	527
<b>13</b>	<b>AMICI developer's guide</b>	<b>557</b>
13.1	Branches / releases . . . . .	557
13.2	When starting to work on some issue . . . . .	557
13.3	Code contributions . . . . .	557
13.4	Further topics . . . . .	559
<b>14</b>	<b>Handling of Discontinuities</b>	<b>565</b>
14.1	Mathematical Considerations . . . . .	565
14.2	Algorithmic Considerations . . . . .	565
<b>15</b>	<b>Indices and tables</b>	<b>567</b>
	<b>Python Module Index</b>	<b>569</b>
	<b>Index</b>	<b>571</b>



Version: 0.11.14

Source code: <https://github.com/AMICI-dev/amici>





## ABOUT AMICI

AMICI provides a multi-language (Python, C++, Matlab) interface to the *SUNDIALS* solvers *CVODES* (for *ODEs*) and *IDAS* (for *DAEs*). 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.

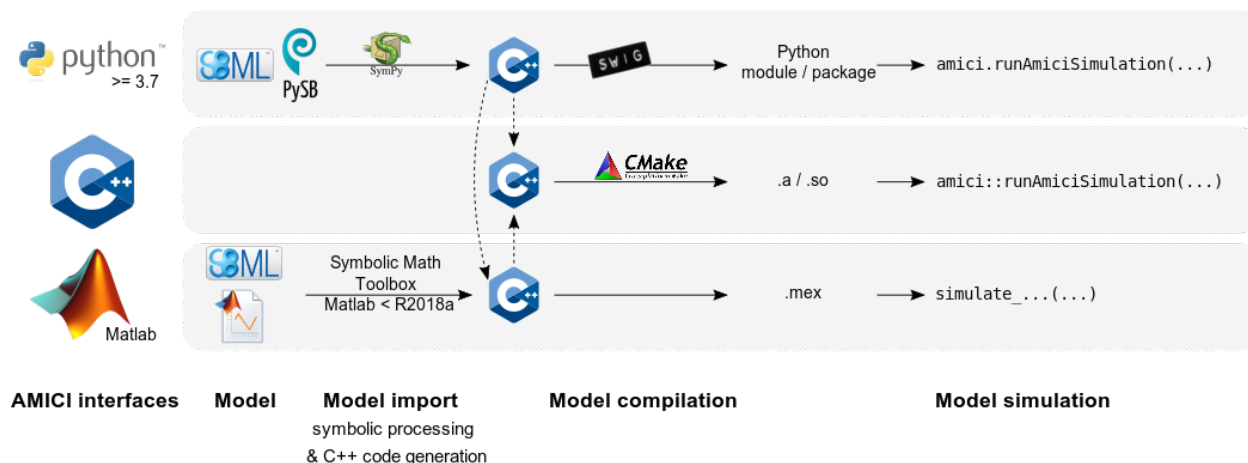
The interface was designed to provide routines for efficient gradient computation in parameter estimation of biochemical reaction models but it is also applicable to a wider range of differential equation constrained optimization problems.

## 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
<i>SBML</i> import	yes ( <i>details</i> )	no	yes (<=R2017b)
<i>PySB</i> import	yes	no	no
<i>DAE</i> 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
<i>preequilibration</i>	yes	yes	yes
<i>presimulation</i>	yes	yes	no

## 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*



## LICENSE CONDITIONS

### 3.1 AMICI

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

Copyright (c) 2015-2020, Fabian Fröhlich, Jan Hasenauer, Daniel Weindl and Paul Stapor All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### 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`

## 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., 2020. AMICI: High-Performance Sensitivity Analysis for Large Ordinary Differential Equation Models. arXiv preprint [arXiv:2012.09122](#).

```
@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={arXiv preprint arXiv:2012.09122},  
  year={2020}  
}
```

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







## REFERENCES

List of publications using AMICI. Total number is 51.

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



## 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1093/bioinformatics/bty230).

---

**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](https://crd-legacy.lbl.gov/~xiaoye/SuperLU/#superlu_mt)). SuperLU\_MT is optional and is so far only available from the C++ interface.

## CHANGELOG

### 7.1 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 v0.11.13 (2021-02-20)

Breaking changes:

- AMICI requires Python $\geq 3.7$
- 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
- ...

## 7.3 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)

## 7.4 v0.11.11 (2020-12-15)

### 7.4.1 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)

### 7.4.2 C++

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

### 7.4.3 CI

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

### 7.4.4 Docker

- Update ubuntu to 20.04

## 7.5 v0.11.10 (2020-11-30)

Bugfix release that restores compatibility with sympy 1.7

## 7.6 v0.11.9 (2020-11-29)

### 7.6.1 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 )

### 7.6.2 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)

### 7.6.3 Doc

- Update references (#1331, #1336)

### 7.6.4 CI

- Update OpenBLAS for windows (#1334)

## 7.7 v0.11.8 (2020-10-21)

### 7.7.1 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)



### 7.7.2 C++

- Make HDF5 an optional dependency (#1285)

### 7.7.3 Doc

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

### 7.7.4 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)

## 7.8 v0.11.7 (2020-09-22)

### 7.8.1 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)

### 7.8.2 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)

### 7.8.3 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)

## 7.8.4 CI

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

## 7.9 v0.11.6 (2020-08-20)

### 7.9.1 Python

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

### 7.9.2 C++

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

### 7.9.3 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)

### 7.9.4 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)

## 7.10 v0.11.5 (2020-08-07)

### 7.10.1 General

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

### 7.10.2 Python

- Fix bug for energyPySB models (#1191)

### 7.10.3 CI

- Fix release deployment (#1189)

## 7.11 v0.11.4 (2020-08-06)

### 7.11.1 Python

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

### 7.11.2 CI

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

## 7.12 v0.11.3 (2020-08-06)

### 7.12.1 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)

### 7.12.2 C++

- Refactoring (#1162, #1163)

### 7.12.3 CI

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

## 7.13 v0.11.2 (2020-07-17)

### 7.13.1 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)

### 7.13.2 C++

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

### 7.13.3 CI

- Store build artifacts from github actions (#1138)

## 7.14 v0.11.1 (2020-06-05)

### 7.14.1 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)

### 7.14.2 Matlab

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

### 7.14.3 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)

### 7.14.4 CI

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

## 7.15 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 equilibration, presimulation and adjoint sensitivity analysis**

CI:

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

## 7.16 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)

## 7.17 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 environment 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)

## 7.18 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

## 7.19 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

## 7.20 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)

## 7.21 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)

## 7.22 v0.10.15 (2019-12-03)

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

No other changes.

## 7.23 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

## 7.24 v0.10.13 (2019-10-09)

- BREAKING CHANGE: Renaming {getset}tNewtonPreequilibration to {getset}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

## 7.25 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)

## 7.26 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)

## 7.27 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

Details: \* 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)

## 7.28 v0.10.9 (2019-07-24)

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

## 7.29 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

## 7.30 v0.10.7 (2019-05-01)

Python

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

## 7.31 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

## 7.32 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 PEBtab import to allow for different noise models

## 7.33 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

## 7.34 v0.10.3 (2019-03-13)

Features / improvements:

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

Bugfixes:

- fixed symbolic processing for unreleased pysb features

## 7.35 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`

## 7.36 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

## 7.37 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.

## 7.38 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

## 7.39 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)

## 7.40 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`

## 7.41 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

## 7.42 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

## 7.43 v0.9.0 (2019-01-18)

Features / improvements:

- Allow computation and application of conservation laws for pysb importet models. This enables use of NewtonSolver 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

## 7.44 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

## 7.45 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

## 7.46 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)

## 7.47 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.

## 7.48 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



## 7.49 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 native 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)

## 7.50 v0.7.10 (2018-08-29)

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

## 7.51 v0.7.9 (2018-08-24)

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

## 7.52 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

## 7.53 v0.7.7 (2018-08-17)

Fixes a couple of bugs just introduced in v0.7.6

## 7.54 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)

## 7.55 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)

## 7.56 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, ...

## 7.57 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/>

## 7.58 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

## 7.59 v0.7.1 (2018-06-12)

Features:

- Allow specifying sigma-parameters from Python interface

Major bugfixes:

- Fix dsigma\_y/dp and downstream sensitivity errors

## 7.60 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.

## 7.61 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

## 7.62 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 maintenance
- Substantially extended testing in continuous integration to assure code quality

## 7.63 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

## 7.64 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

## 7.65 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.

## 7.66 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

## 7.67 v0.1 (2015-11-05)

This is the initial release of the toolbox



## GLOSSARY

**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** *SBML* 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.





## 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](mailto: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](https://www.contributor-covenant.org/version/1/4/code-of-conduct.html), 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>

## 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 $\geq$ 3.7
- *SWIG* $\geq$ 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 `msvcrl40.dll` with `[...] x86_64-w64-mingw32/bin/ld.exe: cannot find -lmsvcrl40`. This is not required for amici, so in `cygwinccompiler.py` return `['msvcrl40']` 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:

C:\Users\travis\build\AMICI\scripts\compileBLAS.cmd

so that it matches your directory structure. This will download OpenBLAS and compile it, creating

`C:\BLASlib\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-v0.3.10\OpenBLAS-0.3.10"
```

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-v0.3.
↪10\OpenBLAS-0.3.10", [System.EnvironmentVariableTarget]::User)
[System.Environment]::SetEnvironmentVariable("BLAS_CFLAGS", "-IC:\BLAS\OpenBLAS-v0.3.
↪10\OpenBLAS-0.3.10", [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.`

This can be tested using the “where” command. For example

`where openblas.dll`

should return

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

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.

## 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 be bothered with any unnecessary dependencies, you can run

```
pip3 install --install-option --no-libs 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 <i>SWIG</i> 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	<b>Compiler flags for, e.g. BLAS</b> 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 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 996 out of the 1780 (~56%) 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
- 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()`.

BioNetGen and Kappa models can be imported into AMICI using PySB.

## 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 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.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  $\sigma$  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](https://github.com/PEtab-dev/PEtab). Another possibility is using SBML's `AssignmentRules` [http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml\\_1\\_1\\_rule.html](http://sbml.org/Software/libSBML/5.13.0/docs/python-api/classlibsbml_1_1_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'}
}
```

## $\sigma$ 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)

2020-04-24 12:00:42.746 - amici.sbml_import - INFO - Finished processing SBML_
↳ parameters (5.40E-04s)
2020-04-24 12:00:42.748 - amici.sbml_import - INFO - Finished processing SBML_
↳ compartments (4.23E-04s)
2020-04-24 12:00:42.809 - amici.sbml_import - INFO - Finished processing SBML species_
↳ (6.01E-02s)
```

(continues on next page)

(continued from previous page)

```

2020-04-24 12:00:42.819 - amici.sbml_import - INFO - Finished processing SBML_
↳reactions (1.00E-02s)
2020-04-24 12:00:42.847 - amici.sbml_import - INFO - Finished processing SBML rules
↳ (2.74E-02s)
2020-04-24 12:00:42.928 - amici.sbml_import - INFO - Finished processing SBML_
↳observables (7.94E-02s)
2020-04-24 12:00:43.022 - amici.ode_export - INFO - Finished writing J.cpp
↳ (4.21E-02s)
2020-04-24 12:00:43.032 - amici.ode_export - INFO - Finished writing JB.cpp
↳ (8.60E-03s)
2020-04-24 12:00:43.036 - amici.ode_export - INFO - Finished writing JDiag.cpp
↳ (3.54E-03s)
2020-04-24 12:00:43.045 - amici.ode_export - INFO - Finished writing JSparse.cpp
↳ (9.11E-03s)
2020-04-24 12:00:43.056 - amici.ode_export - INFO - Finished writing JSparseB.cpp
↳ (8.67E-03s)
2020-04-24 12:00:43.079 - amici.ode_export - INFO - Finished writing Jy.cpp
↳ (2.16E-02s)
2020-04-24 12:00:43.231 - amici.ode_export - INFO - Finished writing dJydsigmay.cpp
↳ (1.51E-01s)
2020-04-24 12:00:43.281 - amici.ode_export - INFO - Finished writing dJydy.cpp
↳ (4.83E-02s)
2020-04-24 12:00:43.292 - amici.ode_export - INFO - Finished writing dwdp.cpp
↳ (9.91E-03s)
2020-04-24 12:00:43.296 - amici.ode_export - INFO - Finished writing dwdx.cpp
↳ (2.48E-03s)
2020-04-24 12:00:43.305 - amici.ode_export - INFO - Finished writing dxdotdw.cpp
↳ (7.81E-03s)
2020-04-24 12:00:43.315 - amici.ode_export - INFO - Finished writing dxdotdp_explicit.
↳cpp (7.82E-03s)
2020-04-24 12:00:43.344 - amici.ode_export - INFO - Finished writing dydx.cpp
↳ (1.94E-02s)
2020-04-24 12:00:43.358 - amici.ode_export - INFO - Finished writing dydp.cpp
↳ (1.23E-02s)
2020-04-24 12:00:43.364 - amici.ode_export - INFO - Finished writing dsigmaydp.cpp
↳ (5.51E-03s)
2020-04-24 12:00:43.368 - amici.ode_export - INFO - Finished writing sigmay.cpp
↳ (3.35E-03s)
2020-04-24 12:00:43.373 - amici.ode_export - INFO - Finished writing w.cpp
↳ (3.53E-03s)
2020-04-24 12:00:43.376 - amici.ode_export - INFO - Finished writing x0.cpp
↳ (2.23E-03s)
2020-04-24 12:00:43.378 - amici.ode_export - INFO - Finished writing x0_
↳fixedParameters.cpp (1.14E-03s)
2020-04-24 12:00:43.382 - amici.ode_export - INFO - Finished writing sx0.cpp
↳ (3.75E-03s)
2020-04-24 12:00:43.387 - amici.ode_export - INFO - Finished writing sx0_
↳fixedParameters.cpp (4.08E-03s)
2020-04-24 12:00:43.397 - amici.ode_export - INFO - Finished writing xdot.cpp
↳ (9.57E-03s)
2020-04-24 12:00:43.401 - amici.ode_export - INFO - Finished writing y.cpp
↳ (3.07E-03s)
2020-04-24 12:00:43.405 - amici.ode_export - INFO - Finished writing x_rdata.cpp
↳ (2.53E-03s)
2020-04-24 12:00:43.407 - amici.ode_export - INFO - Finished writing total_cl.cpp
↳ (1.34E-03s)
2020-04-24 12:00:43.412 - amici.ode_export - INFO - Finished writing x_solver.cpp
↳ (2.74E-03s)

```

(continues on next page)



(continued from previous page)

```
2020-04-24 12:00:43.437 - amici.ode_export - INFO - Finished generating cpp code
↳ (4.67E-01s)
2020-04-24 12:00:58.500 - amici.ode_export - INFO - Finished compiling cpp code
↳ (1.51E+01s)
```

## 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.

```
[11]: # 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)
```

```
[12]: 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:

```
[13]: #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]
 [0.51103766 0.57935661 0.05012037]
 [0.50470029 0.56191592 0.04793052]
 [0.49910666 0.54673518 0.0460508 ]
 [0.49416809 0.53349812 0.04443205]
 [0.48980688 0.52193768 0.04303399]
 [0.48595476 0.51182733 0.04182339]
 [0.48255177 0.50297414 0.04077267]
 [0.47954512 0.4952132  0.03985882]
 [0.47688834 0.48840305 0.03906254]
 [0.47454049 0.482422  0.03836756]
 [0.47246548 0.47716503 0.0377601 ]
 [0.47063147 0.47254128 0.03722844]
 [0.46901037 0.46847203 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]
 [0.4602552  0.44678168 0.03431212]
 [0.45983714 0.44575804 0.03419784]
 [0.45946749 0.44485387 0.03409701]
 [0.45914065 0.44405513 0.03400802]
 [0.45885166 0.44334946 0.03392945]
 [0.45859614 0.44272594 0.03386009]
 [0.45837021 0.44217496 0.03379883]
 [0.45817043 0.44168804 0.03374472]
 [0.45799379 0.44125771 0.03369693]
 [0.45783759 0.44087737 0.03365471]
 [0.45769949 0.44054119 0.0336174 ]
 [0.45757737 0.44024404 0.03358444]
```

(continues on next page)

(continued from previous page)

```

[0.45746939 0.43998136 0.0335553 ]
[0.45737391 0.43974916 0.03352956]
[0.45728948 0.43954388 0.03350681]
[0.45721483 0.43936241 0.0334867 ]
[0.45714882 0.43920197 0.03346892]
[0.45709045 0.43906014 0.03345321]
[0.45703884 0.43893473 0.03343931]
[0.4569932 0.43882386 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 0.43843487 0.03338397]
[0.45681123 0.4383819 0.0333781 ]
[0.45679194 0.43833507 0.03337292]
[0.45677488 0.43829366 0.03336833]
[0.4567598 0.43825704 0.03336428]
[0.45674647 0.43822467 0.0333607 ]
[0.45673467 0.43819604 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]
[0.51103766 0.57935661 0.05012037 1.02207533 3.57935661 0.51103766]
[0.50470029 0.56191592 0.04793052 1.00940059 3.56191592 0.50470029]
[0.49910666 0.54673518 0.0460508 0.99821331 3.54673518 0.49910666]
[0.49416809 0.53349812 0.04443205 0.98833618 3.53349812 0.49416809]
[0.48980688 0.52193768 0.04303399 0.97961375 3.52193768 0.48980688]
[0.48595476 0.51182733 0.04182339 0.97190952 3.51182733 0.48595476]
[0.48255177 0.50297414 0.04077267 0.96510354 3.50297414 0.48255177]
[0.47954512 0.4952132 0.03985882 0.95909023 3.4952132 0.47954512]
[0.47688834 0.48840305 0.03906254 0.95377668 3.48840305 0.47688834]
[0.47454049 0.482422 0.03836756 0.94908098 3.482422 0.47454049]
[0.47246548 0.47716503 0.0377601 0.94493096 3.47716503 0.47246548]
[0.47063147 0.47254128 0.03722844 0.94126294 3.47254128 0.47063147]
[0.46901037 0.46847203 0.03676259 0.93802074 3.46847203 0.46901037]
[0.46757739 0.46488881 0.03635397 0.93515479 3.46488881 0.46757739]
[0.46631065 0.46173207 0.03599523 0.93262131 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 0.92510361 3.45242457 0.4625518 ]
[0.46186768 0.45074016 0.03475519 0.92373536 3.45074016 0.46186768]
[0.46126282 0.44925337 0.03458856 0.92252565 3.44925337 0.46126282]
[0.46072804 0.44794075 0.03444166 0.92145608 3.44794075 0.46072804]
[0.4602552 0.44678168 0.03431212 0.92051041 3.44678168 0.4602552 ]
[0.45983714 0.44575804 0.03419784 0.91967427 3.44575804 0.45983714]
[0.45946749 0.44485387 0.03409701 0.91893498 3.44485387 0.45946749]
[0.45914065 0.44405513 0.03400802 0.9182813 3.44405513 0.45914065]

```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continues on next page)

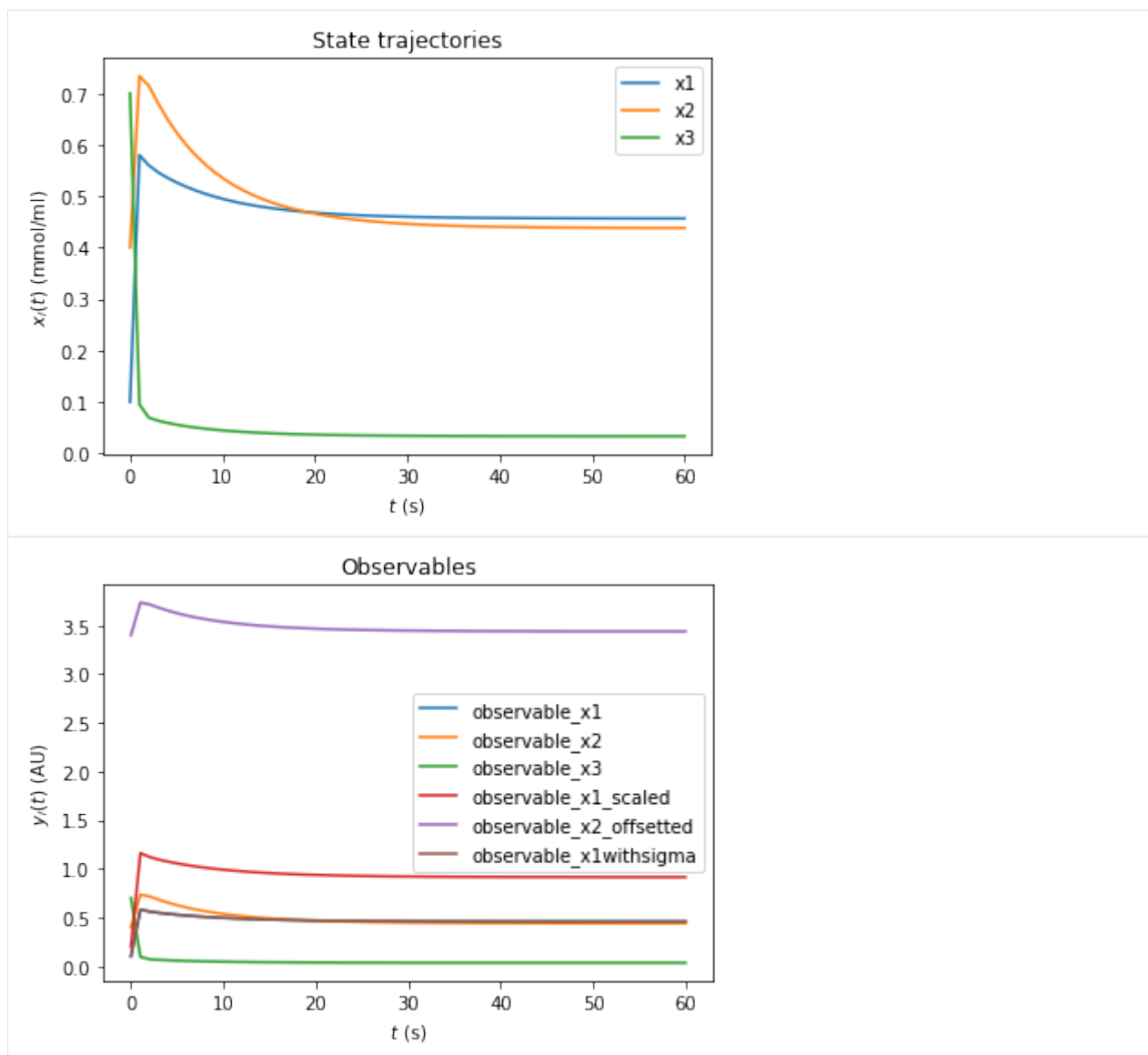
## 10.2. Using AMICI's Python interface 63

(continues on next page)

[illegible]

```
print(model.getParameters())
```

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



## 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.

```
[16]: # 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 -86.872839

```

## 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, steadstates 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.

```

[17]: 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')
          ax.plot(tols, llh_errs, 'b-', label='llh')
          ax.set_xscale('log')
          ax.set_yscale('log')
          ax.set_xlabel(f'{tolname} tolerance')

```

(continues on next page)

(continued from previous page)

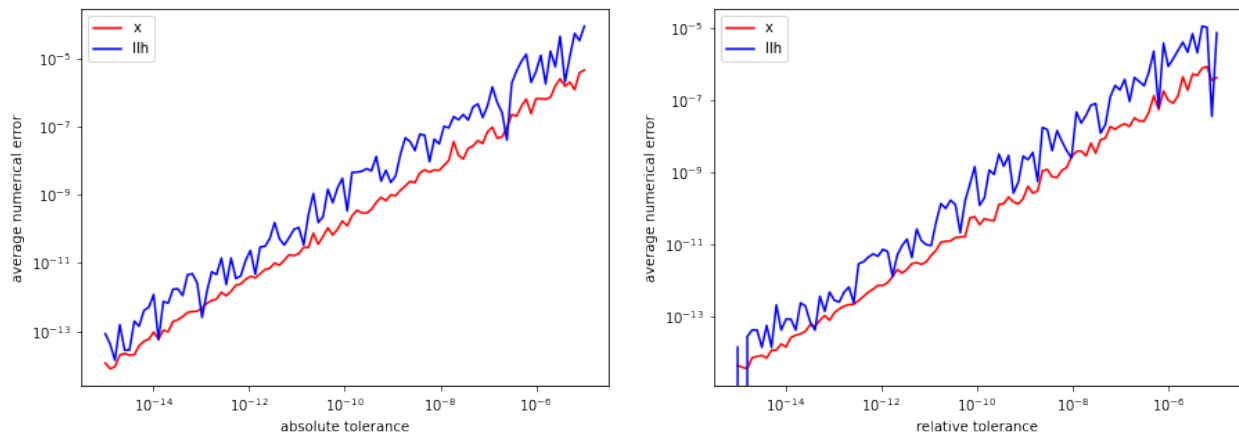
```

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

```

[18]: 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

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]]]

[[-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]

```

(continues on next page)

(continued from previous page)

```

[ 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]
 [ 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.]]
    sx_ss:  [[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]
[nan nan nan]]

```

(continues on next page)

(continued from previous page)

```

    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]]
    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  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
 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]]]

```

(continues on next page)

(continued from previous page)

```

[ 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]
[ 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

```

(continues on next page)

(continued from previous page)

```

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
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]]

```

(continues on next page)

(continued from previous page)

```

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]
[ 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. 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.]

```

(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. 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. 0.]

```

(continues on next page)

(continued from previous page)

[illegible]

(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. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 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

```

[19]: # Set model options
model = model_module.getModel()
p_orig = np.array(model.getParameters())
p_orig[list(model.getParameterIds()).index('observable_xlwithsigma_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_
↳ xlwithsigma'))

# enable sensitivities
solver.setSensitivityOrder(amici.SensitivityOrder.first) # First-order ...

```

(continues on next page)

(continued from previous page)

```

solver.setSensitivityMethod(amici.SensitivityMethod.adjoint)          # ... adjoint_
↪sensitivities
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: -1782.069948
Gradient: [ 1.88753631e+01  2.34371294e+01 -4.34446460e+01 -5.65142354e+00
 -1.05291555e+02  5.87696815e-01 -2.1600777e+00  3.30558521e+04]

```

## Finite differences gradient check

Compare AMICI-computed gradient with finite differences

```

[20]: 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)

    model.setParameters(old_parameters)

```

(continues on next page)

(continued from previous page)

```

    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: 49.727388

sllh: p[0]: |error|_2: 0.015277
sllh: p[1]: |error|_2: 0.013750
sllh: p[2]: |error|_2: 0.012565
sllh: p[3]: |error|_2: 0.015211
sllh: p[4]: |error|_2: 0.047226
sllh: p[5]: |error|_2: 0.000280
sllh: p[6]: |error|_2: 0.001050
sllh: p[7]: |error|_2: 49.727399

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

```

(continues on next page)

(continued from previous page)

```

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

```

```

[21]: 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)

          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}')

```

(continues on next page)

(continued from previous page)

```

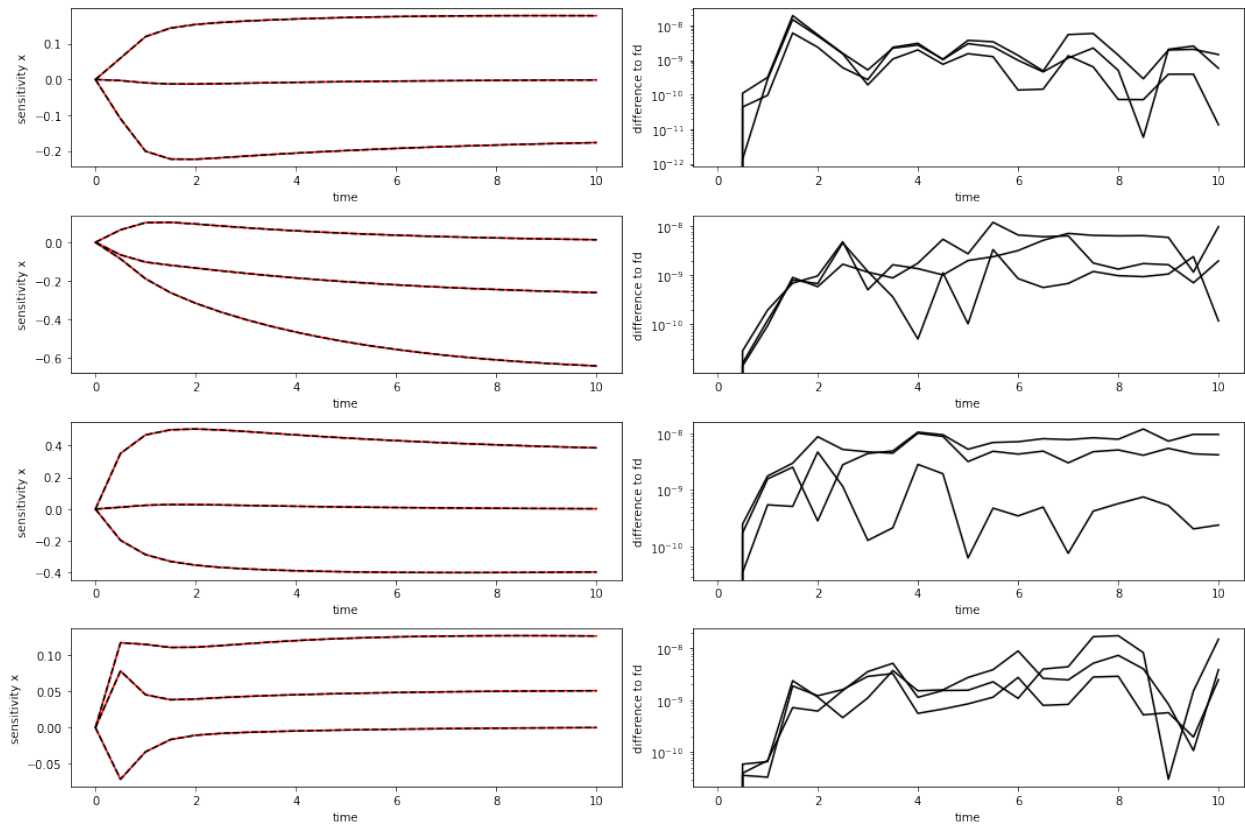
axes[ip,0].set_xlabel('time')

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')

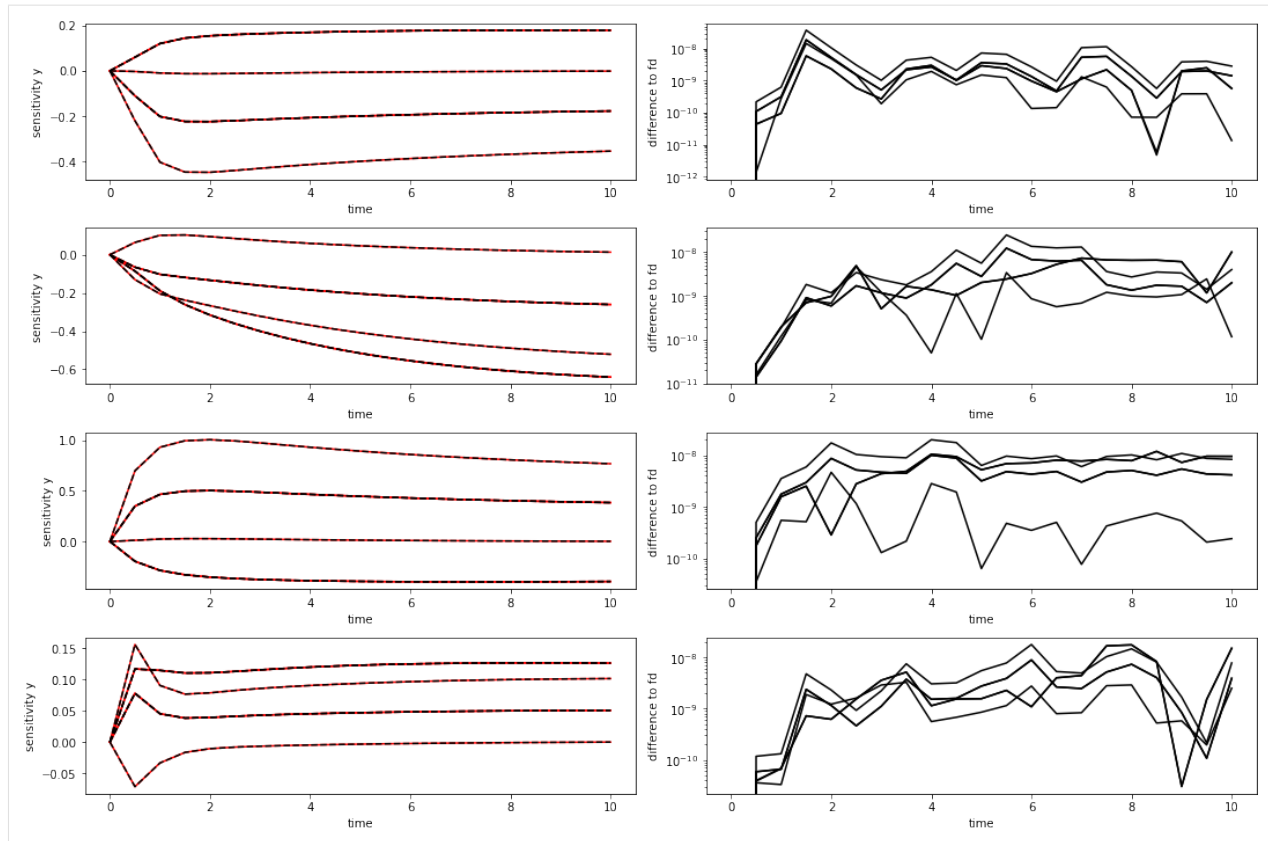
plt.tight_layout()
plt.show()

```

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



```
[23]: 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

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

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

```
[25]:
```

	time	datatype	t_presim	k0	k0_preeq	k0_presim	observable_x1	\
0	0.0	data	0.0	1.0	NaN	NaN	-1.818394	
1	0.5	data	0.0	1.0	NaN	NaN	-0.286706	
2	1.0	data	0.0	1.0	NaN	NaN	-0.112492	
3	1.5	data	0.0	1.0	NaN	NaN	0.836740	
4	2.0	data	0.0	1.0	NaN	NaN	0.233868	
5	2.5	data	0.0	1.0	NaN	NaN	0.915582	
6	3.0	data	0.0	1.0	NaN	NaN	0.919766	
7	3.5	data	0.0	1.0	NaN	NaN	0.166633	
8	4.0	data	0.0	1.0	NaN	NaN	0.638798	
9	4.5	data	0.0	1.0	NaN	NaN	0.187986	
10	5.0	data	0.0	1.0	NaN	NaN	0.405781	
11	5.5	data	0.0	1.0	NaN	NaN	-0.249370	
12	6.0	data	0.0	1.0	NaN	NaN	-1.900392	

(continues on next page)



(continued from previous page)

13	6.5	data	0.0	1.0	NaN	NaN	0.556327
14	7.0	data	0.0	1.0	NaN	NaN	-0.022463
15	7.5	data	0.0	1.0	NaN	NaN	-0.621709
16	8.0	data	0.0	1.0	NaN	NaN	1.328609
17	8.5	data	0.0	1.0	NaN	NaN	0.886509
18	9.0	data	0.0	1.0	NaN	NaN	1.452422
19	9.5	data	0.0	1.0	NaN	NaN	1.493436
20	10.0	data	0.0	1.0	NaN	NaN	1.828229

	observable_x2	observable_x3	observable_x1_scaled	\
0	0.323750	-1.130396	-0.235845	
1	1.481899	-0.190653	0.384839	
2	1.481440	0.010955	1.837763	
3	1.890309	0.855302	1.396028	
4	1.963070	-0.356165	1.342368	
5	-0.012158	0.380968	1.772030	
6	0.240276	0.012171	0.072701	
7	1.250029	-0.248000	2.575797	
8	1.308206	-1.515860	1.361921	
9	-0.482633	0.326894	1.982838	
10	1.092347	-0.468315	1.235247	
11	0.873713	-0.119955	1.518338	
12	1.966896	-2.117369	-0.391538	
13	0.466760	1.585724	2.304242	
14	2.221990	-0.845991	-0.259019	
15	-0.471851	-1.077408	2.055719	
16	1.048814	1.903001	0.758393	
17	0.279781	-0.437013	1.875357	
18	1.597938	-1.280079	0.185404	
19	1.974988	-0.059626	0.466490	
20	0.864279	0.389782	-0.399825	

	observable_x2_offsetted	observable_x1withsigma	observable_x1_std	\
0	3.003267	-1.249817	1.0	
1	3.928552	0.322567	1.0	
2	3.394742	0.547032	1.0	
3	3.634490	1.340781	1.0	
4	3.063575	-1.318266	1.0	
5	2.638614	1.998935	1.0	
6	3.251369	-0.546210	1.0	
7	5.000227	3.267072	1.0	
8	1.807606	0.022304	1.0	
9	2.642325	-0.265486	1.0	
10	1.189159	-1.747961	1.0	
11	6.376749	2.518514	1.0	
12	2.968232	0.551456	1.0	
13	4.338615	-0.543743	1.0	
14	4.663322	-0.035982	1.0	
15	5.205709	0.775267	1.0	
16	2.025149	1.777222	1.0	
17	2.864437	0.984361	1.0	
18	3.925955	0.853126	1.0	
19	4.113576	0.140379	1.0	
20	3.735551	-1.209276	1.0	

	observable_x2_std	observable_x3_std	observable_x1_scaled_std	\
0	1.0	1.0	1.0	

(continues on next page)

(continued from previous page)

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

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

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

```
[27]:   time  t_presim  k0  k0_preeq  k0_presim  observable_x1  observable_x2  \
0    0.0      0.0  1.0      NaN      NaN      1.918394      0.076250
1    0.5      0.0  1.0      NaN      NaN      0.826073      0.797220
2    1.0      0.0  1.0      NaN      NaN      0.692564      0.748152
3    1.5      0.0  1.0      NaN      NaN      0.266340      1.159657
```

(continues on next page)

(continued from previous page)

4	2.0	0.0	1.0	NaN	NaN	0.326667	1.247234
5	2.5	0.0	1.0	NaN	NaN	0.362526	0.710908
6	3.0	0.0	1.0	NaN	NaN	0.372895	0.441686
7	3.5	0.0	1.0	NaN	NaN	0.374727	0.583920
8	4.0	0.0	1.0	NaN	NaN	0.102518	0.656904
9	4.5	0.0	1.0	NaN	NaN	0.343553	1.120148
10	5.0	0.0	1.0	NaN	NaN	0.121310	0.467665
11	5.5	0.0	1.0	NaN	NaN	0.772284	0.260980
12	6.0	0.0	1.0	NaN	NaN	2.419382	1.365294
13	6.5	0.0	1.0	NaN	NaN	0.041028	0.124470
14	7.0	0.0	1.0	NaN	NaN	0.534293	1.640434
15	7.5	0.0	1.0	NaN	NaN	1.130277	1.044380
16	8.0	0.0	1.0	NaN	NaN	0.823108	0.484711
17	8.5	0.0	1.0	NaN	NaN	0.383893	0.276453
18	9.0	0.0	1.0	NaN	NaN	0.952520	1.049056
19	9.5	0.0	1.0	NaN	NaN	0.996087	1.432980
20	10.0	0.0	1.0	NaN	NaN	1.333280	0.328698

	observable_x3	observable_x1_scaled	observable_x2_offsetted	\
0	1.830396	0.435845	0.396733	
1	0.382144	0.693896	0.243874	
2	0.085469	0.677618	0.338545	
3	0.779226	0.255229	0.096162	
4	0.425859	0.221299	0.652261	
5	0.314667	0.665919	1.060136	
6	0.051562	1.021041	0.430593	
7	0.309506	1.493077	1.334118	
8	1.575355	0.289360	1.843696	
9	0.269241	0.919762	0.995190	
10	0.524276	0.181064	2.435522	
11	0.174355	0.472509	2.764016	
12	2.170329	1.429517	0.633370	
13	1.534095	1.273643	0.747386	
14	0.896389	1.282679	1.081766	
15	1.126667	1.038584	1.633180	
16	1.854798	0.252607	1.538954	
17	0.484237	0.870126	0.691797	
18	1.326394	0.814400	0.377074	
19	0.105097	0.528210	0.571568	
20	0.345096	1.389723	0.199970	

	observable_x1withsigma
0	13.498166
1	2.168002
2	0.330408
3	7.703818
4	18.788001
5	14.458795
6	10.930802
7	27.257116
8	5.139764
9	7.970237
10	22.750521
11	19.955992
12	0.324669
13	10.590418
14	5.478118

(continues on next page)

(continued from previous page)

```

15          2.666993
16         12.717218
17          4.817458
18          3.532245
19          3.569704
20         17.042249

```

```
[28]: # look at the Observables in rdata as DataFrame
```

```
amici.getSimulationObservablesAsDataFrame(model, [edata], [rdata])
```

```
[28]:
```

	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	
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	\
--	-------------------------	------------------------	-------------------	---

(continues on next page)

(continued from previous page)

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
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	

(continues on next page)

(continued from previous page)

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

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

```
[29]:
```

	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
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.
```

(continues on next page)

(continued from previous page)

```
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
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)
```

(continues on next page)

(continued from previous page)

```

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)
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)

```

(continues on next page)



(continued from previous page)

```

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
cclplus: 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
cclplus: 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
cclplus: 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

```

(continues on next page)

(continued from previous page)

```

cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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

```

(continued from previous page)

```

cclplus: 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_rowvals.cpp -
↳o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdw_rowvals.o -std=c++14
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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

```

(continued from previous page)

```

cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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.cpp -o build/
↳temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dJydy.o -std=c++14

```

(continued from previous page)

```

cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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

```



(continued from previous page)

```

cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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_solver.cpp -o build/
↳temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_x_solver.o -std=c++14
cclplus: 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

```

(continued from previous page)

```

cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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

```

(continues on next page)

(continued from previous page)

```
cclplus: 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
cclplus: 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
cclplus: 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
cclplus: 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_
↳rowvals.cpp -o build/temp.linux-x86_64-3.7/Boehm_JProteomeRes2014_dxdotdp_explicit_
↳rowvals.o -std=c++14
cclplus: 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_JSparse_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
```



(continued from previous page)

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'),
 ('kdephospho_prot', 'kdephospho_prot'),
```

(continues on next page)

(continued from previous page)

```
( '__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)
```

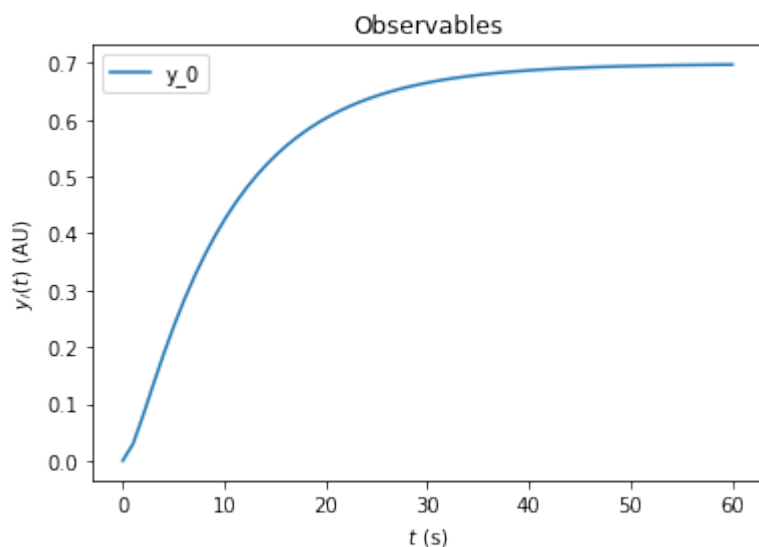
To simulate the model we need to create an instance via the `getModel()` method in the generated model module.

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

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

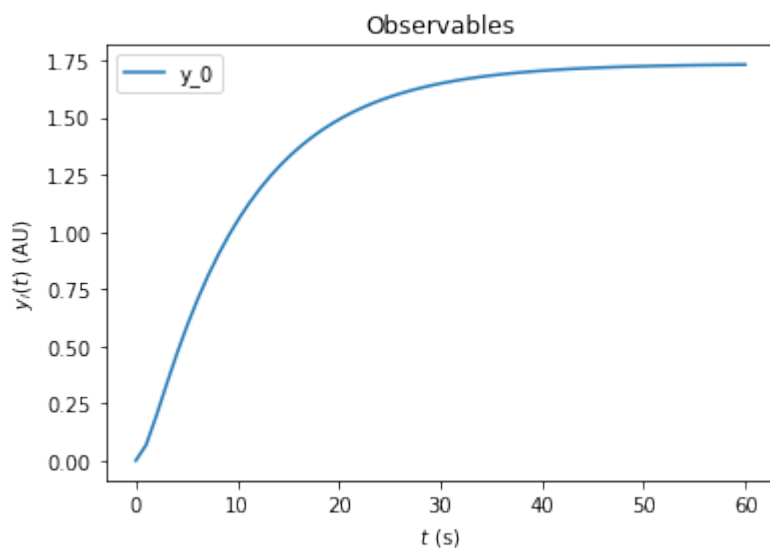
The only thing we need to simulate the model is a timepoint vector, which can be specified using the `setTimepoints` method. If we do not specify any additional options, the default values for `fixedParameters` and `parameters` that were specified in the SBML file will be used.

```
[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.

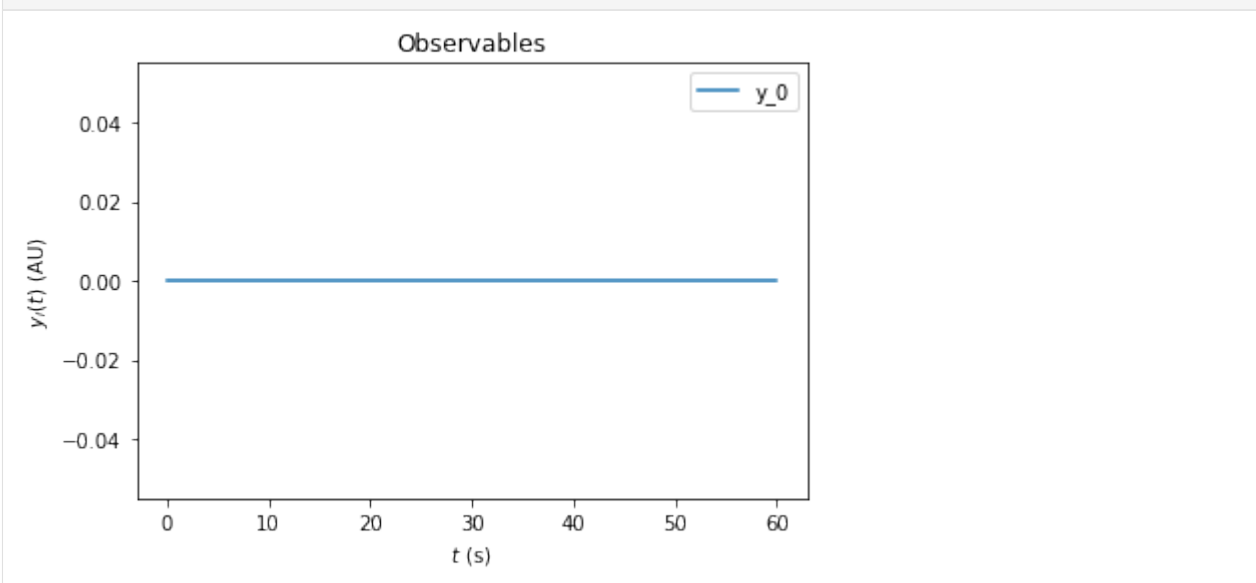
```
[10]: edata = amici.ExpData(rdata, 0.1, 0.0)
edata.fixedParameters = [0,2]
rdata = amici.runAmiciSimulation(model, solver, edata)
amici.plotting.plotObservableTrajectories(rdata)
```



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)
```

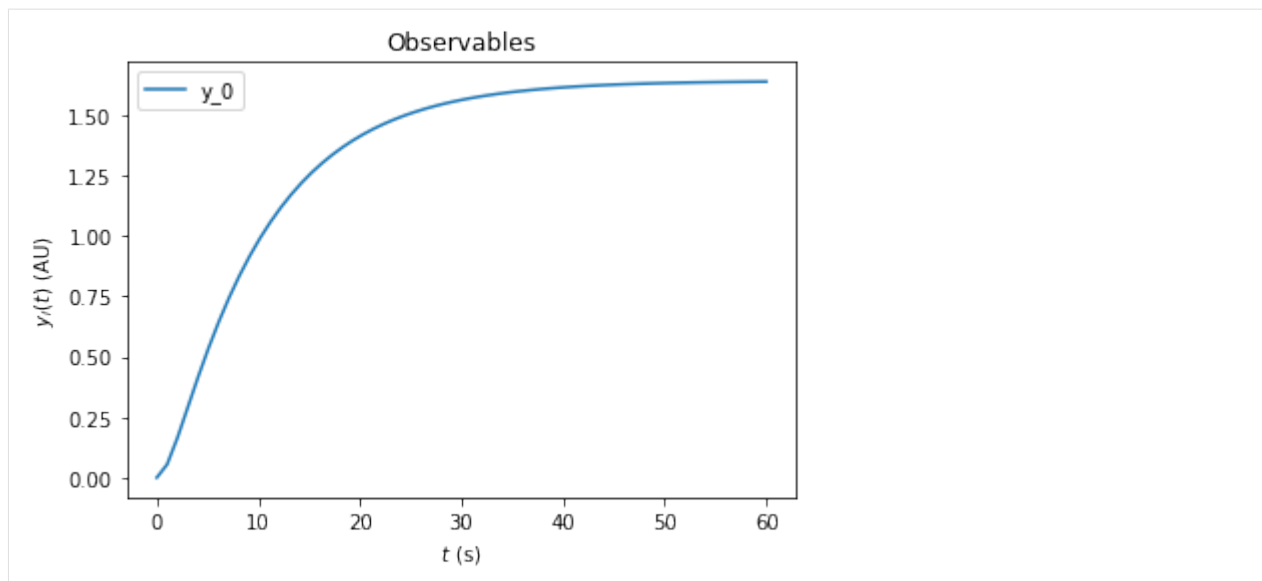


The resulting trajectory is definitely not what one may expect. The problem is that the `DRUG_0` and `KIN_0` set initial conditions for species in the model. By default these initial conditions are only applied at the very beginning of the simulation, i.e., before the preequilibration. Accordingly, the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

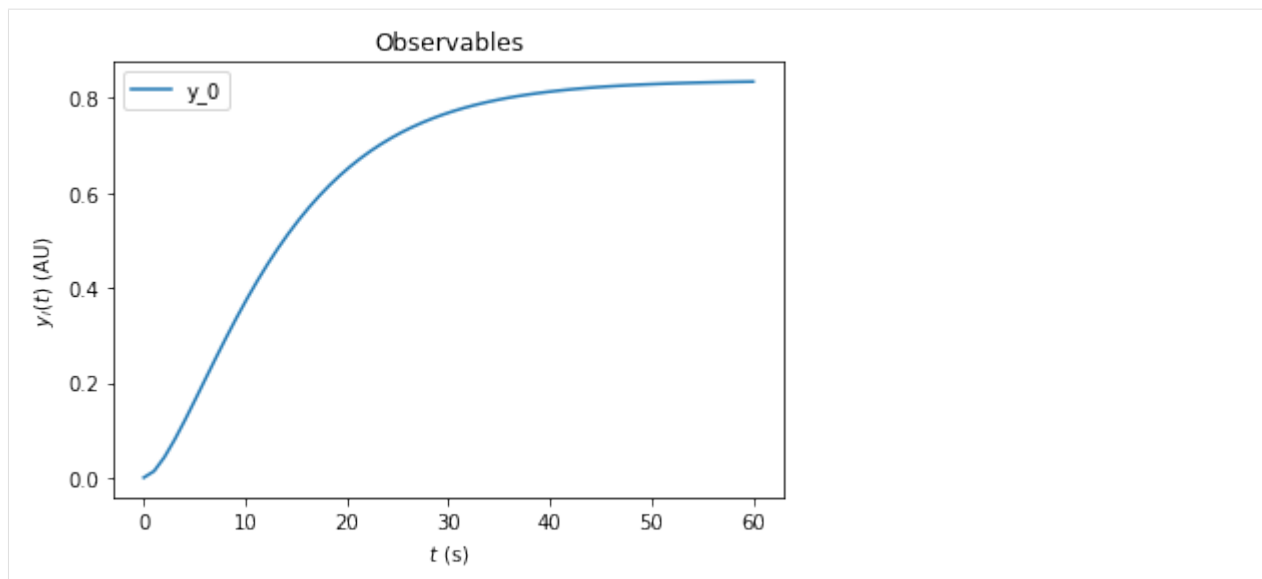
```
[13]: rdata = amici.runAmiciSimulation(model, solver, edata)
      amici.plotting.plotObservableTrajectories(rdata)
```



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

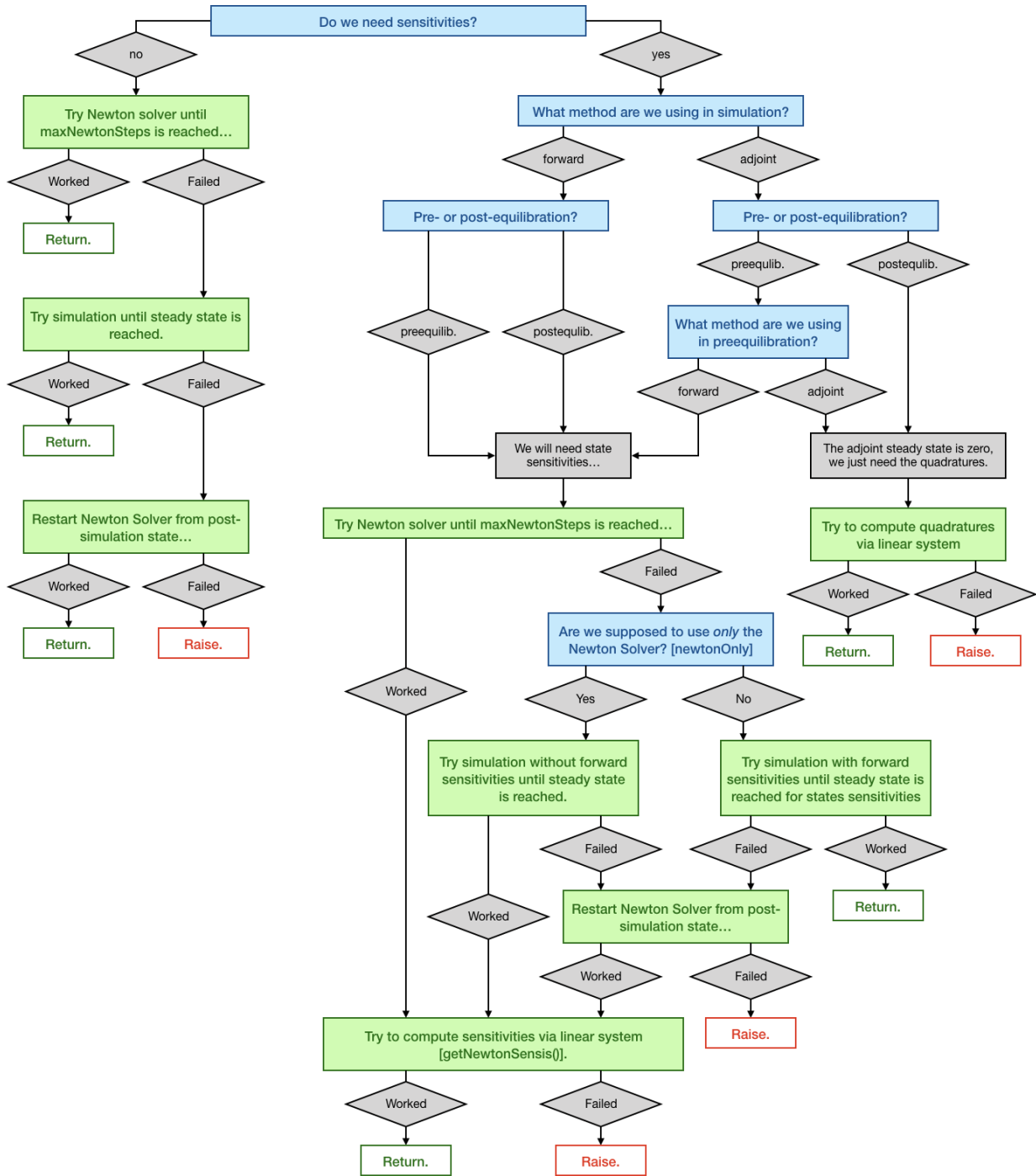
If AMICI has to equilibrate a dynamical system, it can do this either via simulating until the right hand side of the system becomes small, or it can try to find the steady state directly by Newton's method. Amici decides automatically which approach is chosen and how forward or adjoint sensitivities 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]:



## 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,
                          constantParameters=constantParameters,
                          sigmas=sigmas)
sbml_importer.sbml2amici(model_name,
                          model_output_dir,
                          observables=observables,
                          constantParameters=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()

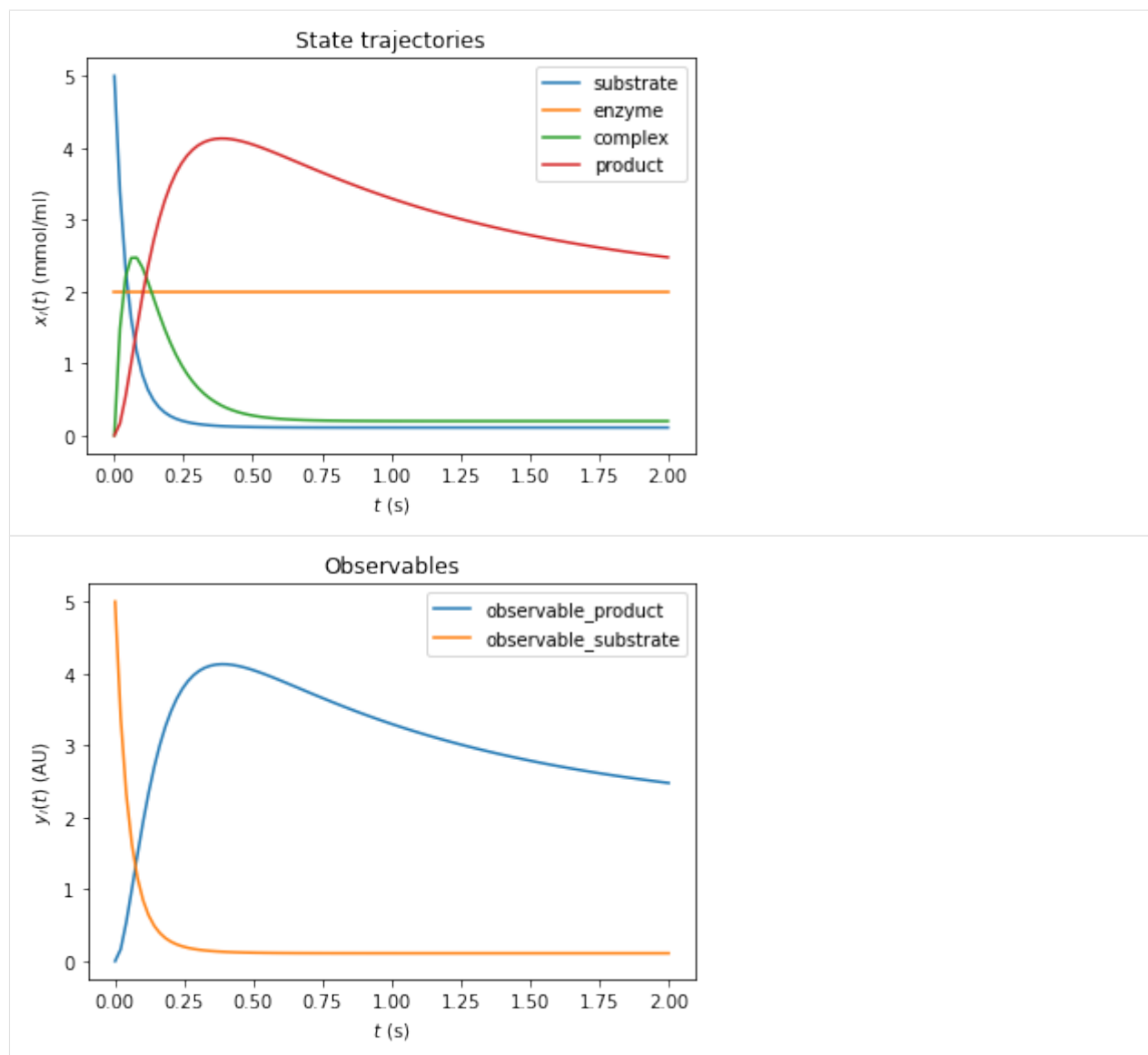
model_module = amici.import_model_module(model_name, os.path.abspath(model_output_
↳ dir))
model = model_module.getModel()

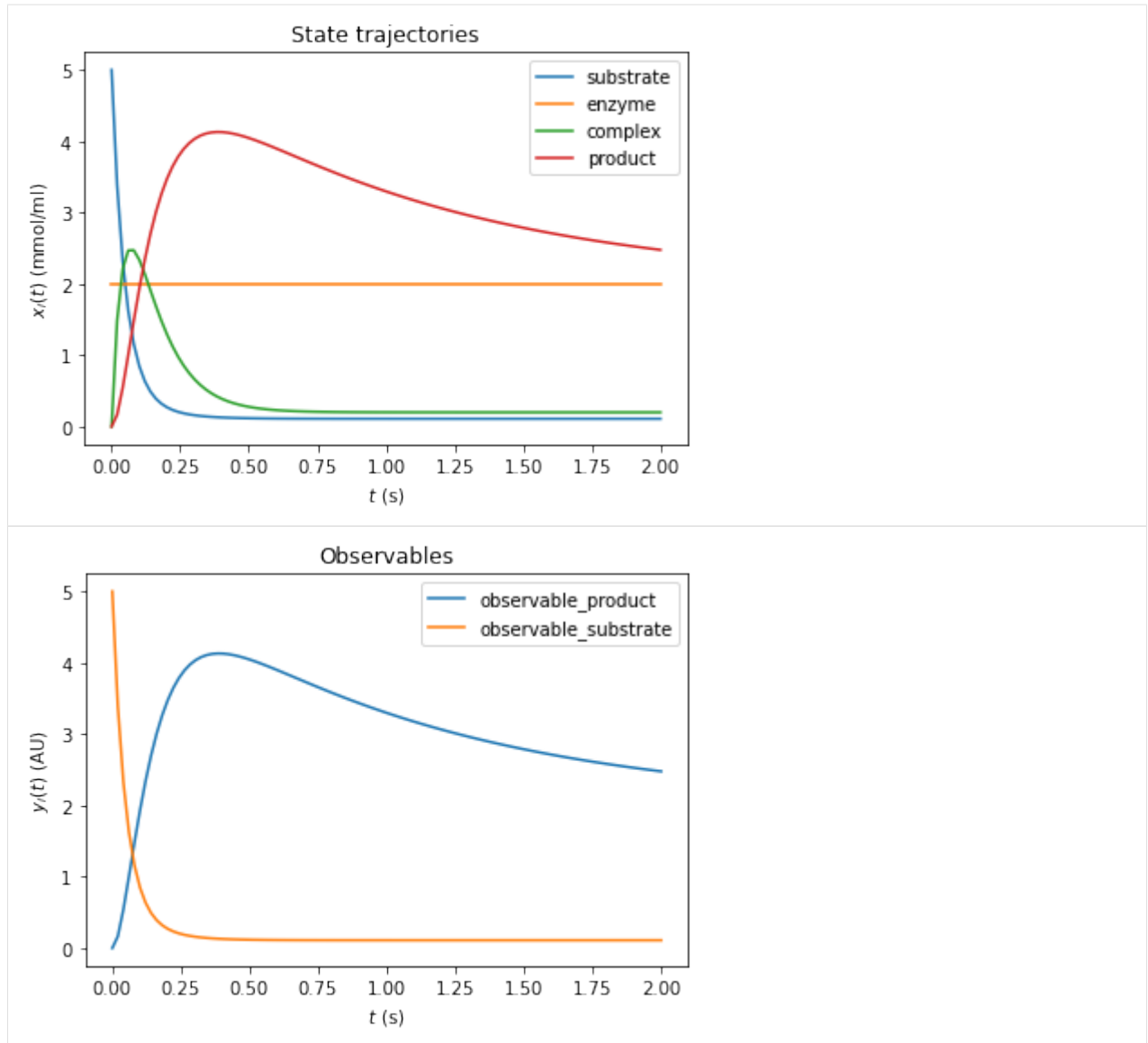
# simulate model with conservation laws
model_reduced.setTimepoints(np.linspace(0, 2, 100))
solver_reduced = model_reduced.getSolver()
rdata_reduced = amici.runAmiciSimulation(model_reduced, solver_reduced)

# 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)
```





### 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
    numerrtestfails: None
    numnonlinsolvconvfails: None
        order: None
        cpu_time: 0.0
        numstepsB: None

```

(continues on next page)

(continued from previous page)

```
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 could 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.
Error occurred 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
```

(continues on next page)

(continued from previous page)

```

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:

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

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:





(continued from previous page)

```

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]
```

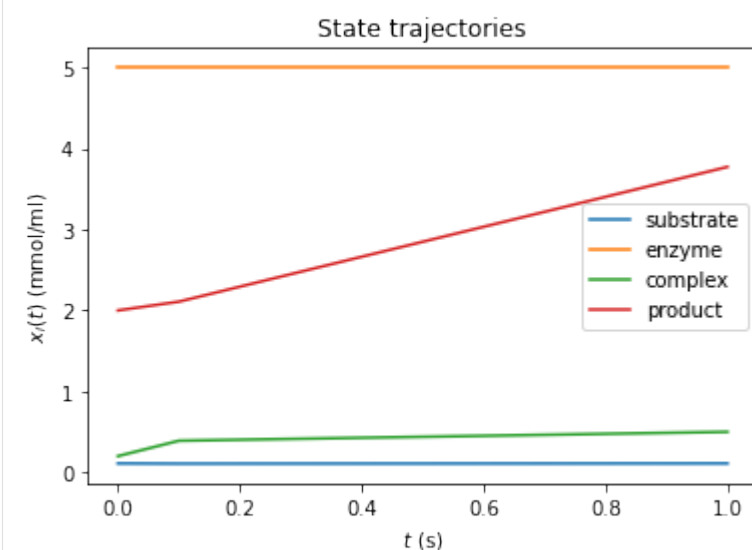
## Preequilibrating the model

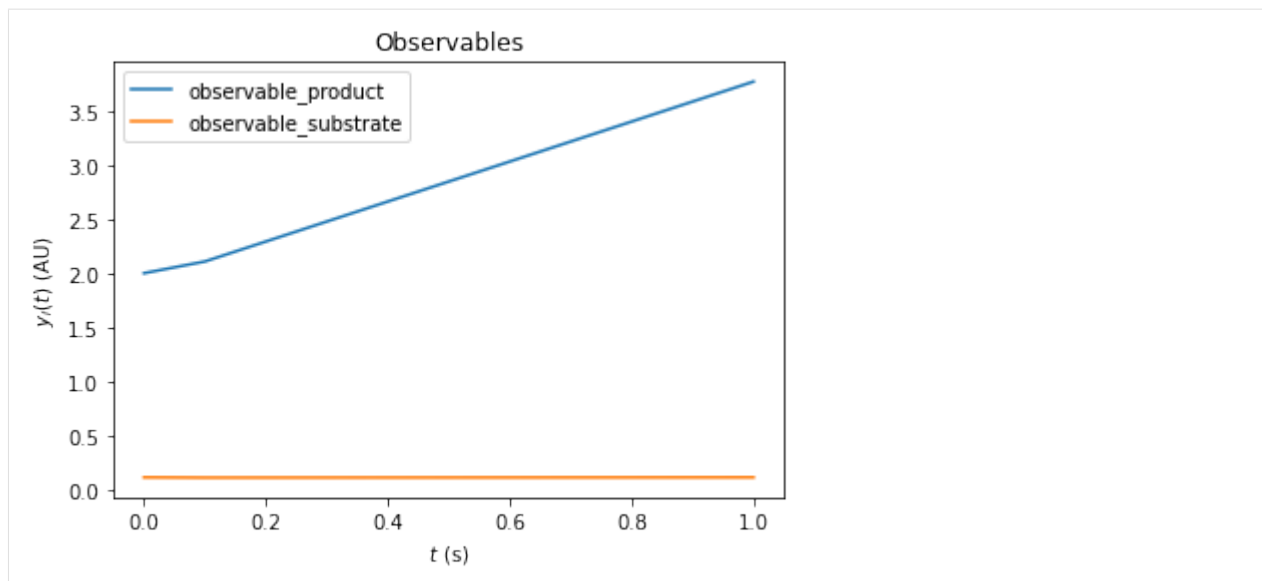
Sometimes, we want to launch a solver run from a steady state which was inferred numerically, i.e., the system was preequilibrated. In order to do this with AMICI, we need to pass an `ExpData` object, which contains fixed parameter for the actual simulation and for preequilibration of the model.

```
[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 postequilibration 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 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      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.simulationFSA)
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: 0.9920376238481097
    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

```

(continues on next page)

(continued from previous page)

```

preeq_numsteps:  [[2 0 0]]
preeq_numstepsB:  0.0
preeq_status:     [[1 0 0]]
preeq_cpu_time:   0.036
preeq_cpu_timeB:  0.0

```

## Preequilibration 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  $s_{x0}$  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  $s_{x0}$ ), with the adjoint state ( $p(t = 0)$ ). Hence, the matrix  $s_{x0}$  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)
```

(continues on next page)



(continued from previous page)

```

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

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

      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 simulation
model_reduced.setSteadyStateSensitivityMode(amici.SteadyStateSensitivityMode.
    ↳simulationFSA)
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'])

```

(continues on next page)

(continued from previous page)

```
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 corresponding to steady state:  
6.002011407974004  
31.0689293433781

computation time to reach steady state:  
7.646  
7.837

### 10.2.3 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

<code>amici</code>	AMICI
<code>amici.amici</code>	Core C++ bindings
<code>amici.sbml_import</code>	SBML Import
<code>amici.pysb_import</code>	PySB Import
<code>amici.petab_import</code>	PETab Import
<code>amici.petab_import_pysb</code>	PySB-PETab Import
<code>amici.petab_objective</code>	PETab Objective
<code>amici.import_utils</code>	Miscellaneous functions related to model import, independent of any specific model format
<code>amici.ode_export</code>	C++ Export
<code>amici.plotting</code>	Plotting
<code>amici.pandas</code>	Pandas Wrappers
<code>amici.logging</code>	Logging
<code>amici.gradient_check</code>	Finite Difference Check
<code>amici.parameter_mapping</code>	Parameter mapping

### 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

**var capture\_cstdout** context to redirect C/C++ stdout to python stdout if python stdout was redirected (doing nothing if not redirected).

## Classes

---

<code>add_path(path)</code>	Context manager for temporarily changing PYTHONPATH
-----------------------------	---

---

### amici.add\_path

**class** `amici.add_path(path)`  
Context manager for temporarily changing PYTHONPATH

**\_\_init\_\_**(*path*)  
Initialize self. See help(type(self)) for accurate signature.

#### Methods Summary

---

<code>__init__(path)</code>	Initialize self.
-----------------------------	------------------

---

#### Methods

**\_\_init\_\_**(*path*)  
Initialize self. See help(type(self)) for accurate signature.

## Functions Summary

---

<code>ExpData(*args)</code>	Convenience wrapper for <code>amici.amici.ExpData</code> constructors
<code>import_model_module(module_name[, module_path])</code>	Import Python module of an AMICI model
<code>readSolverSettingsFromHDF5(file, solver[, ...])</code>	Convenience wrapper for <code>amici.readSolverSettingsFromHDF5()</code>
<code>runAmiciSimulation(model, solver[, edata])</code>	Convenience wrapper around <code>amici.amici.runAmiciSimulation()</code>
<code>runAmiciSimulations(model, solver, edata_list)</code>	Convenience wrapper for loops of <code>amici.runAmiciSimulation</code>
<code>writeSolverSettingsToHDF5(solver, file[, ...])</code>	Convenience wrapper for <code>amici.amici.writeSolverSettingsToHDF5()</code>

---

## Functions

`amici.ExpData(*args)`

Convenience wrapper for `amici.amici.ExpData` constructors

**Parameters** `args` – arguments

**Return type** `amici.amici.ExpData`

**Returns** ExpData Instance

`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.Optional[str]`) – Absolute or relative path of the package directory

**Return type** `module`

**Returns** The model module

`amici.readSolverSettingsFromHDF5(file, solver, location='solverSettings')`

Convenience wrapper for `amici.readSolverSettingsFromHDF5()`

**Parameters**

- **file** (`str`) – hdf5 filename
- **solver** (`typing.Union[amici.amici.Solver, amici.amici.SolverPtr]`) – Solver instance to which settings will be transferred
- **location** (`typing.Optional[str]`) – location of solver settings in hdf5 file

**Return type** `None`

`amici.runAmiciSimulation(model, solver, edata=None)`

Convenience wrapper around `amici.amici.runAmiciSimulation()` (generated by swig)

**param model** Model instance

**`:param solver:**

Solver instance, must be generated from `amici.amici.Model.getSolver()`

**param edata** ExpData instance (optional)

**returns** ReturnData object with simulation results

**Return type** `amici.numpy.ReturnDataView`

`amici.runAmiciSimulations(model, solver, edata_list, failfast=True, num_threads=1)`

Convenience wrapper for loops of `amici.runAmiciSimulation`

**Parameters**

- **model** (`typing.Union[amici.amici.Model, amici.amici.ModelPtr]`) – Model instance
- **solver** (`typing.Union[amici.amici.Solver, amici.amici.SolverPtr]`) – Solver instance, must be generated from `Model.getSolver()`

- **edata\_list** (`typing.Union[amici.amici.ExpDataPtrVector, typing.Sequence[typing.Union[amici.amici.ExpData, amici.amici.ExpDataPtr]]]`) – list of `ExpData` instances
- **failfast** (`bool`) – returns as soon as an integration failure is encountered
- **num\_threads** (`int`) – number of threads to use (only used if compiled with `openmp`)

**Return type** `typing.List[amici.numpy.ReturnDataView]`

**Returns** list of simulation results

`amici.writeSolverSettingsToHDF5(solver, file, location='solverSettings')`

Convenience wrapper for `amici.amici.writeSolverSettingsToHDF5()`

**Parameters**

- **file** (`typing.Union[str, object]`) – hdf5 filename, can also be object created by `amici.amici.createOrOpenForWriting()`
- **solver** (`typing.Union[amici.amici.Solver, amici.amici.SolverPtr]`) – Solver instance from which settings will stored
- **location** (`typing.Optional[str]`) – location of solver settings in hdf5 file

**Return type** `None`

## 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

<code>BoolVector(*args)</code>	Swig-Generated class templating common python types including <code>Iterable [bool]</code> and <code>numpy.array [bool]</code> to facilitate interfacing with C++ bindings.
<code>DoubleVector(*args)</code>	Swig-Generated class templating common python types including <code>Iterable [float]</code> and <code>numpy.array [float]</code> to facilitate interfacing with C++ bindings.
<code>ExpData(*args)</code>	<code>ExpData</code> carries all information about experimental or condition-specific data
<code>ExpDataPtr(*args)</code>	Swig-Generated class that implements smart pointers to <code>ExpData</code> as objects.
<code>ExpDataPtrVector(*args)</code>	Swig-Generated class templating common python types including <code>Iterable [amici.amici.ExpData]</code> and <code>numpy.array [amici.amici.ExpData]</code> to facilitate interfacing with C++ bindings.
<code>FixedParameterContext(value)</code>	An enumeration.

continues on next page

Table 5 – continued from previous page

<i>IntVector</i> (*args)	Swig-Generated class templating common python types including <code>Iterable [int]</code> and <code>numpy.array [int]</code> to facilitate interfacing with C++ bindings.
<i>InternalSensitivityMethod</i> (value)	An enumeration.
<i>InterpolationType</i> (value)	An enumeration.
<i>LinearMultistepMethod</i> (value)	An enumeration.
<i>LinearSolver</i> (value)	An enumeration.
<i>Model</i> (*args, **kwargs)	The <code>Model</code> class represents an AMICI ODE/DAE model.
<i>ModelDimensions</i> (*args)	Container for model dimensions.
<i>ModelPtr</i> (*args)	Swig-Generated class that implements smart pointers to <code>Model</code> as objects.
<i>NewtonDampingFactorMode</i> (value)	An enumeration.
<i>NonlinearSolverIteration</i> (value)	An enumeration.
<i>ParameterScaling</i> (value)	An enumeration.
<i>ParameterScalingVector</i> (*args)	Swig-Generated class, which, in contrast to other <code>Vector</code> classes, does not allow for simple interoperability with common python types, but must be created using <code>amici.amici.parameterScalingFromIntVector()</code>
<i>RDataReporting</i> (value)	An enumeration.
<i>ReturnData</i> (*args)	Stores all data to be returned by <code>amici.amici.runAmiciSimulation()</code> .
<i>ReturnDataPtr</i> (*args)	Swig-Generated class that implements smart pointers to <code>ReturnData</code> as objects.
<i>SecondOrderMode</i> (value)	An enumeration.
<i>SensitivityMethod</i> (value)	An enumeration.
<i>SensitivityOrder</i> (value)	An enumeration.
<i>SimulationParameters</i> (*args)	Container for various simulation parameters.
<i>Solver</i> (*args, **kwargs)	The <code>Solver</code> class provides a generic interface to CVODES and IDAS solvers, individual realizations are realized in the <code>CVodeSolver</code> and the <code>IDASolver</code> class.
<i>SolverPtr</i> (*args)	Swig-Generated class that implements smart pointers to <code>Solver</code> as objects.
<i>SteadyStateSensitivityMode</i> (value)	An enumeration.
<i>SteadyStateStatus</i> (value)	An enumeration.
<i>SteadyStateStatusVector</i> (*args)	
<i>StringDoubleMap</i> (*args)	Swig-Generated class templating <code>Dict [str, float]</code> to facilitate interfacing with C++ bindings.
<i>StringVector</i> (*args)	Swig-Generated class templating common python types including <code>Iterable [str]</code> and <code>numpy.array [str]</code> 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

<code>__init__(*args)</code>	<i>Overload 1:</i>
<code>getObservedData()</code>	get function that copies data from ExpData::observedData to output
<code>getObservedDataPtr(it)</code>	get function that returns a pointer to observed data at index
<code>getObservedDataStdDev()</code>	get function that copies data from ExpData::observedDataStdDev to output
<code>getObservedDataStdDevPtr(it)</code>	get function that returns a pointer to standard deviation of observed data at index
<code>getObservedEvents()</code>	get function that copies data from ExpData::mz to output
<code>getObservedEventsPtr(ie)</code>	get function that returns a pointer to observed data at ieth occurrence
<code>getObservedEventsStdDev()</code>	get function that copies data from ExpData::observedEventsStdDev to output
<code>getObservedEventsStdDevPtr(ie)</code>	get function that returns a pointer to standard deviation of observed event data at ie-th occurrence
<code>getTimepoint(it)</code>	get function that returns timepoint at index

continues on next page

Table 6 – continued from previous page

<code>getTimepoints()</code>	get function that copies data from <code>ExpData::ts</code> to output
<code>isSetObservedData(it, iy)</code>	get function that checks whether data at specified indices has been set
<code>isSetObservedDataStdDev(it, iy)</code>	get function that checks whether standard deviation of data at specified indices has been set
<code>isSetObservedEvents(ie, iz)</code>	get function that checks whether event data at specified indices has been set
<code>isSetObservedEventsStdDev(ie, iz)</code>	get function that checks whether standard deviation of even data at specified indices has been set
<code>nmaxevent()</code>	maximal number of events to track
<code>nt()</code>	number of timepoints
<code>nytrue()</code>	number of observables of the non-augmented model
<code>nztrue()</code>	number of event observables of the non-augmented model
<code>reinitializeAllFixedParameterDependentStates()</code>	Set reinitialization of (all) states based on model constants for all simulation phases.
<code>reinitializeAllFixedParameterDependentStates(preequilibration)</code>	Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).
<code>reinitializeAllFixedParameterDependentStates(simulation)</code>	Set reinitialization of all states based on (model) constants for the ‘main’ simulation (only meaningful if presimulation or preequilibration is performed).
<code>setObservedData(*args)</code>	<i>Overload 1:</i>
<code>setObservedDataStdDev(*args)</code>	<i>Overload 1:</i>
<code>setObservedEvents(*args)</code>	<i>Overload 1:</i>
<code>setObservedEventsStdDev(*args)</code>	<i>Overload 1:</i>
<code>setTimepoints(ts)</code>	Set function that copies data from input to <code>ExpData::ts</code>

### Attributes

<code>fixedParameters</code>	Model constants
<code>fixedParametersPreequilibration</code>	Model constants for pre-equilibration
<code>fixedParametersPresimulation</code>	Model constants for pre-simulation
<code>parameters</code>	Model parameters
<code>plist</code>	Parameter indices w.r.t.
<code>pscale</code>	Parameter scales
<code>reinitialization_state_idxes_presim</code>	Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.
<code>reinitialization_state_idxes_sim</code>	Indices of states to be reinitialized based on provided constants / fixed parameters.
<code>reinitializeFixedParameterInitialStates</code>	Flag indicating whether reinitialization of states depending on fixed parameters is activated
<code>sx0</code>	Initial state sensitivities
<code>t_presim</code>	Duration of pre-simulation.
<code>ts_</code>	Timepoints for which model state/outputs/.
<code>tstart_</code>	starting time
<code>x0</code>	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\_idxes\_presim* and *reinitialization\_state\_idxes\_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\_idxes\_presim* and *reinitialization\_state\_idxes\_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\_idxes\_presim* and *reinitialization\_state\_idxes\_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. .. rubric:: Attributes .. autosummary:

```
~ExpDataPtr.fixedParameters
~ExpDataPtr.fixedParametersPreequilibration
~ExpDataPtr.fixedParametersPresimulation
~ExpDataPtr.parameters
~ExpDataPtr.plist
~ExpDataPtr.pscale
~ExpDataPtr.reinitialization_state_idxes_presim
~ExpDataPtr.reinitialization_state_idxes_sim
~ExpDataPtr.reinitializeFixedParameterInitialStates
~ExpDataPtr.sx0
~ExpDataPtr.t_presim
~ExpDataPtr.ts_
~ExpDataPtr.tstart_
~ExpDataPtr.x0
```

### **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**

preequilibration
presimulation
simulation

### **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

BDF
adams

## amici.amici.LinearSolver

**class** amici.amici.**LinearSolver**(*value*)  
 An enumeration.

### Attributes

KLU
LAPACKBand
LAPACKDense
SPBCG
SPGMR
SPTFQMR
SuperLUMT
band
dense
diag

## 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  $x\dot{w}$
- **ndJydy** (*IntVector*) – Number of nonzero elements in the  $y$  derivative of  $dJy$  (shape *nytrue*)
- **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

<code>__init__</code> (*args, **kwargs)	<i>Overload 1:</i>
<code>clone</code> ()	Clone this instance.
<code>getAlwaysCheckFinite</code> ()	Get setting of whether the result of every call to <i>Model::f*</i> should be checked for finiteness.
<code>getAmiciCommit</code> ()	Returns the AMICI commit that was used to generate the model
<code>getAmiciVersion</code> ()	Returns the AMICI version that was used to generate the model

continues on next page

Table 13 – continued from previous page

<i>getExpressionIds()</i>	Get IDs of the expression.
<i>getExpressionNames()</i>	Get names of the expressions.
<i>getFixedParameterById(par_id)</i>	Get value of fixed parameter with the specified ID.
<i>getFixedParameterByName(par_name)</i>	Get value of fixed parameter with the specified name.
<i>getFixedParameterIds()</i>	Get IDs of the fixed model parameters.
<i>getFixedParameterNames()</i>	Get names of the fixed model parameters.
<i>getFixedParameters()</i>	Get values of fixed parameters.
<i>getInitialStateSensitivities()</i>	Get the initial states sensitivities.
<i>getInitialStates()</i>	Get the initial states.
<i>getName()</i>	Get the model name.
<i>getObservableIds()</i>	Get IDs of the observables.
<i>getObservableNames()</i>	Get names of the observables.
<i>getParameterById(par_id)</i>	Get value of first model parameter with the specified ID.
<i>getParameterByName(par_name)</i>	Get value of first model parameter with the specified name.
<i>getParameterIds()</i>	Get IDs of the model parameters.
<i>getParameterList()</i>	Get the list of parameters for which sensitivities are computed.
<i>getParameterNames()</i>	Get names of the model parameters.
<i>getParameterScale()</i>	Get parameter scale for each parameter.
<i>getParameters()</i>	Get parameter vector.
<i>getReinitializationStateIdxs()</i>	Return indices of states to be reinitialized based on provided constants / fixed parameters
<i>getReinitializeFixedParameterInitialStates()</i>	Get whether initial states depending on fixedParameters are to be reinitialized after preequilibration and presimulation.
<i>getSolver()</i>	Retrieves the solver object
<i>getStateIds()</i>	Get IDs of the model states.
<i>getStateIsNonNegative()</i>	Get flags indicating whether states should be treated as non-negative.
<i>getStateNames()</i>	Get names of the model states.
<i>getSteadyStateSensitivityMode()</i>	Gets the mode how sensitivities are computed in the steadystate simulation.
<i>getTimepoint(it)</i>	Get simulation timepoint for time index <i>it</i> .
<i>getTimepoints()</i>	Get the timepoint vector.
<i>getUnscaledParameters()</i>	Get parameters with transformation according to parameter scale applied.
<i>hasCustomInitialStateSensitivities()</i>	Return whether custom initial state sensitivities have been set.
<i>hasCustomInitialStates()</i>	Return whether custom initial states have been set.
<i>hasExpressionIds()</i>	Report whether the model has expression IDs set.
<i>hasExpressionNames()</i>	Report whether the model has expression names set.
<i>hasFixedParameterIds()</i>	Report whether the model has fixed parameter IDs set.
<i>hasFixedParameterNames()</i>	Report whether the model has fixed parameter names set.
<i>hasObservableIds()</i>	Report whether the model has observable IDs set.
<i>hasObservableNames()</i>	Report whether the model has observable names set.
<i>hasParameterIds()</i>	Report whether the model has parameter IDs set.
<i>hasParameterNames()</i>	Report whether the model has parameter names set.

continues on next page

Table 13 – continued from previous page

<i>hasQuadraticLLH()</i>	Checks whether the defined noise model is gaussian, i.e., the nllh is quadratic
<i>hasStateIds()</i>	Report whether the model has state IDs set.
<i>hasStateNames()</i>	Report whether the model has state names set.
<i>isFixedParameterStateReinitializationFunction()</i>	Function indicating whether reinitialization of states depending on fixed parameters is permissible
<i>k()</i>	Get fixed parameters.
<i>nMaxEvent()</i>	Get maximum number of events that may occur for each type.
<i>ncl()</i>	Get number of conservation laws.
<i>nk()</i>	Get number of constants
<i>np()</i>	Get total number of model parameters.
<i>nplist()</i>	Get number of parameters wrt to which sensitivities are computed.
<i>nt()</i>	Get number of timepoints.
<i>nx_reinit()</i>	Get number of solver states subject to reinitialization.
<i>p[plist(pos)]</i>	Get entry in parameter list by index.
<i>requireSensitivitiesForAllParameters(p)</i>	Require computation of sensitivities for all parameters p [0..np[ in natural order.
<i>setAllStatesNonNegative()</i>	Set flags indicating that all states should be treated as non-negative.
<i>setAlwaysCheckFinite(alwaysCheck)</i>	Set whether the result of every call to <i>Model::f*</i> should be checked for finiteness.
<i>setFixedParameterById(par_id, value)</i>	Set value of first fixed parameter with the specified ID.
<i>setFixedParameterByName(par_name, value)</i>	Set value of first fixed parameter with the specified name.
<i>setFixedParameters(k)</i>	Set values for constants.
<i>setFixedParametersByIdRegex(par_id_regex, value)</i>	Set values of all fixed parameters with the ID matching the specified regex.
<i>setFixedParametersByNameRegex(...)</i>	Set value of all fixed parameters with name matching the specified regex.
<i>setInitialStateSensitivities(sx0)</i>	Set the initial state sensitivities.
<i>setInitialStates(x0)</i>	Set the initial states.
<i>setNMaxEvent(nmaxevent)</i>	Set maximum number of events that may occur for each type.
<i>setParameterById(*args)</i>	<i>Overload 1:</i>
<i>setParameterByName(*args)</i>	<i>Overload 1:</i>
<i>setParameterList(plist)</i>	Set the list of parameters for which sensitivities are to be computed.
<i>setParameterScale(*args)</i>	<i>Overload 1:</i>
<i>setParameters(p)</i>	Set the parameter vector.
<i>setParametersByIdRegex(par_id_regex, value)</i>	Set all values of model parameters with IDs matching the specified regular expression.
<i>setParametersByNameRegex(par_name_regex, value)</i>	Set all values of all model parameters with names matching the specified regex.
<i>setReinitializationStateIdxs(idxs)</i>	Set indices of states to be reinitialized based on provided constants / fixed parameters

continues on next page



Table 13 – continued from previous page

<code>setReinitializeFixedParameterInitialStates(flag)</code>	Set whether initial states depending on fixed parameters are to be reinitialized after preequilibration and presimulation.
<code>setStateIsNonNegative(stateIsNonNegative)</code>	Set flags indicating whether states should be treated as non-negative.
<code>setSteadyStateSensitivityMode(mode)</code>	Set the mode how sensitivities are computed in the steadystate simulation.
<code>setT0(t0)</code>	Set simulation start time.
<code>setTimepoints(ts)</code>	Set the timepoint vector.
<code>setUnscaledInitialStateSensitivities(s0)</code>	Set the initial state sensitivities.
<code>t0()</code>	Get simulation start time.

### Attributes

<code>app</code>	AMICI application context
<code>idlist</code>	Flag array for DAE equations
<code>lbw</code>	Lower bandwidth of the Jacobian
<code>nJ</code>	Dimension of the augmented objective function for 2nd order ASA
<code>ndJydy</code>	Number of nonzero elements in the <i>derivativeof</i> dJy (dimension <i>nytrue</i> )
<code>ndwdp</code>	Number of nonzero elements in the <i>p</i> derivative of the repeating elements
<code>ndwdw</code>	Number of nonzero elements in the <i>w</i> derivative of the repeating elements
<code>ndwdx</code>	Number of nonzero elements in the <i>x</i> derivative of the repeating elements
<code>ndxdotdw</code>	Number of nonzero elements in the <i>derivativeof</i> xdot
<code>ne</code>	Number of events
<code>nnz</code>	Number of nonzero entries in Jacobian
<code>nw</code>	Number of common expressions
<code>nx_rdata</code>	Number of states
<code>nx_solver</code>	Number of states with conservation laws applied
<code>nx_solver_reinit</code>	Number of solver states subject to reinitialization
<code>nxtrue_rdata</code>	Number of states in the unaugmented system
<code>nxtrue_solver</code>	Number of states in the unaugmented system with conservation laws applied
<code>ny</code>	Number of observables
<code>nytrue</code>	Number of observables in the unaugmented system
<code>nz</code>	Number of event outputs
<code>nztrue</code>	Number of event outputs in the unaugmented system
<code>o2mode</code>	Flag indicating whether for <code>amici::Solver::sensi_ == amici::SensitivityOrder::second</code> directional or full second order derivative will be computed
<code>pythonGenerated</code>	Flag indicating Matlab- or Python-based model generation
<code>ubw</code>	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  $x\dot{d}ot$
- `ndJydy` (`IntVector`) – Number of nonzero elements in the  $y$  derivative of  $dJy$  (shape `nytrue`)
- `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

**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** string

**Returns** AMICI commit string

**getAmiciVersion** () → str

Returns the AMICI version that was used to generate the model

**Return type** string

**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

**getName** () → *str*

Get the model name.

**Return type** *string*

**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

**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

**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

**getSteadyStateSensitivityMode** () → *amici.amici.SteadyStateSensitivityMode*

Gets the mode how sensitivities are computed in the steadystate simulation.

**Return type** `int`

**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`

**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*

**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*, *optional*) – 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** (*string*) – 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** (*string*) – 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, optional*) – 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*

*Overload 1:*

Set parameter scale for each parameter.

NOTE: Resets initial state sensitivities.

**Parameters** **pscale** (*int*) – Scalar parameter scale to be set for all parameters

*Overload 2:*

Set parameter scale for each parameter.

NOTE: Resets initial state sensitivities.

**Parameters** **pscaleVec** (*ParameterScalingVector >*) – Vector of parameter scales

**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  $x\dot{d}ot$
- **ndJydy** (*IntVector*) – Number of nonzero elements in the  $y$  derivative of  $dJy$  (shape *nytrue*)
- **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 <i>derivativeof</i> dJy (dimension <i>nytrue</i> )
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 <i>derivativeof</i> xdot
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

continues on next page

Table 16 – continued from previous page

<code>nxtrue_rdata</code>	Number of states in the unaugmented system
<code>nxtrue_solver</code>	Number of states in the unaugmented system with conservation laws applied
<code>ny</code>	Number of observables
<code>nytrue</code>	Number of observables in the unaugmented system
<code>nz</code>	Number of event outputs
<code>nztrue</code>	Number of event outputs in the unaugmented system
<code>ubw</code>	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`)
- **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. .. rubric:: Attributes .. autosummary:

```
~ModelPtr.app
~ModelPtr.idlist
~ModelPtr.lbw
~ModelPtr.nJ
~ModelPtr.ndJydy
~ModelPtr.ndwdp
~ModelPtr.ndwdw
~ModelPtr.ndwdx
~ModelPtr.ndxdotdw
~ModelPtr.ne
~ModelPtr.nnz
~ModelPtr.nw
~ModelPtr.nx_rdata
~ModelPtr.nx_solver
~ModelPtr.nx_solver_reinit
~ModelPtr.nxtrue_rdata
~ModelPtr.nxtrue_solver
~ModelPtr.ny
~ModelPtr.nytrue
~ModelPtr.nz
~ModelPtr.nztrue
~ModelPtr.o2mode
~ModelPtr.pythonGenerated
~ModelPtr.ubw
```

### **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
newton

**amici.amici.ParameterScaling**

**class** amici.amici.**ParameterScaling**(*value*)  
 An enumeration.

**Attributes**

ln
log10
none

**amici.amici.ParameterScalingVector**

**class** amici.amici.**ParameterScalingVector**(\*args)  
 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()`

**amici.amici.RDataReporting**

**class** amici.amici.**RDataReporting**(*value*)  
 An enumeration.

**Attributes**

full
likelihood
residuals

**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** (*int*) – see amici::Solver::sensi
- **sensi\_meth** (*int*) – see amici::Solver::sensi\_meth
- **rdrn** (*int*) – see amici::Solver::rdata\_reporting
- **quadratic\_llh** (*boolean*) – whether model defines a quadratic nllh and computing res, sres and FIM makes sense

*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

<code>__init__(*args)</code>	<i>Overload 1:</i>
------------------------------	--------------------

## Attributes

FIM	fisher information matrix (shape <i>nplist</i> x <i>nplist</i> , row-major)
J	Jacobian of differential equation right hand side (shape <i>nx</i> x <i>nx</i> , row-major)
chi2	$\chi^2$ value
cpu_time	computation time of forward solve [ms]
cpu_timeB	computation time of backward solve [ms]
lbw	Lower bandwidth of the Jacobian
llh	log-likelihood value
nJ	Dimension of the augmented objective function for 2nd order ASA
ndJydy	Number of nonzero elements in the <i>derivativeof</i> dJy (dimension <i>nytrue</i> )
ndwdp	Number of nonzero elements in the <i>p</i> derivative of the repeating elements
ndwdw	Number of nonzero elements in the <i>w</i> derivative of the repeating elements
ndwdx	Number of nonzero elements in the <i>x</i> derivative of the repeating elements
ndxdotdw	Number of nonzero elements in the <i>derivativeof</i> xdot
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 <i>nt</i> )
numerrtestfailsB	number of error test failures backward problem (shape <i>nt</i> )
numnonlinsolvconvfails	number of linear solver convergence failures forward problem (shape <i>nt</i> )
numnonlinsolvconvfailsB	number of linear solver convergence failures backward problem (shape <i>nt</i> )
numrhsevals	number of right hand side evaluations forward problem (shape <i>nt</i> )

continues on next page

Table 22 – continued from previous page

numrhsevalsB	number of right hand side evaluations backward problem (shape <i>nt</i> )
numsteps	number of integration steps forward problem (shape <i>nt</i> )
numstepsB	number of integration steps backward problem (shape <i>nt</i> )
nw	Number of common expressions
nx	number of states (alias <i>nx_rdata</i> , 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 <i>nxtrue_rdata</i> , 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 <i>nt</i> )
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_numlinsteps	number of linear steps by Newton step for steady state problem.
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_numlinsteps	number of linear steps by Newton step for steady state problem.
preeq_numsteps	number of Newton steps for steady state problem (preequilibration) [newton, simulation, newton] (length = 3)

continues on next page

Table 22 – 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 $nt \times ny$ , row-major)
rz	event trigger output (shape $nmaxevent \times nz$ , row-major)
s2llh	second-order parameter derivative of log-likelihood (shape $nJ-1 \times nplist$ , row-major)
s2rz	second-order parameter derivative of event trigger output (shape $nmaxevent \times nztrue \times nplist \times nplist$ , row-major)
sensi	sensitivity order
sensi_meth	sensitivity method
sigmay	observable standard deviation (shape $nt \times ny$ , row-major)
sigmaz	event output sigma standard deviation (shape $nmaxevent \times nz$ , row-major)
sllh	parameter derivative of log-likelihood (shape $nplist$ )
sres	parameter derivative of residual (shape $nt \times ny \times nplist$ , row-major)
srz	parameter derivative of event trigger output (shape $nmaxevent \times nz \times nplist$ , row-major)
ssigmay	parameter derivative of observable standard deviation (shape $nt \times nplist \times ny$ , row-major)
ssigmaz	parameter derivative of event output standard deviation (shape $nmaxevent \times nz$ , row-major)
status	status code
sx	parameter derivative of state (shape $nt \times nplist \times nx$ , row-major)
sx0	initial sensitivities (shape $nplist \times nx$ , row-major)
sx_ss	preequilibration sensitivities found by Newton solver (shape $nplist \times nx$ , row-major)
sy	parameter derivative of observable (shape $nt \times nplist \times ny$ , row-major)
sz	parameter derivative of event output (shape $nmaxevent \times nz$ , row-major)
ts	timepoints (shape $nt$ )
ubw	Upper bandwidth of the Jacobian
w	w data from the model (recurring terms in <code>xdot</code> , for imported SBML models from python, this contains the flux vector) (shape $nt \times nw$ , row major)
x	state (shape $nt \times nx$ , row-major)
x0	initial state (shape $nx$ )

continues on next page

Table 22 – continued from previous page

x_ss	preequilibration steady state found by Newton solver (shape $nx$ )
xdot	time derivative (shape $nx$ )
y	observable (shape $nt \times ny$ , row-major)
z	event output (shape $nmaxevent \times nz$ , 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** (`int`) – see `amici::Solver::sensi`
- **sensi\_meth** (`int`) – see `amici::Solver::sensi_meth`
- **rdrn** (`int`) – see `amici::Solver::rdata_reporting`
- **quadratic\_llh** (`boolean`) – whether model defines a quadratic nllh and computing res, sres and FIM makes sense

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. .. rubric:: Attributes .. autosummary:

```
~ReturnDataPtr.FIM
~ReturnDataPtr.J
~ReturnDataPtr.chi2
~ReturnDataPtr.cpu_time
~ReturnDataPtr.cpu_timeB
~ReturnDataPtr.lbw
~ReturnDataPtr.llh
~ReturnDataPtr.nJ
~ReturnDataPtr.ndJydy
~ReturnDataPtr.ndwdp
~ReturnDataPtr.ndwdw
~ReturnDataPtr.ndwdx
~ReturnDataPtr.ndxdotdw
~ReturnDataPtr.ne
~ReturnDataPtr.newton_maxsteps
~ReturnDataPtr.nk
~ReturnDataPtr.nmaxevent
~ReturnDataPtr.nnz
~ReturnDataPtr.np
~ReturnDataPtr.nplist
~ReturnDataPtr.nt
~ReturnDataPtr.numer_rtestfails
~ReturnDataPtr.numer_rtestfailsB
~ReturnDataPtr.numnonlinsolvconvfails
~ReturnDataPtr.numnonlinsolvconvfailsB
~ReturnDataPtr.numrhsevals
~ReturnDataPtr.numrhsevalsB
~ReturnDataPtr.numsteps
~ReturnDataPtr.numstepsB
~ReturnDataPtr.nw
~ReturnDataPtr.nx
~ReturnDataPtr.nx_rdata
~ReturnDataPtr.nx_solver
~ReturnDataPtr.nx_solver_reinit
~ReturnDataPtr.nxtrue
~ReturnDataPtr.nxtrue_rdata
~ReturnDataPtr.nxtrue_solver
~ReturnDataPtr.ny
~ReturnDataPtr.nytrue
~ReturnDataPtr.nz
~ReturnDataPtr.nztrue
~ReturnDataPtr.o2mode
~ReturnDataPtr.order
~ReturnDataPtr.posteq_cpu_time
~ReturnDataPtr.posteq_cpu_timeB
~ReturnDataPtr.posteq_numlinsteps
~ReturnDataPtr.posteq_numsteps
~ReturnDataPtr.posteq_numstepsB
~ReturnDataPtr.posteq_status
```

(continues on next page)

(continued from previous page)

```
~ReturnDataPtr.posteq_t
~ReturnDataPtr.posteq_wrms
~ReturnDataPtr.preeq_cpu_time
~ReturnDataPtr.preeq_cpu_timeB
~ReturnDataPtr.preeq_numlinsteps
~ReturnDataPtr.preeq_numsteps
~ReturnDataPtr.preeq_numstepsB
~ReturnDataPtr.preeq_status
~ReturnDataPtr.preeq_t
~ReturnDataPtr.preeq_wrms
~ReturnDataPtr.pscale
~ReturnDataPtr.rdata_reporting
~ReturnDataPtr.res
~ReturnDataPtr.rz
~ReturnDataPtr.s2llh
~ReturnDataPtr.s2rz
~ReturnDataPtr.sensi
~ReturnDataPtr.sensi_meth
~ReturnDataPtr.sigmay
~ReturnDataPtr.sigmaz
~ReturnDataPtr.sllh
~ReturnDataPtr.sres
~ReturnDataPtr.srz
~ReturnDataPtr.ssigmay
~ReturnDataPtr.ssigmaz
~ReturnDataPtr.status
~ReturnDataPtr.sx
~ReturnDataPtr.sx0
~ReturnDataPtr.sx_ss
~ReturnDataPtr.sy
~ReturnDataPtr.sz
~ReturnDataPtr.ts
~ReturnDataPtr.ubw
~ReturnDataPtr.w
~ReturnDataPtr.x
~ReturnDataPtr.x0
~ReturnDataPtr.x_ss
~ReturnDataPtr.xdot
~ReturnDataPtr.y
~ReturnDataPtr.z
```

### **amici.amici.SecondOrderMode**

**class** amici.amici.**SecondOrderMode** (*value*)  
An enumeration.



**Attributes**

directional
full
none

**amici.amici.SensitivityMethod**

**class** amici.amici.**SensitivityMethod**(*value*)  
 An enumeration.

**Attributes**

adjoint
forward
none

**amici.amici.SensitivityOrder**

**class** amici.amici.**SensitivityOrder**(*value*)  
 An enumeration.

**Attributes**

first
none
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**

<code>__init__</code> (*args)	<i>Overload 1:</i>
<code>reinitializeAllFixedParameterDependentVariables</code>	Set reinitialization of (all) states based on model constants for all simulation phases.
<code>reinitializeAllFixedParameterDependentVariablesPresimulation</code>	Set reinitialization of all states based on model constants for presimulation (only meaningful if preequilibration is performed).
<code>reinitializeAllFixedParameterDependentVariablesMainSimulation</code>	Set reinitialization of all states based on model constants for the ‘main’ simulation (only meaningful if presimulation or preequilibration is performed).

## Attributes

<code>fixedParameters</code>	Model constants
<code>fixedParametersPreequilibration</code>	Model constants for pre-equilibration
<code>fixedParametersPresimulation</code>	Model constants for pre-simulation
<code>parameters</code>	Model parameters
<code>plist</code>	Parameter indices w.r.t.
<code>pscale</code>	Parameter scales
<code>reinitialization_state_idxes_presim</code>	Indices of states to be reinitialized based on provided presimulation constants / fixed parameters.
<code>reinitialization_state_idxes_sim</code>	Indices of states to be reinitialized based on provided constants / fixed parameters.
<code>reinitializeFixedParameterInitialStates</code>	Flag indicating whether reinitialization of states depending on fixed parameters is activated
<code>sx0</code>	Initial state sensitivities
<code>t_presim</code>	Duration of pre-simulation.
<code>ts_</code>	Timepoints for which model state/outputs/.
<code>tstart_</code>	starting time
<code>x0</code>	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_idxes_presim` and `reinitialization_state_idxes_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_idxes_presim` and `reinitialization_state_idxes_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_idxes_presim` and `reinitialization_state_idxes_sim`

**Parameters** `nx_rdata` (`int`) – Number of states (`Model::nx_rdata`)

**Return type** `None`

**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 CNodeSolver 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::readSolverSettingsFromHDF5 in hdf5.cpp.

**\_\_init\_\_**(\*args, \*\*kwargs)

Initialize self. See help(type(self)) for accurate signature.

**Methods Summary**

<code>__init__(*args, **kwargs)</code>	Initialize self.
<code>clone()</code>	Clone this instance
<code>computingASA()</code>	check if ASA is being computed
<code>computingFSA()</code>	check if FSA is being computed
<code>getAbsoluteTolerance()</code>	Get the absolute tolerances for the forward problem
<code>getAbsoluteToleranceB()</code>	Returns the absolute tolerances for the backward problem for adjoint sensitivity analysis
<code>getAbsoluteToleranceFSA()</code>	Returns the absolute tolerances for the forward sensitivity problem
<code>getAbsoluteToleranceQuadratures()</code>	returns the absolute tolerance for the quadrature problem
<code>getAbsoluteToleranceSteadyState()</code>	returns the absolute tolerance for the steady state problem
<code>getAbsoluteToleranceSteadyStateSensi()</code>	returns the absolute tolerance for the sensitivities of the steady state problem
<code>getCpuTime()</code>	Reads out the CPU time needed for forward solve
<code>getCpuTimeB()</code>	Reads out the CPU time needed for backward solve
<code>getInternalSensitivityMethod()</code>	returns the internal sensitivity method
<code>getInterpolationType()</code>	<b>rtype</b> int
<code>getLastOrder()</code>	Accessor order
<code>getLinearMultistepMethod()</code>	returns the linear system multistep method
<code>getLinearSolver()</code>	<b>rtype</b> int
<code>getMaxSteps()</code>	returns the maximum number of solver steps for the forward problem
<code>getMaxStepsBackwardProblem()</code>	returns the maximum number of solver steps for the backward problem
<code>getNewtonDampingFactorLowerBound()</code>	Get a lower bound of the damping factor used in the Newton solver
<code>getNewtonDampingFactorMode()</code>	Get a state of the damping factor used in the Newton solver
<code>getNewtonMaxLinearSteps()</code>	Get maximum number of allowed linear steps per Newton step for steady state computation

continues on next page

Table 28 – continued from previous page

<code>getNewtonMaxSteps()</code>	Get maximum number of allowed Newton steps for steady state computation
<code>getNonlinearSolverIteration()</code>	returns the nonlinear system solution method
<code>getNumErrTestFails()</code>	Accessor netf
<code>getNumErrTestFailsB()</code>	Accessor netfB
<code>getNumNonlinSolvConvFails()</code>	Accessor nnlscf
<code>getNumNonlinSolvConvFailsB()</code>	Accessor nnlscfB
<code>getNumRhsEvals()</code>	Accessor nrhs
<code>getNumRhsEvalsB()</code>	Accessor nrhsB
<code>getNumSteps()</code>	Accessor ns
<code>getNumStepsB()</code>	Accessor nsB
<code>getPreequilibration()</code>	Get if model preequilibration is enabled
<code>getRelativeTolerance()</code>	Get the relative tolerances for the forward problem
<code>getRelativeToleranceB()</code>	Returns the relative tolerances for the adjoint sensitivity problem
<code>getRelativeToleranceFSA()</code>	Returns the relative tolerances for the forward sensitivity problem
<code>getRelativeToleranceQuadratures()</code>	Returns the relative tolerance for the quadrature problem
<code>getRelativeToleranceSteadyState()</code>	returns the relative tolerance for the steady state problem
<code>getRelativeToleranceSteadyStateSensi()</code>	returns the relative tolerance for the sensitivities of the steady state problem
<code>getReturnDataReportingMode()</code>	returns the ReturnData reporting mode
<code>getSensitivityMethod()</code>	Return current sensitivity method
<code>getSensitivityMethodPreequilibration()</code>	Return current sensitivity method during preequilibration
<code>getSensitivityOrder()</code>	Get sensitivity order
<code>getStabilityLimitFlag()</code>	returns stability limit detection mode
<code>getStateOrdering()</code>	Gets KLU / SuperLUMT state ordering mode
<code>gett()</code>	current solver timepoint
<code>nplist()</code>	number of parameters with which the solver was initialized
<code>nquad()</code>	number of quadratures with which the solver was initialized
<code>nx()</code>	number of states with which the solver was initialized
<code>setAbsoluteTolerance(atol)</code>	Sets the absolute tolerances for the forward problem
<code>setAbsoluteToleranceB(atol)</code>	Sets the absolute tolerances for the backward problem for adjoint sensitivity analysis
<code>setAbsoluteToleranceFSA(atol)</code>	Sets the absolute tolerances for the forward sensitivity problem
<code>setAbsoluteToleranceQuadratures(atol)</code>	sets the absolute tolerance for the quadrature problem
<code>setAbsoluteToleranceSteadyState(atol)</code>	sets the absolute tolerance for the steady state problem
<code>setAbsoluteToleranceSteadyStateSensi(atol)</code>	sets the absolute tolerance for the sensitivities of the steady state problem
<code>setInternalSensitivityMethod(ism)</code>	sets the internal sensitivity method
<code>setInterpolationType(interpType)</code>	sets the interpolation of the forward solution that is used for the backwards problem

continues on next page

Table 28 – continued from previous page

<code>setLinearMultistepMethod(lmm)</code>	sets the linear system multistep method
<code>setLinearSolver(linsol)</code>	<b>type</b> <code>linsol</code> <code>int</code>
<code>setMaxSteps(maxsteps)</code>	sets the maximum number of solver steps for the forward problem
<code>setMaxStepsBackwardProblem(maxsteps)</code>	sets the maximum number of solver steps for the backward problem
<code>setNewtonDampingFactorLowerBound(...)</code>	Set a lower bound of the damping factor in the Newton solver
<code>setNewtonDampingFactorMode(dampingFactorMode)</code>	Turn on/off a damping factor in the Newton method
<code>setNewtonMaxLinearSteps(newton_maxlinsteps)</code>	Set maximum number of allowed linear steps per Newton step for steady state computation
<code>setNewtonMaxSteps(newton_maxsteps)</code>	Set maximum number of allowed Newton steps for steady state computation
<code>setNonlinearSolverIteration(iter)</code>	sets the nonlinear system solution method
<code>setPreequilibration(require_preequilibration)</code>	Enable/disable model preequilibration
<code>setRelativeTolerance(rtol)</code>	Sets the relative tolerances for the forward problem
<code>setRelativeToleranceB(rtol)</code>	Sets the relative tolerances for the adjoint sensitivity problem
<code>setRelativeToleranceFSA(rtol)</code>	Sets the relative tolerances for the forward sensitivity problem
<code>setRelativeToleranceQuadratures(rtol)</code>	sets the relative tolerance for the quadrature problem
<code>setRelativeToleranceSteadyState(rtol)</code>	sets the relative tolerance for the steady state problem
<code>setRelativeToleranceSteadyStateSensi(rtol)</code>	sets the relative tolerance for the sensitivities of the steady state problem
<code>setReturnDataReportingMode(rdrm)</code>	sets the ReturnData reporting mode
<code>setSensitivityMethod(sensi_meth)</code>	Set sensitivity method
<code>setSensitivityMethodPreequilibration(sensi_meth)</code>	Set sensitivity method for preequilibration
<code>setSensitivityOrder(sensi)</code>	Set the sensitivity order
<code>setStabilityLimitFlag(stldet)</code>	set stability limit detection mode
<code>setStateOrdering(ordering)</code>	Sets KLU / SuperLUMT state ordering mode
<code>switchForwardSensisOff()</code>	Disable forward sensitivity integration (used in steady state sim)

## Attributes

<code>app</code>	AMICI context
------------------	---------------

## Methods

**\_\_init\_\_** (\*args, \*\*kwargs)  
Initialize self. See help(type(self)) for accurate signature.

**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** `int`

**Returns** internal sensitivity method

**getInterpolationType** () → *amici.amici.InterpolationType*

**Return type** `int`

**Returns**

**getLastOrder** () → *amici.amici.IntVector*  
Accessor order

**Return type** *IntVector*

**Returns** order

**getLinearMultistepMethod** () → *amici.amici.LinearMultistepMethod*  
returns the linear system multistep method

**Return type** `int`

**Returns** linear system multistep method

**getLinearSolver** () → *amici.amici.LinearSolver*

**Return type** `int`

**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

**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** `int`

**Returns**

**getNewtonMaxLinearSteps** () → `int`  
Get maximum number of allowed linear steps per Newton step for steady state computation

**Return type** `int`

**Returns****getNewtonMaxSteps** () → *int*

Get maximum number of allowed Newton steps for steady state computation

**Return type** *int***Returns****getNonlinearSolverIteration** () → *amici.amici.NonlinearSolverIteration*

returns the nonlinear system solution method

**Return type** *int***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 nnlsf

**Return type** *IntVector***Returns** *nnlsf***getNumNonlinSolvConvFailsB** () → *amici.amici.IntVector*

Accessor nnlsfB

**Return type** *IntVector***Returns** *nnlsfB***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

**getPreequilibration** () → *bool*

Get if model preequilibration is enabled

**Return type** *boolean*

**Returns**

**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** *int*

**Returns** ReturnData reporting mode

**getSensitivityMethod** () → *amici.amici.SensitivityMethod*

Return current sensitivity method

**Return type** *int*

**Returns** method enum

**getSensitivityMethodPreequilibration** () → *amici.amici.SensitivityMethod*

Return current sensitivity method during preequilibration

**Return type** *int*

**Returns** method enum

**getSensitivityOrder** () → *amici.amici.SensitivityOrder*

Get sensitivity order

**Return type** *int*

**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).

**gett** () → *float*

current solver timepoint

**Return type** *float*

**Returns** *t*

**np1ist** () → *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*

**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*

**setNewtonMaxLinearSteps** (*newton\_maxlinsteps: int*) → *None*

Set maximum number of allowed linear steps per Newton step for steady state computation

**Parameters** *newton\_maxlinsteps* (*int*) –

**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*

**setNonlinearSolverIteration** (*iter: amici.amici.NonlinearSolverIteration*) → *None*

sets the nonlinear system solution method

**Parameters** *iter* (*amici.amici.NonlinearSolverIteration*) – nonlinear system solution method

**Return type** *None*

**setPreequilibration** (*require\_preequilibration: bool*) → *None*

Enable/disable model preequilibration

**Parameters** *require\_preequilibration* (*bool*) –

**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`

**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*

**switchForwardSensisOff** () → *None*

Disable forward sensitivity integration (used in steady state sim)

**Return type** *None*

### **amici.amici.SolverPtr**

**class** amici.amici.**SolverPtr** (\*args)

Swig-Generated class that implements smart pointers to Solver as objects. .. rubric:: Attributes .. autosummary:

~SolverPtr.app
----------------

### **amici.amici.SteadyStateSensitivityMode**

**class** amici.amici.**SteadyStateSensitivityMode** (*value*)

An enumeration.

#### **Attributes**

newtonOnly
------------

simulationFSA
---------------

### **amici.amici.SteadyStateStatus**

**class** amici.amici.**SteadyStateStatus** (*value*)

An enumeration.

#### **Attributes**

failed
--------

failed_convergence
--------------------

failed_damping
----------------

failed_factorization
----------------------

failed_too_long_simulation
----------------------------

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**

<code>backtraceString(maxFrames)</code>	Returns the current backtrace as <code>std::string</code>
<code>compiledWithOpenMP()</code>	<b>rtype</b> <code>bool</code>
<code>enum(prefix)</code>	
<code>getScaledParameter(unscaledParameter, scaling)</code>	Apply parameter scaling according to <i>scaling</i>
<code>getUnscaledParameter(scaledParameter, scaling)</code>	Remove parameter scaling according to <i>scaling</i>
<code>parameterScalingFromIntVector(intVec)</code>	<b>rtype</b> <code>amici.amici.ParameterScalingVector</code>
<code>runAmiciSimulation(solver, edata, model[, ...])</code>	Core integration routine.
<code>runAmiciSimulations(solver, edatas, model, ...)</code>	Same as <code>runAmiciSimulation</code> , but for multiple <code>ExpData</code> instances.
<code>scaleParameters(bufferUnscaled, pscale, ...)</code>	Apply parameter scaling according to <i>scaling</i>
<code>unscaleParameters(bufferScaled, pscale, ...)</code>	Remove parameter scaling according to the parameter scaling in <i>pscale</i>

**Functions**

```
amici.amici.backtraceString(maxFrames: int) → str
```

Returns the current backtrace as `std::string`

**Parameters** `maxFrames (int)` – Number of frames to include

**Return type** `string`

**Returns** Backtrace

```
amici.amici.compiledWithOpenMP() → bool
```

**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*

**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.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
- **\_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, 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

<code>__init__(sbml_source[, show_sbml_warnings, ...])</code>	Create a new Model instance.
<code>add_d_dt(d_dt, variable, variable0, name)</code>	Creates or modifies species, to implement rate rules for compartments and species, respectively.
<code>add_local_symbol(key, value)</code>	Add local symbols with some sanity checking for duplication which would indicate redefinition of internals, which SBML permits, but we don't.
<code>check_event_support()</code>	Check possible events in the model, as AMICI does currently not support
<code>check_support()</code>	Check whether all required SBML features are supported.
<code>is_assignment_rule_target(element)</code>	Checks if an element has a valid assignment rule in the specified model.
<code>is_rate_rule_target(element)</code>	Checks if an element has a valid assignment rule in the specified model.
<code>process_conservation_laws(ode_model, ...)</code>	Find conservation laws in reactions and species.
<code>sbml2amici([model_name, output_dir, ...])</code>	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, 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, volume\_updates)**

Find conservation laws in reactions and species.

**Parameters**

- **ode\_model** – ODEModel object with basic definitions
- **volume\_updates** – List with updates for the stoichiometric matrix accounting for compartment volumes

**Returns** **volume\_updates\_solver** List (according to reduced stoichiometry) with updates for the stoichiometric matrix accounting for compartment volumes

**Return type** `typing.List`

**sbml2amici** (`model_name=None, output_dir=None, observables=None, constant_parameters=None, sigmas=None, noise_distributions=None, verbose=40, assume_pow_positivity=False, compiler=None, allow_reinit_fixpar_initcond=True, compile=True, compute_conservation_laws=True, simplify=<function SbmlImporter.<lambda>>, log_as_log10=True, generate_sensitivity_code=True, **kwargs`)

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** (`typing.Optional[str]`) – name of the model/model directory
- **output\_dir** (`typing.Optional[str]`) – 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
- **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)
- **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.
- **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.
- **simplify** (`typing.Callable`) – see `ODEModel._simplify`
- **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

<code>assignmentRules2observables(sbml_model[, ...])</code>	Turn assignment rules into observables.
<code>get_species_initial(species)</code>	Extract the initial concentration from a given species
<code>grouper(iterable, n[, fillvalue])</code>	Collect data into fixed-length chunks or blocks
<code>replace_logx(math_str)</code>	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.grouper` (*iterable, n, fillvalue=None*)

Collect data into fixed-length chunks or blocks

`grouper(‘ABCDEFGF’, 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.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

<code>extract_monomers(complex_patterns)</code>	Constructs a list of monomer names contained in complex patterns.
<code>has_fixed_parameter_ic(specie, pysb_model, ...)</code>	Wrapper to interface <code>ode_export.ODEModel.state_has_fixed_parameter_initial_condition()</code> from a pysb specie/model arguments
<code>ode_model_from_pysb_importer(model[, ...])</code>	Creates an <code>amici.ODEModel</code> instance from a <code>pysb.Model</code> instance.
<code>pysb2amici(model[, output_dir, observables, ...])</code>	Generate AMICI C++ files for the provided model.
<code>pysb_model_from_path(pysb_model_file)</code>	Load a pysb model module and return the <code>pysb.Model</code> 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>, 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`
- **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>>, generate_sensitivity_code=True)
```

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.Optional[str]`) – 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`
- **generate\_sensitivity\_code** (`bool`) – if set to False, code for sensitivity computation will not be generated

`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` (`str`) – Full or relative path to the PySB model module

**Return type** `pysb.core.Model`

**Returns** The pysb Model instance

## 10.4.5 amici.petab\_import

### PEtab Import

Import a model in the `petab` (<https://github.com/PEtab-dev/PEtab>) format into AMICI.

### Functions Summary

<code>constant_species_to_parameters(sbml_model)</code>	Convert constant species in the SBML model to constant parameters.
<code>get_fixed_parameters(sbml_model[, ...])</code>	Determine, set and return fixed model parameters.
<code>get_observation_model(observable_df)</code>	Get observables, sigmas, and noise distributions from PETA tab observation table in a format suitable for <code>amici.sbml_import.SbmlImporter.sbml2amici()</code> .
<code>import_model(sbml_model[, condition_table, ...])</code>	Create AMICI model from PETA tab problem
<code>import_model_sbml(sbml_model[, ...])</code>	Create AMICI model from PETA tab problem
<code>import_petab_problem(petab_problem[, ...])</code>	Import model from petab problem.

continues on next page

Table 37 – continued from previous page

<code>main()</code>	Command line interface to import a model in the PETab ( <a href="https://github.com/PETab-dev/PETab/">https://github.com/PETab-dev/PETab/</a> ) format into AMICI.
<code>parse_cli_args()</code>	Parse command line arguments
<code>petab_noise_distributions_to_amici(observable_df)</code>	Map from the petab to the amici format of noise distribution identifiers.
<code>petab_scale_to_amici_scale(scale_str)</code>	Convert PETab parameter scaling string to AMICI scaling integer
<code>show_model_info(sbml_model)</code>	Log some model quantities
<code>species_to_parameters(species_ids, sbml_model)</code>	Turn a SBML species into parameters and replace species references inside the model instance.

## Functions

`amici.petab_import.constant_species_to_parameters(sbml_model)`

Convert constant species in the SBML model to constant parameters.

This can be used e.g. for setting up models with condition-specific constant species for PETab, since there it is not possible to specify constant species in the condition table.

**Parameters** `sbml_model` (`libsbml.Model`) – SBML Model

**Return type** `typing.List[str]`

**Returns** List of IDs of SBML species that have been turned into constants

`amici.petab_import.get_fixed_parameters(sbml_model, condition_df=None, const_species_to_parameters=False)`

Determine, set and return fixed model parameters.

Parameters specified in `condition_df` are turned into constants. Only global SBML parameters are considered. Local parameters are ignored.

**Parameters**

- **condition\_df** (`typing.Optional[pandas.core.frame.DataFrame]`) – PETab condition table. If provided, the respective parameters will be turned into AMICI constant parameters.
- **sbml\_model** (`libsbml.Model`) – `libsbml.Model` instance
- **const\_species\_to\_parameters** (`bool`) – If `True`, species which are marked constant within the SBML model will be turned into constant parameters *within* the given `sbml_model`.

**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, condition_table=None, observable_table=None,
                                measurement_table=None, model_name=None,
                                model_output_dir=None, verbose=True, allow_reinit_fixpar_initcond=True, **kwargs)
```

Create AMICI model from PETab problem

#### Parameters

- **sbml\_model** (`typing.Union[str, libsbml.Model]`) – PETab SBML model or SBML file name.
- **condition\_table** (`typing.Union[str, 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).
- **observable\_table** (`typing.Union[str, pandas.core.frame.DataFrame, None]`) – PETab observable table.
- **measurement\_table** (`typing.Union[str, pandas.core.frame.DataFrame, None]`) – PETab measurement table.
- **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.Optional[str]`) – 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.
- **kwargs** – Additional keyword arguments to be passed to `amici.sbml_import.SbmlImporter.sbml2amici()`.

Return type `None`

```
amici.petab_import.import_model_sbml(sbml_model, condition_table=None, observable_table=None,
                                      measurement_table=None, model_name=None, model_output_dir=None,
                                      verbose=True, allow_reinit_fixpar_initcond=True, **kwargs)
```

Create AMICI model from PETab problem

#### Parameters

- **sbml\_model** (`typing.Union[str, libsbml.Model]`) – PETab SBML model or SBML file name.
- **condition\_table** (`typing.Union[str, 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).
- **observable\_table** (`typing.Union[str, pandas.core.frame.DataFrame, None]`) – PETab observable table.
- **measurement\_table** (`typing.Union[str, pandas.core.frame.DataFrame, None]`) – PETab measurement table.
- **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.Optional[str]`) – 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.
- **kwargs** – Additional keyword arguments to be passed to `amici.sbml_import.SbmlImporter.sbml2amici()`.

**Return type** `None`

```
amici.petab_import.import_petab_problem(petab_problem,          model_output_dir=None,
                                         model_name=None,        force_compile=False,
                                         **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.Optional[str]`) – 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 SBML 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.
- **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.parse_cli_args()
```

Parse command line arguments

**Returns** Parsed CLI arguments from `argparse`.

```
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`

**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

## 10.4.6 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

<code>PysbPetabProblem([pysb_model])</code>	Representation of a PySB-model-based PÉtab problem
---	--

### `amici.petab_import_pysb.PysbPetabProblem`

**class** `amici.petab_import_pysb.PysbPetabProblem(pysb_model=None, *args, **kwargs)`

Representation of a PySB-model-based PÉtab problem

This class extends `petab.Problem` with a PySB model. The model is augmented with the observation model based on the PÉtab observable table. For now, a dummy SBML model is created which allows used the existing SBML-PÉtab API.

**Variables** `pysb_model` – PySB model instance from of this PÉtab problem.

**\_\_init\_\_** (`pysb_model=None, *args, **kwargs`)

Constructor

#### Parameters

- **pysb\_model** (`typing.Optional[pysb.core.Model]`) – PySB model instance for this PÉtab problem
- **args** – See `petab.Problem.__init__()`
- **kwargs** – See `petab.Problem.__init__()`

## Methods Summary

<code>__init__([pysb_model])</code>	Constructor
<code>create_parameter_df(*args, **kwargs)</code>	Create a new PETab parameter table
<code>from_combine(filename)</code>	Read PETab COMBINE archive ( <a href="http://co.mbine.org/documents/archive">http://co.mbine.org/documents/archive</a> ).
<code>from_files([condition_file, ...])</code>	Factory method to load model and tables from files.
<code>from_folder(folder[, model_name])</code>	Factory method to use the standard folder structure and file names, i.e.
<code>from_yaml(yaml_config)</code>	Factory method to load model and tables as specified by YAML file.
<code>get_lb([free, fixed, scaled])</code>	Generic function to get lower parameter bounds.
<code>get_model_parameters()</code>	See <code>petab.sbml.get_model_parameters</code>
<code>get_noise_distributions()</code>	See <code>get_noise_distributions</code> .
<code>get_observable_ids()</code>	Returns dictionary of observable ids.
<code>get_observables([remove])</code>	Returns dictionary of observables definitions.
<code>get_optimization_parameter_scales()</code>	Return list of optimization parameter scaling strings.
<code>get_optimization_parameters()</code>	Return list of optimization parameter IDs.
<code>get_optimization_to_simulation_parameter_mapping([remove])</code>	See <code>get_simulation_to_optimization_parameter_mapping</code> .
<code>get_sigmas([remove])</code>	Return dictionary of observableId => sigma as defined in the SBML model.
<code>get_simulation_conditions_from_measurement_df(measurement_df)</code>	See <code>petab.get_simulation_conditions</code>
<code>get_ub([free, fixed, scaled])</code>	Generic function to get upper parameter bounds.
<code>get_x_ids([free, fixed])</code>	Generic function to get parameter ids.
<code>get_x_nominal([free, fixed, scaled])</code>	Generic function to get parameter nominal values.
<code>sample_parameter_startpoints([n_starts])</code>	Create starting points for optimization
<code>to_files([sbml_file, condition_file, ...])</code>	Write PETab tables to files for this problem

## Attributes

<code>lb</code>	Parameter table lower bounds.
<code>lb_scaled</code>	Parameter table lower bounds with applied parameter scaling
<code>ub</code>	Parameter table upper bounds
<code>ub_scaled</code>	Parameter table upper bounds with applied parameter scaling
<code>x_fixed_ids</code>	Parameter table parameter IDs, for fixed parameters.
<code>x_fixed_indices</code>	Parameter table non-estimated parameter indices.
<code>x_free_ids</code>	Parameter table parameter IDs, for free parameters.
<code>x_free_indices</code>	Parameter table estimated parameter indices.
<code>x_ids</code>	Parameter table parameter IDs
<code>x_nominal</code>	Parameter table nominal values
<code>x_nominal_fixed</code>	Parameter table nominal values, for fixed parameters.
<code>x_nominal_fixed_scaled</code>	Parameter table nominal values with applied parameter scaling, for fixed parameters.
<code>x_nominal_free</code>	Parameter table nominal values, for free parameters.
<code>x_nominal_free_scaled</code>	Parameter table nominal values with applied parameter scaling, for free parameters.

continues on next page



Table 40 – continued from previous page

<code>x_nominal_scaled</code>	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 `create_combine_archive`.

**Parameters** **filename** (`str`) – Path to the PETab-COMBINE archive

**Return type** `petab.problem.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*)

Factory method to load model and tables from files.

### Parameters

- **condition\_file** (`typing.Optional[str]`) – PETab condition table
- **measurement\_file** (`typing.Union[str, typing.Iterable[str], None]`) – PETab measurement table
- **parameter\_file** (`typing.Union[str, typing.List[str], None]`) – PETab parameter table
- **visualization\_files** (`typing.Union[str, typing.Iterable[str], None]`) – PETab visualization tables
- **observable\_files** (`typing.Union[str, typing.Iterable[str], None]`) – PETab observables tables
- **pysb\_model\_file** (`typing.Optional[str]`) – PySB model file

**Return type** `amici.petab_import_pysb.PysbPetabProblem`

`static from_folder` (*folder, model\_name=None*)

Factory method to use the standard folder structure and file names, i.e.

```
${model_name}/  
+-- experimentalCondition_${model_name}.tsv  
+-- measurementData_${model_name}.tsv  
+-- model_${model_name}.xml  
+-- parameters_${model_name}.tsv
```

### Parameters

- **folder** (`str`) – Path to the directory in which the files are located.
- **model\_name** (`typing.Optional[str]`) – If specified, overrides the model component in the file names. Defaults to the last component of `folder`.

**Return type** `petab.problem.Problem`

**static from\_yaml** (`yaml_config`)

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, str]`) – PETab configuration as dictionary or YAML file name

**Return type** `amici.petab_import_pysb.PysbPetabProblem`

**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.

**Returns** The lower parameter bounds.

**Return type** `v`

**get\_model\_parameters** ()

See `petab.sbml.get_model_parameters`

**get\_noise\_distributions** ()

See `get_noise_distributions`.

**get\_observable\_ids** ()

Returns dictionary of observable ids.

**get\_observables** (`remove=False`)

Returns dictionary of observables definitions. See `assignment_rules_to_dict` for details.

**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** (*warn\_unmapped=True*,  
*scaled\_parameters=False*)

See `get_simulation_to_optimization_parameter_mapping`.

**get\_sigmas** (*remove=False*)

Return dictionary of observableId => sigma as defined in the SBML model. This does not include parameter mappings defined in the measurement table.

**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.

**Returns** The upper parameter bounds.

**Return type** `v`

**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.

**Returns** The parameter ids.

**Return type** `v`

**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.

**Returns** The parameter nominal values.

**Return type** `v`

**sample\_parameter\_startpoints** (*n\_starts=100*)

Create starting points for optimization

See `sample_parameter_startpoints`

**to\_files** (*sbml\_file=None*, *condition\_file=None*, *measurement\_file=None*, *parameter\_file=None*, *visualization\_file=None*, *observable\_file=None*, *yaml\_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.Optional[str]`) – SBML model destination
- **condition\_file** (`typing.Optional[str]`) – Condition table destination
- **measurement\_file** (`typing.Optional[str]`) – Measurement table destination
- **parameter\_file** (`typing.Optional[str]`) – Parameter table destination
- **visualization\_file** (`typing.Optional[str]`) – Visualization table destination
- **observable\_file** (`typing.Optional[str]`) – Observables table destination
- **yaml\_file** (`typing.Optional[str]`) – YAML file destination

#### Raises

- **ValueError** – If a destination was provided for a non-existing
- **entity.** –

**Return type** `None`

## Functions Summary

<code>create_dummy_sbml(pysb_model[, observable_ids])</code>	<code>observable_ids</code>	Create SBML dummy model for to use PySB models with PETab.
<code>import_model_pysb(petab_problem[, ...])</code>		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, **kwargs)`

Create AMICI model from PySB-PETab problem

#### Parameters

- **petab\_problem** (`amici.petab_import_pysb.PysbPetabProblem`) – PySB PETab problem

- **model\_output\_dir** (`typing.Optional[str]`) – 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.
- **kwargs** – Additional keyword arguments to be passed to `amici.pysb_import.pysb2amici()`.

Return type `None`

## 10.4.7 amici.petab\_objective

### PEtab Objective

Functionality related to running simulations or evaluating the objective function as defined by a PEPtab problem

### Functions Summary

<code>aggregate_sllh(amici_model, rdatas, ...)</code>	Aggregate likelihood gradient for all conditions, according to PEPtab parameter mapping.
<code>create_edata_for_condition(condition, ...)</code>	Get <code>amici.amici.ExpData</code> for the given PEPtab condition.
<code>create_edatas(amici_model, petab_problem[, ...])</code>	Create list of <code>amici.amici.ExpData</code> objects for PEPtab problem.
<code>create_parameter_mapping(petab_problem, ...)</code>	Generate AMICI specific parameter mapping.
<code>create_parameter_mapping_for_condition(condition, ...)</code>	Generate AMICI specific parameter mapping for condition.
<code>create_parameterized_edatas(amici_model, ...)</code>	Create list of <code>amici.ExpData</code> objects with parameters filled in.
<code>rdatas_to_measurement_df(rdatas, model, ...)</code>	Create a measurement dataframe in the PEPtab format from the passed <code>rdatas</code> and own information.
<code>rdatas_to_simulation_df(rdatas, model, ...)</code>	Create a PEPtab simulation dataframe from <code>amici.ReturnData</code> .
<code>simulate_petab(petab_problem, amici_model[, ...])</code>	Simulate PEPtab model.
<code>subset_dict(full, *args)</code>	Get subset of dictionary based on provided keys

### Functions

`amici.petab_objective.aggregate_sllh(amici_model, rdatas, parameter_mapping)`

Aggregate likelihood gradient for all conditions, according to PEPtab parameter mapping.

#### Parameters

- **amici\_model** (`typing.Union[amici.amici.Model, amici.amici.ModelPtr]`) – AMICI model from which `rdatas` were obtained.
- **rdatas** (`typing.Sequence[amici.numpy.ReturnDataView]`) – Simulation results.
- **parameter\_mapping** (`typing.Optional[amici.parameter_mapping.ParameterMapping]`) –

**PETab parameter mapping to condition-specific** simulation parameters

**Return type** `typing.Optional[typing.Dict[str, float]]`

**Returns** aggregated `sllh`

`amici.petab_objective.create_edata_for_condition(condition, 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`.
- **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.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 been obtained before.

**Return type** `typing.List[amici.ExpData]`

**Returns** List with one `amici.amici.ExpData` per simulation condition, with filled in time-points and data.

`amici.petab_objective.create_parameter_mapping(petab_problem, simulation_conditions, scaled_parameters, amici_model)`

Generate AMICI specific parameter mapping.

**Parameters**

- **petab\_problem** (`petab.problem.Problem`) – PETab problem
- **simulation\_conditions** (`typing.Union[pandas.core.frame.DataFrame, 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.

**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.ExpData]`

**Returns** List with one `amici.amici.ExpData` per simulation condition, with filled in time-points, 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.ReturnDatas*.

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)`

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 been obtained before. Not required if *edatas* and *parameter\_mapping* are provided.
- **edatas** (`typing.Optional[typing.List[typing.Union[amici.amici.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.



**Return type** `typing.Dict[str, typing.Any]`

**Returns**

Dictionary of

- cost function value (LLH),
- const function sensitivity w.r.t. parameters (SLLH), (**NOTE:** Sensitivities are computed for the scaled parameters)
- list of *ReturnData* (RDATAS),

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

## 10.4.8 amici.import\_utils

Miscellaneous functions related to model import, independent of any specific model format

### Functions Summary

<code>noise_distribution_to_cost_function(...)</code>	Parse noise distribution string to a cost function definition amici can work with.
<code>smart_subs(element, old, new)</code>	Optimized substitution that checks whether anything needs to be done first
<code>smart_subs_dict(sym, subs[, field, reverse])</code>	Substitutes expressions completely flattening them out.
<code>toposort_symbols(symbols[, field])</code>	Topologically sort symbol definitions according to their interdependency

### Functions

`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  $\sigma$  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.  $\log_{10}(m)$  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.  $\log_{10}(m)$  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  $p = \sigma$ :

$$\pi(m|y, \sigma) = \text{Heaviside}(y - m) \cdot \frac{\Gamma(y + 1)}{\Gamma(m + 1)\Gamma(y - m + 1)} \sigma^m (1 - \sigma)^{(y-m)}$$

- ‘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` (`str`) – 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.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`) – substitution 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.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 interdependency

**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

## 10.4.9 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

<code>ConservationLaw(identifier, name, value)</code>	A conservation law defines the absolute the total amount of a (weighted) sum of states
<code>Constant(identifier, name, value)</code>	A Constant is a fixed variable in the model with respect to which sensitivities cannot be computed, abbreviated by <i>k</i>
<code>Event(identifier, name, value, state_update, ...)</code>	An Event defines either a SBML event or a root of the argument of a Heaviside function.
<code>Expression(identifier, name, value)</code>	An Expressions is a recurring elements in symbolic formulas.
<code>LogLikelihood(identifier, name, value)</code>	A LogLikelihood defines the distance between measurements and experiments for a particular observable.
<code>ModelQuantity(identifier, name, value)</code>	Base class for model components
<code>ODEExporter(ode_model[, outdir, verbose, ...])</code>	The ODEExporter class generates AMICI C++ files for ODE model as defined in symbolic expressions.
<code>ODEModel([verbose, simplify])</code>	Defines an Ordinary Differential Equation as set of ModelQuantities.
<code>Observable(identifier, name, value[, ...])</code>	An Observable links model simulations to experimental measurements, abbreviated by <i>y</i>
<code>Parameter(identifier, name, value)</code>	A Parameter is a free variable in the model with respect to which sensitivities may be computed, abbreviated by <i>p</i>
<code>SigmaY(identifier, name, value)</code>	A Standard Deviation SigmaY rescales the distance between simulations and measurements when computing residuals or objective functions, abbreviated by <i>sigmay</i>
<code>State(identifier, name, init, dt)</code>	A State variable defines an entity that evolves with time according to the provided time derivative, abbreviated by <i>x</i>
<code>TemplateAmici(template)</code>	Template format used in AMICI (see string.template for more details).

**amici.ode\_export.ConservationLaw**

**class** amici.ode\_export.**ConservationLaw** (*identifier, name, value*)

A conservation law defines the absolute the total amount of a (weighted) sum of states

**\_\_init\_\_** (*identifier, name, value*)

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)

**Methods Summary**

<code>__init__</code> ( <i>identifier, name, value</i> )	Create a new ConservationLaw instance.
<code>get_id</code> ()	ModelQuantity identifier
<code>get_name</code> ()	ModelQuantity name
<code>get_val</code> ()	ModelQuantity value
<code>set_val</code> ( <i>val</i> )	Set ModelQuantity value

**Methods**

**\_\_init\_\_** (*identifier, name, value*)

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)

**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\_export.Constant**

**class** amici.ode\_export.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**

<code>__init__</code> ( <i>identifier, name, value</i> )	Create a new Expression instance.
<code>get_id</code> ()	ModelQuantity identifier
<code>get_name</code> ()	ModelQuantity name
<code>get_val</code> ()	ModelQuantity value
<code>set_val</code> ( <i>val</i> )	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\_export.Event

**class** `amici.ode_export.Event` (*identifier, name, value, state\_update, event\_observable*)

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, event\_observable*)

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)
- **event\_observable** (`typing.Optional[sympy.core.expr.Expr]`) – formula a potential observable linked to the event (None for Heaviside functions, empty events without observable)

## Methods Summary

<code>__init__(identifier, name, value, ...)</code>	Create a new Event instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	Set ModelQuantity value

## Methods

**\_\_init\_\_** (*identifier, name, value, state\_update, event\_observable*)

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)

- **event\_observable** (`typing.Optional[sympy.core.expr.Expr]`) – formula a potential observable linked to the event (None for Heaviside functions, empty events without observable)

**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\_export.Expression**

**class** `amici.ode_export.Expression` (*identifier, name, value*)

An Expressions 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**

<code>__init__(identifier, name, value)</code>	Create a new Expression instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	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\_export.LogLikelihood

**class** `amici.ode_export.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

<code>__init__(identifier, name, value)</code>	Create a new Expression instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	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\_export.ModelQuantity**

**class** amici.ode\_export.**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**

<code>__init__</code> ( <i>identifier, name, value</i> )	Create a new ModelQuantity instance.
<code>get_id</code> ()	ModelQuantity identifier
<code>get_name</code> ()	ModelQuantity name
<code>get_val</code> ()	ModelQuantity value
<code>set_val</code> ( <i>val</i> )	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\_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)
```

The ODEExporter class generates AMICI C++ files for ODE model as defined in symbolic expressions.

#### **Variables**

- **model** – ODE definition
- **outdir** – see `amici.ode_export.ODEExporter.set_paths()`
- **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

```
__init__ (ode_model, outdir=None, verbose=False, assume_pow_positivity=False, compiler=None, allow_reinit_fixpar_initcond=True, generate_sensitivity_code=True)
```

Generate AMICI C++ files for the ODE provided to the constructor.

#### **Parameters**

- **ode\_model** (`amici.ode_export.ODEModel`) – ODE definition
- **outdir** (`typing.Optional[str]`) – 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`

:param **generate\_sensitivity\_code** specifies whether code required for sensitivity computation will be generated

## Methods Summary

<code>__init__</code> ( <code>ode_model</code> [, <code>outdir</code> , <code>verbose</code> , ...])	Generate AMICI C++ files for the ODE provided to the constructor.
<code>compile_model</code> ()	Compiles the generated code it into a simulatable module
<code>generate_model_code</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
<code>set_name</code> ( <code>model_name</code> )	Sets the model name
<code>set_paths</code> ( <code>output_dir</code> )	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`)  
Generate AMICI C++ files for the ODE provided to the constructor.

### Parameters

- **ode\_model** (`amici.ode_export.ODEModel`) – ODE definition
- **outdir** (`typing.Optional[str]`) – 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`

:param **generate\_sensitivity\_code** specifies whether code required for sensitivity computation will be generated

`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*)

Set output paths for the model and create if necessary

**Parameters** **output\_dir** (`str`) – relative or absolute path where the generated model code is to be placed. will be created if does not exists.

**Return type** `None`

### **amici.ode\_export.ODEModel**

**class** `amici.ode_export.ODEModel` (*verbose=False, simplify=<function powsimp>*)

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
- **\_sigmayes** – list of sigmayes
- **\_parameters** – list of parameters
- **\_loglikelihoods** – list of loglikelihoods
- **\_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
- **\_strippedsyms** – carries symbolic identifiers that were stripped of additional class information
- **\_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.

**`__init__`** (*verbose=False, simplify=<function powsimp>*)

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()`

#### Methods Summary

<code>__init__</code> ( <i>verbose, simplify</i> )	Create a new <code>ODEModel</code> instance.
<code>add_component</code> ( <i>component[, insert_first]</i> )	Adds a new <code>ModelQuantity</code> to the model.
<code>add_conservation_law</code> ( <i>state, total_abundance, ...</i> )	to- Adds a new conservation law to the model.
<code>colptrs</code> ( <i>name</i> )	Returns (and constructs if necessary) the column pointers for a sparsified symbolic variable.
<code>conservation_law_has_multispecies</code> ( <i>tcl</i> )	Checks whether a conservation law has multiple species or it just defines one constant species
<code>eq</code> ( <i>name</i> )	Returns (and constructs if necessary) the formulas for a symbolic entity.
<code>free_symbols</code> ()	Returns list of free symbols that appear in ODE rhs and initial conditions.
<code>generate_basic_variables</code> ( <i>*[, from_sbml]</i> )	Generates the symbolic identifiers for all variables in <code>ODEModel.variable_prototype</code>

continues on next page

Table 52 – continued from previous page

<code>get_appearance_counts(idxs)</code>		Counts how often a state appears in the time derivative of another state and expressions for a subset of states
<code>get_conservation_laws()</code>		Returns a list of states with conservation law set
<code>import_from_sbml_importer(si[, compute_cls])</code>	com-	Imports a model specification from a <code>amici.sbml_import.SbmlImporter</code> instance.
<code>name(name)</code>		Returns (and constructs if necessary) the names of a symbolic variable
<code>num_cons_law()</code>		Number of conservation laws.
<code>num_const()</code>		Number of Constants.
<code>num_events()</code>		Number of Events.
<code>num_expr()</code>		Number of Expressions.
<code>num_obs()</code>		Number of Observables.
<code>num_par()</code>		Number of Parameters.
<code>num_state_reinits()</code>		Number of solver states which would be reinitialized after preequilibration
<code>num_states_rdata()</code>		Number of states.
<code>num_states_solver()</code>		Number of states after applying conservation laws.
<code>parse_events()</code>		This functions 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
<code>rowvals(name)</code>		Returns (and constructs if necessary) the row values for a sparsified symbolic variable.
<code>sparseseq(name)</code>		Returns (and constructs if necessary) the sparsified formulas for a sparsified symbolic variable.
<code>sparsesym(name[, force_generate])</code>		Returns (and constructs if necessary) the sparsified identifiers for a sparsified symbolic variable.
<code>state_has_conservation_law(ix)</code>		Checks whether the state at specified index has a conservation law set
<code>state_has_fixed_parameter_initial_condition(ix)</code>		Checks whether the state at specified index has a fixed parameter initial condition
<code>state_is_constant(ix)</code>		Checks whether the temporal derivative of the state is zero
<code>sym(name[, stripped])</code>		Returns (and constructs if necessary) the identifiers for a symbolic entity.
<code>sym_names()</code>		Returns a list of names of generated symbolic variables
<code>sym_or_eq(name, varname)</code>		Returns symbols or equations depending on whether a given variable appears in the function signature or not.
<code>val(name)</code>		Returns (and constructs if necessary) the numeric values of a symbolic entity



## Methods

**\_\_init\_\_** (*verbose=False, simplify=<function powsimp>*)

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()`

**add\_component** (*component, insert\_first=False*)

Adds a new ModelQuantity to the model.

### Parameters

- **component** (`amici.ode_export.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, state\_expr, abundance\_expr*)

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 ( $T/a_j$ )
- **state\_expr** (`sympy.core.expr.Expr`) – symbolic algebraic formula that replaces the the state. This is used to compute the numeric value of of *state* during simulations.  $x_j = T/a_j - \sum_{ij} (a_i * x_i)/a_j$
- **abundance\_expr** (`sympy.core.expr.Expr`) – symbolic algebraic formula that computes the value of the conserved quantity. This is used to update the numeric value for *total\_abundance* after (re-)initialization.  $T/a_j = \sum_{ij} (a_i * x_i)/a_j + x_j$

**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_export.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** (\*, *from\_sbml=False*)

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.basic.Basic]]`

**Returns** list of state identifiers

**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

**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\_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 functions 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*, *stripped=False*)

Returns (and constructs if necessary) the identifiers for a symbolic entity.

**Parameters**

- **name** (`str`) – name of the symbolic variable
- **stripped** (`typing.Optional[bool]`) – should additional class information be stripped from the symbolic variables (default=False)

**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.Observable

**class** `amici.ode_export.Observable` (*identifier*, *name*, *value*, *measurement\_symbol=None*)

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

**\_\_init\_\_** (*identifier*, *name*, *value*, *measurement\_symbol=None*)

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

## Methods Summary

<code>__init__(identifier, name, value[, ...])</code>	Create a new Observable instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_measurement_symbol()</code>	<b>rtype</b> <code>sympy.core.symbol.Symbol</code>
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	Set ModelQuantity value

## Methods

**\_\_init\_\_** (*identifier, name, value, measurement\_symbol=None*)

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

**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\_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_export.Parameter`

**class** `amici.ode_export.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

<code>__init__(identifier, name, value)</code>	Create a new Expression instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	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_export.SigmaY`

**class** `amici.ode_export.SigmaY(identifier, name, value)`

A Standard Deviation SigmaY rescales the distance between simulations and measurements when computing residuals or objective functions, abbreviated by *sigmay*

`__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

<code>__init__(identifier, name, value)</code>	Create a new Standard Deviation instance.
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_val(val)</code>	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\_export.State**

**class** amici.ode\_export.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**

<code>__init__(identifier, name, init, dt)</code>	Create a new State instance.
<code>get_dt()</code>	Gets the time derivative
<code>get_free_symbols()</code>	Gets the set of free symbols in time derivative and initial conditions
<code>get_id()</code>	ModelQuantity identifier
<code>get_name()</code>	ModelQuantity name
<code>get_val()</code>	ModelQuantity value
<code>set_conservation_law(law)</code>	Sets the conservation law of a state.
<code>set_dt(dt)</code>	Sets the time derivative
<code>set_val(val)</code>	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\_free\_symbols()**

Gets the set of free symbols in time derivative and initial conditions

**Return type** `typing.Set[sympy.core.symbol.Symbol]`**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**set\_conservation\_law(law)**

Sets the conservation law of a state. If the a conservation law is set, the respective state will be replaced by an algebraic formula according to the respective conservation law.

**Parameters** **law** (`sympy.core.expr.Expr`) – 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

### **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*)Initialize self. See `help(type(self))` for accurate signature.

## Methods Summary

<code>__init__(template)</code>	Initialize self.
<code>safe_substitute([mapping])</code>	
<code>substitute([mapping])</code>	

## Attributes

<code>braceidpattern</code>
<code>delimiter</code>
<code>flags</code>
<code>idpattern</code>
<code>pattern</code>

## Methods

`__init__(template)`  
Initialize self. See `help(type(self))` for accurate signature.

`safe_substitute(mapping={}, /, **kws)`

`substitute(mapping={}, /, **kws)`

## Functions Summary

<code>apply_template(source_file, target_file, ...)</code>	Load source file, apply template substitution as provided in <code>templateData</code> and save as <code>targetFile</code> .
<code>cast_to_sym(value, input_name)</code>	Typecasts the value to <code>sympy.Float</code> if possible, and ensures the value is a symbolic expression.
<code>csc_matrix(matrix, rownames, colnames[, ...])</code>	Generates the sparse symbolic identifiers, symbolic identifiers, sparse matrix, column pointers and row values for a symbolic variable
<code>generate_flux_symbol(reaction_index)</code>	Generate identifier symbol for a reaction flux.
<code>generate_measurement_symbol(observable_id)</code>	Generates the appropriate measurement symbol for the provided observable
<code>get_function_extern_declaration(fun, name)</code>	Constructs the extern function declaration for a given function
<code>get_model_override_implementation(fun, name)</code>	Constructs <code>amici::Model::*</code> override implementation for a given function
<code>get_sunindex_extern_declaration(fun, name, ...)</code>	Constructs the function declaration for an index function of a given function
<code>get_sunindex_override_implementation(fun, ...)</code>	Constructs the <code>amici::Model::</code> function implementation for an index function of a given function
<code>get_switch_statement(condition, cases[, ...])</code>	Generate code for switch statement
<code>is_valid_identifier(x)</code>	Check whether <code>x</code> is a valid identifier for conditions, parameters, observables.
<code>remove_typedefs(signature)</code>	Strips typedef info from a function signature
<code>smart_is_zero_matrix(x)</code>	A faster implementation of <code>sympy's is_zero_matrix</code>

continues on next page

Table 59 – continued from previous page

<code>smart_jacobian(eq, sym_var)</code>	Wrapper around symbolic jacobian with some additional checks that reduce computation time for large matrices
<code>smart_multiply(x, y)</code>	Wrapper around symbolic multiplication with some additional checks that reduce computation time for large matrices
<code>strip_pysb(symbol)</code>	Strips pysb info from a <code>pysb.Component</code> object
<code>symbol_with_assumptions(name)</code>	Central function to create symbols with consistent, canonical assumptions
<code>var_in_function_signature(name, varname)</code>	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** (`str`) – relative or absolute path to template file
- **target\_file** (`str`) – 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.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.ode_export.csc_matrix(matrix, rownames, colnames, identifier=0, pattern_only=False)`

Generates the sparse symbolic identifiers, symbolic identifiers, sparse matrix, column pointers and row values for a symbolic variable

### Parameters

- **matrix** (`sympy.matrices.dense.MutableDenseMatrix`) – dense matrix to be sparsified
- **rownames** (`typing.List[sympy.core.symbol.Symbol]`) – ids of the variable of which the derivative is computed (assuming matrix is the jacobian)
- **colnames** (`typing.List[sympy.core.symbol.Symbol]`) – ids of the variable with respect to which the derivative is computed (assuming matrix is the jacobian)
- **identifier** (`typing.Optional[int]`) – additional identifier that gets appended to symbol names to ensure their uniqueness in outer loops

- **pattern\_only** (`typing.Optional[bool]`) – flag for computing sparsity pattern without whole matrix

**Return type** `typing.Tuple[typing.List[int], typing.List[int], sympy.matrices.dense.MutableDenseMatrix, typing.List[str], sympy.matrices.dense.MutableDenseMatrix]`

**Returns** `symbol_col_ptrs, symbol_row_vals, sparse_list, symbol_list, sparse_matrix`

`amici.ode_export.generate_flux_symbol(reaction_index)`

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

**Return type** `sympy.core.symbol.Symbol`

**Returns** identifier symbol

`amici.ode_export.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.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.get_switch_statement` (*condition*, *cases*, *indentation\_level=0*, *indentation\_step=' '*)

Generate code for switch statement

**Parameters**

- **condition** (*str*) – Condition for switch
- **cases** (*typing.Dict[int, typing.List[str]]*) – Cases as dict with expressions as keys and statement as list of strings
- **indentation\_level** (*typing.Optional[int]*) – indentation level
- **indentation\_step** (*typing.Optional[str]*) – indentation whitespace per level

**Returns** Code for switch expression as list of strings

`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_typedefs` (*signature*)

Strips typedef info 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.dense.MutableDenseMatrix`

**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.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.ode_export.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.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

## 10.4.10 amici.plotting

### Plotting

Plotting related functions

### Functions Summary

---

<code>plotObservableTrajectories(rdata[, ...])</code>	Plot observable trajectories
<code>plotStateTrajectories(rdata[, ...])</code>	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`



## 10.4.11 amici.pandas

### Pandas Wrappers

This module contains convenience wrappers that allow for easy interconversion between C++ objects from *amici* and pandas DataFrames

### Functions Summary

<code>constructEdataFromDataFrame(df, model, condition)</code>	Constructs an ExpData instance according to the provided Model and DataFrame.
<code>getDataObservablesAsDataFrame(model, edata_list)</code>	Write Observables from experimental data as DataFrame.
<code>getEdataFromDataFrame(model, df[, by_id])</code>	Constructs a ExpData instances according to the provided Model and DataFrame.
<code>getResidualsAsDataFrame(model, edata_list, ...)</code>	Convert a list of ExpData to pandas DataFrame.
<code>getSimulationObservablesAsDataFrame(model, ...)</code>	Write Observables from simulation results as DataFrame.
<code>getSimulationStatesAsDataFrame(model, ..., ...)</code>	Compute model residuals according to lists of Return-Data and ExpData.

### 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 *ExpData* to *pandas 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]`) – 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 observables.

`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 state variables as columns.

`amici.pandas.getSimulationStatesAsDataFrame(model, edata_list, rdata_list, by_id=False)`  
 Compute model residuals 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/timpoints as rows and observables as columns.

## 10.4.12 amici.logging

### Logging

This module provides custom logging functionality for other amici modules

#### Functions Summary

<code>get_logger([logger_name, log_level])</code>	Returns (if existant) or creates an AMICI logger
<code>log_execution_time(description, logger)</code>	Parameterized function decorator that enables automatic execution time tracking
<code>set_log_level(logger, log_level)</code>	<b>rtype</b> <code>None</code>

## 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`

## 10.4.13 amici.gradient\_check

### Finite Difference Check

This module provides functions to automatically check correctness of amici computed sensitivities using finite difference approximations

## Functions Summary

<code>assert_fun(x)</code>	
<code>check_close(result, expected, assert_fun, ...)</code>	Compares computed values against expected values and provides rich output information.
<code>check_derivatives(model, solver[, edata, ...])</code>	Finite differences check for likelihood gradient.
<code>check_finite_difference(x0, model, solver, ...)</code>	Checks the computed sensitivity based derivatives against a finite difference approximation.
<code>check_results(rdata, field, expected, ...)</code>	Checks whether <code>rdata[field]</code> agrees with expected according to provided tolerances.

## Functions

`amici.gradient_check.assert_fun(x)`

`amici.gradient_check.check_close(result, expected, assert_fun, atol, rtol, field, ip=None)`

Compares computed values against expected values and provides rich output information.

### Parameters

- **result** (`numpy.array`) – computed values
- **expected** (`numpy.array`) – expected values
- **field** (`str`) – rdata field for which the gradient is checked, only for error reporting
- **assert\_fun** (`typing.Callable`) – function that asserts the return values of comparison, enables passing of custom assert function from testing frameworks
- **atol** (`float`) – absolute tolerance for comparison
- **rtol** (`float`) – relative tolerance for comparison
- **ip** (`typing.Optional[int]`) – parameter index

### Return type `None`

`amici.gradient_check.check_derivatives(model, solver, edata=None, assert_fun=<function assert_fun>, 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.ExpData]`) – exp data
- **assert\_fun** (`typing.Optional[typing.Callable]`) – function that asserts the return values of comparison, enables passing of custom assert function from testing frameworks
- **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, assert_fun, 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.ExpData`) – exp data
- **ip** (`int`) – parameter index
- **fields** (`typing.List[str]`) – rdata fields for which to check the gradient
- **assert\_fun** (`typing.Callable`) – function that asserts the return values of comparison, enables passing of custom assert function from testing frameworks
- **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`

`amici.gradient_check.check_results(rdata, field, expected, assert_fun, atol, rtol)`

Checks whether `rdata[field]` agrees with `expected` according to provided tolerances.

**Parameters**

- **rdata** (`amici.amici.ReturnData`) – simulation results as returned by `amici.amici.runAmiciSimulation()`
- **field** (`str`) – name of the field to check
- **expected** (`numpy.array`) – expected values
- **assert\_fun** (`typing.Callable`) – function that asserts the return values of comparison, enables passing of custom assert function from testing frameworks
- **atol** (`float`) – absolute tolerance for comparison
- **rtol** (`float`) – relative tolerance for comparison

**Return type** `None`

## 10.4.14 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 PEstab, for which the whole workflow of generating the mapping is automatized.

### Classes

<code>ParameterMapping([parameter_mappings])</code>	Parameter mapping for multiple conditions.
<code>ParameterMappingForCondition([map_sim_var, Parameter mapping for condition. ...])</code>	

### 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 *ParameterMappingForConditions*.

**Parameters** **parameter\_mappings** (`typing.Optional[typing.List[amici.parameter_mapping.ParameterMappingForCondition]]`) – List of parameter mappings for specific conditions.

**\_\_init\_\_** (*parameter\_mappings=None*)

Initialize self. See help(type(self)) for accurate signature.

### Methods Summary

<code>__init__([parameter_mappings])</code>	Initialize self.
<code>append(parameter_mapping_for_condition)</code>	Append a condition specific parameter mapping.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.

### Methods

**\_\_init\_\_** (*parameter\_mappings=None*)

Initialize self. See help(type(self)) for accurate signature.

**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)  
Initialize self. See help(type(self)) for accurate signature.
```

#### **Methods Summary**

---

```
__init__ ([map_sim_var,    scale_map_sim_var,    Initialize self.
```

---

```
...])
```

---



## 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*)  
Initialize self. See help(type(self)) for accurate signature.

## Functions Summary

<code>amici_to_petab_scale(amici_scale)</code>	Convert amici scale id to petab scale id.
<code>fill_in_parameters(edatas, ...)</code>	Fill fixed and dynamic parameters into the edatas (in-place).
<code>fill_in_parameters_for_condition(edata, ...)</code>	Fill fixed and dynamic parameters into the edata for condition (in-place).
<code>petab_to_amici_scale(petab_scale)</code>	Convert petab scale id to amici scale id.
<code>scale_parameter(value, petab_scale)</code>	Bring parameter from linear scale to target scale.
<code>scale_parameters_dict(value_dict, ...)</code>	Bring parameters from linear scale to target scale.
<code>unscale_parameter(value, petab_scale)</code>	Bring parameter from scale to linear scale.
<code>unscale_parameters_dict(value_dict, ...)</code>	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.ExpData]`) – List of experimental datas `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.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`



## 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/buildSundials.sh  
./scripts/buildSuitesparse.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. On 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*
- *Classes*
- *Enums*
- *Functions*
- *Typedefs*
- *Variables*



## Namespaces

- *Namespace amici::generic\_model*
- *Namespace amici::hdf5*

## Classes

- *Struct ModelDimensions*
- *Struct ModelState*
- *Struct ModelStateDerived*
- *Struct SimulationState*
- *Class AbstractModel*
- *Class AmiciApplication*
- *Class AmiException*
- *Class AmiVector*
- *Class AmiVectorArray*
- *Class BackwardProblem*
- *Class ConditionContext*
- *Class ContextManager*
- *Class CcodeException*
- *Class CCodeSolver*
- *Class ExpData*
- *Class FinalStateStorer*
- *Class ForwardProblem*
- *Class IDAException*
- *Class IDASolver*
- *Class IntegrationFailure*
- *Class IntegrationFailureB*
- *Class Model*
- *Class Model\_DAE*
- *Class Model\_ODE*
- *Class ModelContext*
- *Class NewtonFailure*
- *Class NewtonSolver*
- *Class NewtonSolverDense*
- *Class NewtonSolverIterative*
- *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 NewtonDampingFactorMode*
- *Enum NonlinearSolverIteration*
- *Enum ParameterScaling*
- *Enum RDataReporting*
- *Enum SecondOrderMode*
- *Enum SensitivityMethod*
- *Enum SensitivityOrder*
- *Enum SteadyStateContext*
- *Enum SteadyStateSensitivityMode*

- Enum *SteadyStateStatus*

## Functions

- Function *amici::amici\_daxpy*
- Function *amici::amici\_dgemm*
- Function *amici::amici\_dgemv*
- Function *amici::backtraceString*
- Template Function *amici::checkBufferSize*
- Function *amici::checkFieldNames*
- Function *amici::checkSigmaPositivity(std::vector<realtype> const &sigmaVector, const char \*vectorName)*
- Function *amici::checkSigmaPositivity(realtype sigma, const char \*sigmaName)*
- Function *amici::DDspline*
- Function *amici::DDspline\_pos*
- Template Function *amici::deserializeFromChar*
- Template Function *amici::deserializeFromString*
- Function *amici::dirac*
- Function *amici::Dmax*
- Function *amici::Dmin*
- Function *amici::Dspline*
- Function *amici::Dspline\_pos*
- Function *amici::expDataFromMatlabCall*
- Function *amici::getNaN*
- Function *amici::getReturnDataMatlabFromAmiciCall*
- Function *amici::getScaledParameter*
- Function *amici::getUnscaledParameter*
- Function *amici::heaviside*
- Function *amici::initAndAttachArray*
- Function *amici::initMatlabDiagnosisFields*
- Function *amici::initMatlabReturnFields*
- Function *amici::isInf*
- Function *amici::isNaN*
- Function *amici::log*
- Function *amici::max*
- Function *amici::min*
- Function *amici::N\_VGetArrayPointerConst*
- Function *amici::operator==(const Model &a, const Model &b)*

- Function `amici::operator==(const ModelDimensions &a, const ModelDimensions &b)`
- Function `amici::operator==(const SimulationParameters &a, const SimulationParameters &b)`
- Function `amici::operator==(const Solver &a, const Solver &b)`
- Function `amici::pos_pow`
- Function `amici::printErrMsgIdAndTxt`
- Function `amici::printfToString`
- Function `amici::printWarnMsgIdAndTxt`
- Function `amici::regexErrorToString`
- Template Function `amici::reorder`
- Function `amici::runAmiciSimulation`
- Function `amici::runAmiciSimulations`
- Function `amici::scaleParameters`
- Template Function `amici::serializeToChar`
- Template Function `amici::serializeToStdVec`
- Template Function `amici::serializeToString`
- Function `amici::setModelData`
- Function `amici::setSolverOptions`
- Function `amici::setupReturnData`
- Function `amici::seval`
- Function `amici::sign`
- Function `amici::sinteg`
- Template Function `amici::slice(std::vector<T> &data, int index, unsigned size)`
- Template Function `amici::slice(const std::vector<T> &data, int index, unsigned size)`
- Function `amici::spline`
- Function `amici::spline_pos`
- Function `amici::unscaleParameters`
- Function `amici::wrapErrHandlerFn`
- Template Function `amici::writeMatlabField0`
- Template Function `amici::writeMatlabField1`
- Template Function `amici::writeMatlabField2`
- Template Function `amici::writeMatlabField3`
- Template Function `amici::writeMatlabField4`
- Template Function `amici::writeSlice(const gsl::span<const T> slice, gsl::span<T> buffer)`
- Template Function `amici::writeSlice(const std::vector<T> &s, std::vector<T> &b)`
- Template Function `amici::writeSlice(const std::vector<T> &s, gsl::span<T> b)`
- Function `amici::writeSlice(const AmiVector &s, gsl::span<realtype> b)`

## Typedefs

- *Typedef amici::const\_N\_Vector*
- *Typedef amici::outputFunctionType*
- *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\_ILL\_INPUT*
- *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\_ROOT\_RETURN*
- *Variable amici::AMICI\_SINGULAR\_JACOBIAN*
- *Variable amici::AMICI\_SUCCESS*
- *Variable amici::AMICI\_TOO\_MUCH\_ACC*
- *Variable amici::AMICI\_TOO\_MUCH\_WORK*
- *Variable amici::AMICI\_UNRECOVERABLE\_ERROR*
- *Variable amici::defaultContext*
- *Variable amici::pi*

## Namespace amici::generic\_model

### Contents

- *Functions*

## Functions

- *Function amici::generic\_model::getModel*

## Namespace amici::hdf5

### Contents

- *Functions*

## Functions

- *Function amici::hdf5::attributeExists(H5::H5File const &file, const std::string &optionsObject, const std::string &attributeName)*
- *Function amici::hdf5::attributeExists(H5::H5Object const &object, const std::string &attributeName)*
- *Function amici::hdf5::createAndWriteDouble1DDataset*
- *Function amici::hdf5::createAndWriteDouble2DDataset*
- *Function amici::hdf5::createAndWriteDouble3DDataset*
- *Function amici::hdf5::createAndWriteInt1DDataset*
- *Function amici::hdf5::createAndWriteInt2DDataset*
- *Function amici::hdf5::createGroup*
- *Function amici::hdf5::createOrOpenForWriting*
- *Function amici::hdf5::getDoubleDataset1D*
- *Function amici::hdf5::getDoubleDataset2D*
- *Function amici::hdf5::getDoubleDataset3D*
- *Function amici::hdf5::getDoubleScalarAttribute*
- *Function amici::hdf5::getIntDataset1D*
- *Function amici::hdf5::getIntScalarAttribute*
- *Function amici::hdf5::locationExists(std::string const &filename, std::string const &location)*
- *Function amici::hdf5::locationExists(H5::H5File const &file, std::string const &location)*
- *Function amici::hdf5::readModelDataFromHDF5(std::string const &hdf5file, Model &model, std::string const &datasetPath)*
- *Function amici::hdf5::readModelDataFromHDF5(H5::H5File const &file, Model &model, std::string const &datasetPath)*
- *Function amici::hdf5::readSimulationExpData*
- *Function amici::hdf5::readSolverSettingsFromHDF5(const H5::H5File &file, Solver &solver, std::string const &datasetPath)*
- *Function amici::hdf5::readSolverSettingsFromHDF5(std::string const &hdf5file, Solver &solver, std::string const &datasetPath)*

- *Function amici::hdf5::writeReturnData(const ReturnData &rdata, std::string const &hdf5Filename, const std::string &hdf5Location)*
- *Function amici::hdf5::writeReturnData(const ReturnData &rdata, H5::H5File const &file, const std::string &hdf5Location)*
- *Function amici::hdf5::writeReturnDataDiagnosis*
- *Function amici::hdf5::writeSimulationExpData*
- *Function amici::hdf5::writeSolverSettingsToHDF5(Solver const &solver, H5::H5File const &file, std::string const &hdf5Location)*
- *Function amici::hdf5::writeSolverSettingsToHDF5(Solver const &solver, std::string const &hdf5Filename, std::string const &hdf5Location)*

## Namespace boost

### Contents

- *Namespaces*

## Namespaces

- *Namespace boost::serialization*

## Namespace boost::serialization

### Contents

- *Functions*

## Functions

- *Template Function boost::serialization::archiveVector*
- *Template Function boost::serialization::serialize(Archive &ar, amici::Solver &s, unsigned int version)*
- *Template Function boost::serialization::serialize(Archive &ar, amici::CVodeSolver &s, unsigned int version)*
- *Template Function boost::serialization::serialize(Archive &ar, amici::IDASolver &s, unsigned int version)*
- *Template Function boost::serialization::serialize(Archive &ar, amici::ReturnData &r, unsigned int version)*
- *Template Function boost::serialization::serialize(Archive &ar, amici::Model &m, unsigned int version)*

## Namespace gsl

### Contents

- *Functions*

## Functions

- *Function* `gsl::make_span(N_Vector nv)`
- *Function* `gsl::make_span(SUNMatrix m)`

## Namespace std

STL namespace.

## Classes and Structs

### 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** `amici::ModelDimensions`

Container for model dimensions.

Holds number of states, observables, etc.

Subclassed by *amici::Model*, *amici::ReturnData*



## Public Functions

**ModelDimensions** () = default

Default ctor

**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 *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  $\dot{x}$
- *ndJydy*: Number of nonzero elements in the  $y$  derivative of  $dJy$  (shape *nytrue*)
- *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  $x_{dot}$

std::vector<int> **ndJydy**  
Number of nonzero elements in the  $y$  derivative of  $dJy$  (dimension `nytrue`)

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** amici::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** amici::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

**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*)

*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> 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\_forwardproblem.h

## Struct Documentation

**struct** amici::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 amici::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

**~AbstractModel** () = default

`std::unique_ptr<Solver> getSolver () = 0`

Retrieves the solver object.

**Return** The *Solver* instance

`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

`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

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

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

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

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



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

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

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

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

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)

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)

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)

std::string **getAmiciVersion** () **const**  
Returns the AMICI version that was used to generate the model.

**Return** AMICI version string

`std::string getAmiciCommit() const`

Returns the AMICI commit that was used to generate the model.

**Return** AMICI commit string

`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

`bool isFixedParameterStateReinitializationAllowed() const`

Function indicating whether reinitialization of states depending on fixed parameters is permissible.

**Return** flag indicating whether reinitialization of states depending on fixed parameters is permissible

`void fx0_fixedParameters(realtype *x0, const realtype t, const realtype *p, const realtype *k, gsl::span<const int> reinitialization_state_idx)`

Model-specific implementation of fx0\_fixedParameters.

#### Parameters

- `x0`: initial state
- `t`: initial time
- `p`: parameter vector
- `k`: constant vector
- `reinitialization_state_idx`: Indices of states to be reinitialized based on provided constants / fixed parameters.

`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_idx)`

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_idx`s: Indices of states to be reinitialized based on provided constants / fixed parameters.

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

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)

void **fstau** (*realtype* \*stau, const *realtype* t, const *realtype* \*x, const *realtype* \*p, const *realtype* \*k, const *realtype* \*h, 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
- sx: current state sensitivity
- ip: sensitivity index
- ie: event index

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

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.

#### 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

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

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

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

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

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

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

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

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

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

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

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)

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

```
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

```
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

void **fsgimay** (*realtype* \*sigmay, const *realtype* t, const *realtype* \*p, const *realtype* \*k)  
Model-specific implementation of fsgimay.

#### Parameters

- `sigmay`: standard deviation of measurements
- `t`: current time
- `p`: parameter vector
- `k`: constant vector

void **fdsigmaydp** (*realtype* \*dsigmaydp, const *realtype* t, const *realtype* \*p, const *realtype* \*k, int  
ip)  
Model-specific implementation of fsgimay.

#### Parameters

- `dsigmaydp`: partial derivative of standard deviation of measurements
- `t`: current time
- `p`: parameter vector
- `k`: constant vector
- `ip`: sensitivity index

void **fsgimaz** (*realtype* \*sigmaz, const *realtype* t, const *realtype* \*p, const *realtype* \*k)  
Model-specific implementation of fsgimaz.

#### Parameters

- `sigmaz`: standard deviation of event measurements
- `t`: current time
- `p`: parameter vector
- `k`: constant vector

void **fdsigmazdp** (*realtype* \*dsigmazdp, const *realtype* t, const *realtype* \*p, const *realtype* \*k, int  
ip)  
Model-specific implementation of fsgimaz.

#### Parameters

- `dsigmazdp`: partial derivative of standard deviation of event measurements
- `t`: current time
- `p`: parameter vector
- `k`: constant vector
- `ip`: sensitivity index

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

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

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

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

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

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

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

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

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

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

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

```
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

```
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

```
void fdwdp_colptrs(SUNMatrixWrapper &dwdp)  
Model-specific implementation for dwdp, column pointers.
```

**Parameters**

- dwdp: sparse matrix to which colptrs will be written

```
void fdwdp_rowvals(SUNMatrixWrapper &dwdp)  
Model-specific implementation for dwdp, row values.
```

**Parameters**

- dwdp: sparse matrix to which rowvals will be written

```
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

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

void **fdwdx\_colptrs** (*SUNMatrixWrapper* &`dwdx`)  
 Model-specific implementation for `dwdx`, column pointers.

**Parameters**

- `dwdx`: sparse matrix to which `colptrs` will be written

void **fdwdx\_rowvals** (*SUNMatrixWrapper* &`dwdx`)  
 Model-specific implementation for `dwdx`, row values.

**Parameters**

- `dwdx`: sparse matrix to which `rowvals` will be written

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
- `tc1`: Total abundances for conservation laws

void **fdwdw\_colptrs** (*SUNMatrixWrapper* &*dwdw*)  
 Model-specific implementation of fdwdw, colptrs part.

#### Parameters

- *dwdw*: sparse matrix to which colptrs will be written

void **fdwdw\_rowvals** (*SUNMatrixWrapper* &*dwdw*)  
 Model-specific implementation of fdwdw, rowvals part.

#### Parameters

- *dwdw*: sparse matrix to which rowvals will be written

## Class AmiciApplication

- Defined in `file_include_amici_amici.h`

## Class Documentation

**class** `amici::AmiciApplication`

Main class for making calls to AMICI.

This class is used to provide separate AMICI contexts, for example, for use in multi-threaded applications where different threads want to use AMICI with different settings, such custom logging functions.

NOTE: For this moment, the context object needs to be set manually to any *Model* and *Solver* object. If not set, they will use the default output channel.

## Public Functions

**AmiciApplication** () = default

`std::unique_ptr<ReturnData>` **runAmiciSimulation** (*Solver* &*solver*, **const** *ExpData* \**edata*,  
*Model* &*model*, **bool** *rethrow* = false)

Core integration routine. Initializes the solver and runs the forward and backward problem.

**Return** *rdata* pointer to return data object

#### Parameters

- *solver*: *Solver* instance
- *edata*: pointer to experimental data object



- `model`: model specification object
- `rethrow`: rethrow integration exceptions?

```
std::vector<std::unique_ptr<ReturnData>> runAmiciSimulations (Solver const &solver,
                                                            const std::vector<ExpData*>
                                                            &edatas, Model const
                                                            &model, bool failfast, int
                                                            num_threads)
```

Same as `runAmiciSimulation`, but for multiple *ExpData* instances.

**Return** vector of pointers to return data objects

#### 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

```
void warningF (const char *identifier, const char *format, ...) const
printf interface to warning()
```

#### Parameters

- `identifier`: warning identifier
- `format`: string with warning message printf-style format
- `...`: arguments to be formatted

```
void errorF (const char *identifier, const char *format, ...) const
printf interface to error()
```

#### Parameters

- `identifier`: warning identifier
- `format`: string with error message printf-style format
- `...`: arguments to be formatted

```
int checkFinite (gsl::span<const realtype> array, const char *fun)
Checks the values in an array for NaNs and Infs.
```

**Return** `AMICI_RECOVERABLE_ERROR` if a NaN/Inf value was found, `AMICI_SUCCESS` otherwise

#### Parameters

- `array`: array
- `fun`: name of calling function

## Public Members

*outputFunctionType* **warning** = *printWarnMsgIdAndTxt*  
Function to process warnings

*outputFunctionType* **error** = *printErrMsgIdAndTxt*  
Function to process errors

## 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** `amici::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** (`char const *fmt, ...`)  
Constructor with printf style interface.

#### Parameters

- `fmt`: error message with printf format
- `...`: printf formatting variables

**const** char \***what** () **const noexcept override**  
 Override of default error message function.

**Return** msg error message

**const** char \***getBacktrace** () **const**  
 Returns the stored backtrace.

**Return** trace backtrace

void **storeBacktrace** (int *nMaxFrames*)  
 Stores the current backtrace.

#### Parameters

- *nMaxFrames*: number of frames to go back in stacktrace

## Class AmiVector

- Defined in file\_include\_amici\_vector.h

## Class Documentation

**class** amici::AmiVector

*AmiVector* class provides a generic interface to the NVector\_Serial struct

### Public Functions

**AmiVector** () = default  
 Default constructor.

**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

**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

**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

**AmiVector** (`const` *AmiVector* &`vold`)  
copy constructor

**Parameters**

- `vold`: vector from which the data will be copied

**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

**Return** left hand side

**Parameters**

- `other`: right hand side

*realtype* \***data** ()  
data accessor

**Return** pointer to data array

`const` *realtype* \***data** () `const`  
const data accessor

**Return** const pointer to data array

`N_Vector` **getNVector** ()  
`N_Vector` accessor.

**Return** `N_Vector`

`const` *N\_Vector* **getNVector** () `const`  
`N_Vector` accessor.

**Return** `N_Vector`

`std::vector`<*realtype*> **const** &**getVector** () `const`  
Vector accessor.

**Return** Vector

`int` **getLength** () `const`  
returns the length of the vector

**Return** 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

**Return** element

**Parameters**

- pos: index of element

*realtype* &**at** (int pos)  
accessor to data elements of the vector

**Return** element

**Parameters**

- pos: index of element

**const** *realtype* &**at** (int pos) **const**  
accessor to data elements of the vector

**Return** element

**Parameters**

- pos: index of element

void **copy** (**const** *AmiVector* &*other*)  
copies data from another *AmiVector*

**Parameters**

- other: data source

## Class *AmiVectorArray*

- Defined in file\_include\_amici\_vector.h

## Class Documentation

**class** amici::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<realtype> 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

**Return** left hand side

#### Parameters

- *other*: right hand side

*realtype* \***data** (int *pos*)

accessor to data of *AmiVector* elements

**Return** pointer to data array

#### Parameters

- *pos*: index of *AmiVector*

const *realtype* \***data** (int *pos*) const

const accessor to data of *AmiVector* elements

**Return** const pointer to data array

#### Parameters

- *pos*: index of *AmiVector*

*realtype* &**at** (int *ipos*, int *jpos*)  
 accessor to elements of *AmiVector* elements

**Return** element

**Parameters**

- *ipos*: inner index in *AmiVector*
- *jpos*: outer index in *AmiVectorArray*

**const** *realtype* &**at** (int *ipos*, int *jpos*) **const**  
 const accessor to elements of *AmiVector* elements

**Return** element

**Parameters**

- *ipos*: inner index in *AmiVector*
- *jpos*: outer index in *AmiVectorArray*

*N\_Vector* \***getNVectorArray** ()  
 accessor to *NVectorArray*

**Return** *N\_VectorArray*

*N\_Vector* **getNVector** (int *pos*)  
 accessor to *NVector* element

**Return** *N\_Vector*

**Parameters**

- *pos*: index of corresponding *AmiVector*

**const** *N\_Vector* **getNVector** (int *pos*) **const**  
 const accessor to *NVector* element

**Return** *N\_Vector*

**Parameters**

- *pos*: index of corresponding *AmiVector*

*AmiVector* &**operator** [] (int *pos*)  
 accessor to *AmiVector* elements

**Return** *AmiVector*

**Parameters**

- *pos*: index of *AmiVector*

**const** *AmiVector* &**operator** [] (int *pos*) **const**  
 const accessor to *AmiVector* elements

**Return** **const** *AmiVector*

**Parameters**

- pos: index of *AmiVector*

int **getLength** () **const**  
length of *AmiVectorArray*

**Return** 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** amici::BackwardProblem  
class to solve backwards problems.  
solves the backwards problem for adjoint sensitivity analysis and handles events and data-points

## Public Functions

**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.

*realtype* **gett** () **const**  
Accessor for current time t.



**Return** t

int **getwhich** () **const**  
 Accessor for which.

**Return** which

int \***getwhichptr** ()  
 Accessor for pointer to which.

**Return** which

std::vector<*realtype*> **const &getdJydx** () **const**  
 Accessor for dJydx.

**Return** dJydx

*AmiVector* **const &getAdjointState** () **const**  
 Accessor for xB.

**Return** xB

*AmiVector* **const &getAdjointQuadrature** () **const**  
 Accessor for xQB.

**Return** xQB

## Class ConditionContext

- Defined in file\_include\_amici\_edata.h

## Inheritance Relationships

### Base Type

- public amici::ContextManager (*Class ContextManager*)

## Class Documentation

**class** amici::ConditionContext : public amici::ContextManager

The *ConditionContext* class applies condition-specific *amici::Model* settings and restores them when going out of scope.

## Public Functions

**ConditionContext** (*Model* \*model, **const** *ExpData* \*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= (**const** *ConditionContext* &other) = delete

**~ConditionContext** ()

void **applyCondition** (**const** *ExpData* \*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** amici::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 CCodeException

- Defined in file\_include\_amici\_exception.h

## Inheritance Relationships

### Base Type

- public amici::AmiException (*Class AmiException*)

## Class Documentation

### **class** amici::CCodeException : public amici::AmiException

ccode exception handler class

#### Public Functions

**CCodeException** (int *error\_code*, const char \**function*)

Constructor.

#### Parameters

- *error\_code*: error code returned by ccode function
- *function*: ccode function name

## Class CCodeSolver

- Defined in file\_include\_amici\_solver\_ccodes.h

## Inheritance Relationships

### Base Type

- `public amici::Solver (Class Solver)`

### Class Documentation

**class** `amici::CNodeSolver` : **public** `amici::Solver`

The *CNodeSolver* class is a wrapper around the SUNDIALS CVODES solver.

#### Public Functions

`~CNodeSolver () override` = default

*Solver* \*`clone () const override`

Clone this instance.

**Return** The clone

`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)

`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)

`void sensToggleOff () const override`

Switches off computation of state sensitivities without deallocating the memory for sensitivities.

`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

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

int **solve** (*realtype* *tout*, int *itask*) **const override**  
 Solves the forward problem until a predefined timepoint.

**Return** status flag indicating success of execution

**Parameters**

- *tout*: timepoint until which simulation should be performed
- *itask*: task identifier, can be CV\_NORMAL or CV\_ONE\_STEP

int **solveF** (*realtype* *tout*, int *itask*, int \**ncheckPtr*) **const override**  
 Solves the forward problem until a predefined timepoint (adjoint only)

**Return** status flag indicating success of execution

**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

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

void **getDky** (*realtype* *t*, int *k*) **const override**  
 interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- *t*: timepoint
- *k*: derivative order

void **getSensDky** (*realtype* *t*, int *k*) **const override**  
 interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- *t*: timepoint
- *k*: derivative order

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

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

void **getRootInfo** (int *\*rootsfound*) **const override**  
*getRootInfo* extracts information which event occurred

**Parameters**

- *rootsfound*: array with flags indicating whether the respective event occurred

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

void **turnOffRootFinding** () **const override**  
Disable rootfinding.

**const** *Model* **\*getModel** () **const override**  
Accessor function to the model stored in the user data

**Return** user data model

void **setLinearSolver** () **const override**  
Sets the linear solver for the forward problem.

void **setLinearSolverB** (int *which*) **const override**  
Sets the linear solver for the backward problem.

**Parameters**

- *which*: index of the backward problem

void **setNonLinearSolver** () **const override**  
Set the non-linear solver for the forward problem.

void **setNonLinearSolverSens** () **const override**  
Set the non-linear solver for sensitivities.

void **setNonLinearSolverB**(int *which*) **const override**  
 Set the non-linear solver for the backward problem.

#### Parameters

- *which*: index of the backward problem

### Protected Functions

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)

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)

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

void **getSens**() **const override**  
 extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

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

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

void **getQuadDky**(*realtype t*, int *k*) **const override**  
 interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- `t`: timepoint
- `k`: derivative order

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)

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.

void **allocateSolver** () **const override**  
Create specifies solver method and initializes solver memory for the forward problem.

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

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

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



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

void **setQuadErrCon** (bool *flag*) **const override**  
 Specifies whether error control is also enforced for the forward quadrature problem.

**Parameters**

- *flag*: activation flag

void **setErrHandlerFn** () **const override**  
 Attaches the error handler function (*errMsgIdAndTxt*) to the solver.

void **setUserData** (*Model* \**model*) **const override**  
 Attaches the user data instance (here this is a *Model*) to the forward problem.

**Parameters**

- *model*: *Model* instance

void **setUserDataB** (int *which*, *Model* \**model*) **const override**  
 attaches the user data instance (here this is a *Model*) to the backward problem

**Parameters**

- *which*: identifier of the backwards problem
- *model*: *Model* instance

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

void **setStabLimDet** (int *stldet*) **const override**  
 activates stability limit detection for the forward problem

**Parameters**

- *stldet*: flag for stability limit detection (TRUE or FALSE)

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)

void **setId** (const *Model* \**model*) **const override**  
specify algebraic/differential components (DAE only)

**Parameters**

- *model*: model specification

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

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

void **adjInit** () **const override**  
initializes the adjoint problem

void **quadInit** (const *AmiVector* &*xQ0*) **const override**  
initializes the quadratures

**Parameters**

- *xQ0*: vector with initial values for *xQ*

void **allocateSolverB** (int \**which*) **const override**  
Specifies solver method and initializes solver memory for the backward problem.

**Parameters**

- *which*: identifier of the backwards problem

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

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

void **quadSStolerances** (*realtype* *reltolQ*, *realtype* *abstolQ*) **const override**  
sets relative and absolute tolerances for the quadrature problem

#### Parameters

- *reltolQB*: relative tolerances
- *abstolQB*: absolute tolerances

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

void **diag** () **const override**  
attaches a diagonal linear solver to the forward problem

void **diagB** (int *which*) **const override**  
attaches a diagonal linear solver to the backward problem

#### Parameters

- *which*: identifier of the backwards problem

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

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

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

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

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

void \***getAdjBmem** (void \**ami\_mem*, int *which*) **const override**  
Retrieves the solver memory instance for the backward problem.

**Return** A (void \*) pointer to the CVODES memory allocated for the backward problem.

**Parameters**

- *which*: identifier of the backwards problem
- *ami\_mem*: pointer to the forward solver memory instance

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

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

void **sensInit1** (const *AmiVectorArray* &sx0, const *AmiVectorArray* &sdx0) const override  
 Initializes the forward sensitivities.

#### Parameters

- sx0: initial states sensitivities
- sdx0: initial derivative states sensitivities

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

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

void **rootInit** (int *ne*) const override  
 Initializes the rootfinding for events.

#### Parameters

- ne: number of different events

void **setDenseJacFn** () const override  
 Set the dense Jacobian function.

void **setSparseJacFn** () const override  
 sets the sparse Jacobian function

void **setBandJacFn** () const override  
 sets the banded Jacobian function

void **setJacTimesVecFn** () const override  
 sets the Jacobian vector multiplication function

void **setDenseJacFnB** (int *which*) const override  
 sets the dense Jacobian function

#### Parameters

- which: identifier of the backwards problem

void **setSparseJacFnB** (int *which*) const override  
 sets the sparse Jacobian function

**Parameters**

- `which`: identifier of the backwards problem

void **setBandJacFnB** (int *which*) **const override**  
sets the banded Jacobian function

**Parameters**

- `which`: identifier of the backwards problem

void **setJacTimesVecFnB** (int *which*) **const override**  
sets the Jacobian vector multiplication function

**Parameters**

- `which`: identifier of the backwards problem

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.

**Return** Whether *a* and *b* are equal

**Parameters**

- *a*:
- *b*:

**Class ExpData**

- Defined in `file_include_amici_edata.h`

## Inheritance Relationships

### Base Type

- `public amici::SimulationParameters` (*Class SimulationParameters*)

### Class Documentation

**class** `amici::ExpData` : **public** `amici::SimulationParameters`  
*ExpData* carries all information about experimental or condition-specific data.

#### Public Functions

**ExpData** () = default  
 default constructor

**ExpData** (const *ExpData*&) = 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: *nmax-events* x *nztrue*, row-major)

**ExpData** (**const** *Model* &*model*)  
constructor that initializes with *Model*

#### Parameters

- *model*: pointer to model specification object

**ExpData** (**const** *ReturnData* &*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** (**const** *ReturnData* &*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

**Return** number of observables of the non-augmented model



`int nztrue () const`

number of event observables of the non-augmented model

**Return** number of event observables of the non-augmented model

`int nmaxevent () const`

maximal number of events to track

**Return** maximal number of events to track

`int nt () const`

number of timepoints

**Return** number of timepoints

`void setTimepoints (const std::vector<realtype> &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

**Return** ExpData::ts

`realtype getTimepoint (int it) const`

get function that returns timepoint at index

**Return** timepoint timepoint at index

**Parameters**

- `it`: timepoint index

`void setObservedData (const std::vector<realtype> &observedData)`

set function that copies data from input to ExpData::my

**Parameters**

- `observedData`: observed data (dimension: `nt` x `nytrue`, row-major)

`void setObservedData (const std::vector<realtype> &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

**Return** boolean specifying if data was set

**Parameters**

- *it*: time index
- *iy*: observable index

`std::vector<realtype> const &getObservedData () const`  
get function that copies data from `ExpData::observedData` to output

**Return** observed data (dimension: `nt` x `nytrue`, row-major)

`const realtype*getObservedDataPtr (int it) const`  
get function that returns a pointer to observed data at index

**Return** pointer to observed data at index (dimension: `nytrue`)

**Parameters**

- *it*: timepoint index

`void setObservedDataStdDev (const std::vector<realtype> &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 (const std::vector<realtype> &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

**Return** boolean specifying if standard deviation of data was set

**Parameters**

- `it`: time index
- `iy`: observable index

`std::vector<realtype> const &getObservedDataStdDev () const`  
 get function that copies data from `ExpData::observedDataStdDev` to output

**Return** standard deviation of observed data

`const realtype *getObservedDataStdDevPtr (int it) const`  
 get function that returns a pointer to standard deviation of observed data at index

**Return** pointer to standard deviation of observed data at index

**Parameters**

- `it`: timepoint index

`void setObservedEvents (const std::vector<realtype> &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 (const std::vector<realtype> &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

**Return** boolean specifying if data was set

**Parameters**

- `ie`: event index
- `iz`: event observable index

`std::vector<realtype> const &getObservedEvents () const`  
 get function that copies data from `ExpData::mz` to output

**Return** observed event data

`const realtype *getObservedEventsPtr (int ie) const`  
 get function that returns a pointer to observed data at `ie`th occurrence

**Return** pointer to observed event data at `ie`th occurrence

**Parameters**

- `ie`: event occurrence

void **setObservedEventsStdDev** (**const** std::vector<*realtype*> &*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** (**const** std::vector<*realtype*> &*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

**Return** boolean specifying if standard deviation of event data was set

**Parameters**

- *ie*: event index
- *iz*: event observable index

std::vector<*realtype*> **const** &**getObservedEventsStdDev** () **const**  
get function that copies data from ExpData::observedEventsStdDev to output

**Return** standard deviation of observed event data

**const** *realtype* \***getObservedEventsStdDevPtr** (int *ie*) **const**  
get function that returns a pointer to standard deviation of observed event data at ie-th occurrence

**Return** pointer to standard deviation of observed event data at ie-th occurrence

**Parameters**

- *ie*: event occurrence

## 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, const char \*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, const char \*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)

## Class FinalStateStorer

- Defined in file\_include\_amici\_forwardproblem.h

## Inheritance Relationships

### Base Type

- `public amici::ContextManager` (*Class ContextManager*)

## Class Documentation

**class** `amici::FinalStateStorer` : **public** `amici::ContextManager`  
stores the stimulation state when it goes out of scope

### Public Functions

**FinalStateStorer** (*ForwardProblem* \*fwd)  
constructor, attaches problem pointer

#### Parameters

- fwd: problem from which the simulation state is to be stored

*FinalStateStorer* &operator= (const *FinalStateStorer* &other) = delete

**~FinalStateStorer** ()  
destructor, stores simulation state

## Class ForwardProblem

- Defined in file\_include\_amici\_forwardproblem.h

## Class Documentation

**class** `amici::ForwardProblem`  
The *ForwardProblem* class groups all functions for solving the forward problem.

### Public Functions

**ForwardProblem** (const *ExpData* \*edata, *Model* \*model, *Solver* \*solver, const *SteadystateProblem* \*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*, const *ExpData* &*edata*)

computes adjoint updates `dJydx` according to provided model and expdata

#### Parameters

- `model`: *Model* instance
- `edata`: experimental data

*realtype* **getTime()** const

Accessor for `t`.

**Return** `t`

*AmiVector* const &**getState()** const

Accessor for `x`.

**Return** `x`

*AmiVector* const &**getStateDerivative()** const

Accessor for `dx`.

**Return** `dx`

*AmiVectorArray* const &**getStateSensitivity()** const

Accessor for `sx`.

**Return** `sx`

std::vector<*AmiVector*> const &**getStatesAtDiscontinuities()** const

Accessor for `x_disc`.

**Return** `x_disc`

std::vector<*AmiVector*> const &**getRHSAtDiscontinuities()** const

Accessor for `xdot_disc`.

**Return** `xdot_disc`

std::vector<*AmiVector*> const &**getRHSBeforeDiscontinuities()** const

Accessor for `xdot_old_disc`.

**Return** `xdot_old_disc`

std::vector<int> const &**getNumberOfRoots()** const

Accessor for `nroots`.

**Return** nroots

`std::vector<realtype> const &getDiscontinuities () const`  
Accessor for discs.

**Return** discs

`std::vector<std::vector<int>> const &getRootIndexes () const`  
Accessor for rootidx.

**Return** rootidx

`std::vector<realtype> const &getDJydx () const`  
Accessor for dJydx.

**Return** dJydx

`std::vector<realtype> const &getDJzdx () const`  
Accessor for dJzdx.

**Return** dJzdx

`AmiVector *getStatePointer ()`  
Accessor for pointer to x.

**Return** &x

`AmiVector *getStateDerivativePointer ()`  
Accessor for pointer to dx.

**Return** &dx

`AmiVectorArray *getStateSensitivityPointer ()`  
accessor for pointer to sx

**Return** &sx

`AmiVectorArray *getStateDerivativeSensitivityPointer ()`  
Accessor for pointer to sdx.

**Return** &sdx

`int getCurrentTimeIteration () const`  
Accessor for it.

**Return** it

`realtype getFinalTime () const`  
Returns final time point for which simulations are available.

**Return** time point

`int getEventCounter () const`  
Returns maximal event index for which simulations are available.



**Return** index

int **getRootCounter** () **const**

Returns maximal event index for which the timepoint is available.

**Return** index

**const** *SimulationState* &**getSimulationStateTimepoint** (int *it*) **const**

Retrieves the carbon copy of the simulation state variables at the specified timepoint index.

**Return** state

**Parameters**

- *it*: timepoint index

**const** *SimulationState* &**getSimulationStateEvent** (int *iroot*) **const**

Retrieves the carbon copy of the simulation state variables at the specified event index.

**Return** *SimulationState*

**Parameters**

- *iroot*: event index

**const** *SimulationState* &**getInitialSimulationState** () **const**

Retrieves the carbon copy of the simulation state variables at the initial timepoint.

**Return** *SimulationState*

**const** *SimulationState* &**getFinalSimulationState** () **const**

Retrieves the carbon copy of the simulation state variables at the final timepoint (or when simulation failed)

**Return** *SimulationState*

## Public Members

*Model* \***model**

pointer to model instance

*Solver* \***solver**

pointer to solver instance

**const** *ExpData* \***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** `amici::IDAException` : **public** `amici::AmiException`  
ida exception handler class

### Public Functions

**IDAException** (int *error\_code*, **const** char \**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** `amici::IDASolver` : **public** `amici::Solver`  
The *IDASolver* class is a wrapper around the SUNDIALS IDAS solver.

## Public Functions

**~IDASolver()** **override** = default

*Solver* \***clone()** **const override**

Clone this instance.

**Return** The clone

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)

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 **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)

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)

void **sensToggleOff**() **const override**

Switches off computation of state sensitivities without deallocating the memory for sensitivities.

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

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

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

void **quadSStolerances** (*realtype* *reltolQ*, *realtype* *abstolQ*) **const override**  
sets relative and absolute tolerances for the quadrature problem

**Parameters**

- *reltolQB*: relative tolerances
- *abstolQB*: absolute tolerances

int **solve** (*realtype* *tout*, int *itask*) **const override**  
Solves the forward problem until a predefined timepoint.

**Return** status flag indicating success of execution

**Parameters**

- *tout*: timepoint until which simulation should be performed
- *itask*: task identifier, can be CV\_NORMAL or CV\_ONE\_STEP

int **solveF** (*realtype* *tout*, int *itask*, int \**ncheckPtr*) **const override**  
Solves the forward problem until a predefined timepoint (adjoint only)

**Return** status flag indicating success of execution

**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

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

void **getRootInfo** (int \*rootsfound) **const override**  
 getRootInfo extracts information which event occurred

**Parameters**

- rootsfound: array with flags indicating whether the respective event occurred

void **getDky** (*realtype* t, int k) **const override**  
 interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- t: timepoint
- k: derivative order

void **getSens** () **const override**  
 extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

void **getSensDky** (*realtype* t, int k) **const override**  
 interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- t: timepoint
- k: derivative order

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

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

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

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

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

void **getQuadDky** (*realtype* `t`, int `k`) **const override**

interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- `t`: timepoint
- `k`: derivative order

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)

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)

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

void **turnOffRootFinding** () **const override**

Disable rootfinding.

**const** *Model* \***getModel** () **const override**

Accessor function to the model stored in the user data

**Return** user data model

void **setLinearSolver** () **const override**

Sets the linear solver for the forward problem.

void **setLinearSolverB** (int *which*) **const override**  
 Sets the linear solver for the backward problem.

#### Parameters

- *which*: index of the backward problem

void **setNonLinearSolver** () **const override**  
 Set the non-linear solver for the forward problem.

void **setNonLinearSolverSens** () **const override**  
 Set the non-linear solver for sensitivities.

void **setNonLinearSolverB** (int *which*) **const override**  
 Set the non-linear solver for the backward problem.

#### Parameters

- *which*: index of the backward problem

### 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.

void **allocateSolver** () **const override**  
 Create specifies solver method and initializes solver memory for the forward problem.

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

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

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

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

void **setQuadErrCon** (bool *flag*) **const override**

Specifies whether error control is also enforced for the forward quadrature problem.

**Parameters**

- *flag*: activation flag

void **setErrHandlerFn** () **const override**

Attaches the error handler function (*errMsgIdAndTxt*) to the solver.

void **setUserData** (*Model* \**model*) **const override**

Attaches the user data instance (here this is a *Model*) to the forward problem.

**Parameters**

- *model*: *Model* instance

void **setUserDataB** (int *which*, *Model* \**model*) **const override**

attaches the user data instance (here this is a *Model*) to the backward problem

**Parameters**

- *which*: identifier of the backwards problem
- *model*: *Model* instance

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

void **setStabLimDet** (int *stldet*) **const override**

activates stability limit detection for the forward problem

**Parameters**

- *stldet*: flag for stability limit detection (TRUE or FALSE)



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)

void **setId** (const *Model* \**model*) **const override**  
 specify algebraic/differential components (DAE only)

#### Parameters

- *model*: model specification

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

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

void **adjInit** () **const override**  
 initializes the adjoint problem

void **quadInit** (const *AmiVector* &*xQ0*) **const override**  
 initializes the quadratures

#### Parameters

- *xQ0*: vector with initial values for xQ

void **allocateSolverB** (int \**which*) **const override**  
 Specifies solver method and initializes solver memory for the backward problem.

#### Parameters

- *which*: identifier of the backwards problem

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

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

void **diag** () **const override**  
attaches a diagonal linear solver to the forward problem

void **diagB** (int *which*) **const override**  
attaches a diagonal linear solver to the backward problem

#### Parameters

- *which*: identifier of the backwards problem

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

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

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

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

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

void \***getAdjBmem** (void \**ami\_mem*, int *which*) **const override**  
 Retrieves the solver memory instance for the backward problem.

**Return** A (void \*) pointer to the CVODES memory allocated for the backward problem.

#### Parameters

- *which*: identifier of the backwards problem
- *ami\_mem*: pointer to the forward solver memory instance

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

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

void **sensInit1** (**const** *AmiVectorArray* &*sx0*, **const** *AmiVectorArray* &*sdx0*) **const override**  
 Initializes the forward sensitivities.

#### Parameters

- *sx0*: initial states sensitivities
- *sdx0*: initial derivative states sensitivities

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

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

void **rootInit** (int *ne*) const **override**  
Initializes the rootfinding for events.

**Parameters**

- *ne*: number of different events

void **setDenseJacFn** () const **override**  
Set the dense Jacobian function.

void **setSparseJacFn** () const **override**  
sets the sparse Jacobian function

void **setBandJacFn** () const **override**  
sets the banded Jacobian function

void **setJacTimesVecFn** () const **override**  
sets the Jacobian vector multiplication function

void **setDenseJacFnB** (int *which*) const **override**  
sets the dense Jacobian function

**Parameters**

- *which*: identifier of the backwards problem

void **setSparseJacFnB** (int *which*) const **override**  
sets the sparse Jacobian function

**Parameters**

- *which*: identifier of the backwards problem

void **setBandJacFnB** (int *which*) const **override**  
sets the banded Jacobian function

**Parameters**

- `which`: identifier of the backwards problem

void **setJacTimesVecFnB** (int *which*) **const override**  
 sets the Jacobian vector multiplication function

#### Parameters

- `which`: identifier of the backwards problem

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** `amici::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 `cvoid/ida`
- `t`: time of integration failure

### Public Members

int **error\_code**  
 error code returned by `cvoides/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** `amici::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 ccode/ida
- *t*: time of integration failure

### Public Members

int **error\_code**

error code returned by ccode/ida

*realtype* **time**

time of integration failure

## 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** `amici::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.

#### Return

#### Parameters

- other: Object to copy from

*Model* \*clone() const = 0

Clone this instance.

**Return** The clone

void initialize(*AmiVector* &x, *AmiVector* &dx, *AmiVectorArray* &sx, *AmiVectorArray* &sdx, bool  
computeSensitivities)

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

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 **initHeaviside** (const *AmiVector* &x, const *AmiVector* &dx)

Initialize the Heaviside variables  $h$  at the initial time  $t_0$ .

Heaviside variables activate/deactivate on event occurrences.

#### Parameters

- x: Reference to state variables
- dx: Reference to time derivative of states (DAE only)

int **nplist** () const

Get number of parameters wrt to which sensitivities are computed.

**Return** Length of sensitivity index vector

int **np** () const

Get total number of model parameters.

**Return** Length of parameter vector

int **nk** () const

Get number of constants.

**Return** Length of constant vector

int **ncl** () const

Get number of conservation laws.

**Return** 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.

**Return** *Model* member `nx_solver_reinit`

const double \***k** () const

Get fixed parameters.

**Return** Pointer to constants array

int **nMaxEvent** () const

Get maximum number of events that may occur for each type.

**Return** 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.

**Return** Number of timepoints

`std::vector<ParameterScaling> const &getParameterScale () const`  
Get parameter scale for each parameter.

**Return** 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.

**Return** Unscaled parameters

`std::vector<realtype> const &getParameters () const`  
Get parameter vector.

**Return** 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.

**Return** Parameter value

#### Parameters

- par\_id: Parameter ID

*realtype* **getParameterByName** (std::string const &par\_name) const  
Get value of first model parameter with the specified name.

**Return** Parameter value

#### Parameters

- par\_name: Parameter name

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.

**Return** Number of parameter IDs that matched the regex

#### Parameters

- `par_id_regex`: Parameter ID regex
- `value`: Parameter value

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.

**Return** Number of fixed parameter names that matched the regex

#### Parameters

- `par_name_regex`: Parameter name regex
- `value`: Parameter value

`std::vector<realtype> const &getFixedParameters () const`  
Get values of fixed parameters.

**Return** 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.

**Return** Parameter value

**Parameters**

- `par_id`: Parameter ID

*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.

**Return** Parameter value

**Parameters**

- `par_name`: Parameter name

`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.

**Return** Number of fixed parameter IDs that matched the regex

**Parameters**

- `par_id_regex`: Fixed parameter name regex
- `value`: Fixed parameter value

`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.

**Return** Number of fixed parameter names that matched the regex

**Parameters**

- par\_name\_regex: Fixed parameter name regex
- value: Fixed parameter value

std::string **getName** () const  
Get the model name.

**Return** *Model* name

bool **hasParameterNames** () const  
Report whether the model has parameter names set.

**Return** Boolean indicating whether parameter names were set. Also returns `true` if the number of corresponding variables is just zero.

std::vector<std::string> **getParameterNames** () const  
Get names of the model parameters.

**Return** The parameter names

bool **hasStateNames** () const  
Report whether the model has state names set.

**Return** Boolean indicating whether state names were set. Also returns `true` if the number of corresponding variables is just zero.

std::vector<std::string> **getStateNames** () const  
Get names of the model states.

**Return** State names

bool **hasFixedParameterNames** () const  
Report whether the model has fixed parameter names set.

**Return** Boolean indicating whether fixed parameter names were set. Also returns `true` if the number of corresponding variables is just zero.

std::vector<std::string> **getFixedParameterNames** () const  
Get names of the fixed model parameters.

**Return** Fixed parameter names

bool **hasObservableNames** () const  
Report whether the model has observable names set.

**Return** Boolean indicating whether observable names were set. Also returns `true` if the number of corresponding variables is just zero.

`std::vector<std::string> getObservableNames () const`

Get names of the observables.

**Return** Observable names

`bool hasExpressionNames () const`

Report whether the model has expression names set.

**Return** Boolean indicating whether expression names were set. Also returns `true` if the number of corresponding variables is just zero.

`std::vector<std::string> getExpressionNames () const`

Get names of the expressions.

**Return** Expression names

`bool hasParameterIds () const`

Report whether the model has parameter IDs set.

**Return** Boolean indicating whether parameter IDs were set. Also returns `true` if the number of corresponding variables is just zero.

`std::vector<std::string> getParameterIds () const`

Get IDs of the model parameters.

**Return** Parameter IDs

`bool hasStateIds () const`

Report whether the model has state IDs set.

**Return** Boolean indicating whether state IDs were set. Also returns `true` if the number of corresponding variables is just zero.

`std::vector<std::string> getStateIds () const`

Get IDs of the model states.

**Return** State IDs

`bool hasFixedParameterIds () const`

Report whether the model has fixed parameter IDs set.

**Return** Boolean indicating whether fixed parameter IDs were set. Also returns `true` if the number of corresponding variables is just zero.

`std::vector<std::string> getFixedParameterIds () const`

Get IDs of the fixed model parameters.

**Return** Fixed parameter IDs

`bool hasObservableIds () const`

Report whether the model has observable IDs set.

**Return** Boolean indicating whether observable ids were set. Also returns `true` if the number of corresponding variables is just zero.

```
std::vector<std::string> getObservableIds () const
```

Get IDs of the observables.

**Return** Observable IDs

```
bool hasExpressionIds () const
```

Report whether the model has expression IDs set.

**Return** Boolean indicating whether expression ids were set. Also returns `true` if the number of corresponding variables is just zero.

```
std::vector<std::string> getExpressionIds () const
```

Get IDs of the expression.

**Return** Expression IDs

```
bool hasQuadraticLLH () const
```

Checks whether the defined noise model is gaussian, i.e., the nllh is quadratic.

**Return** boolean flag

```
std::vector<realtype> const &getTimepoints () const
```

Get the timepoint vector.

**Return** Timepoint vector

```
realtype getTimepoint (int it) const
```

Get simulation timepoint for time index `it`.

**Return** Timepoint

**Parameters**

- `it`: Time index

```
void setTimepoints (std::vector<realtype> const &ts)
```

Set the timepoint vector.

**Parameters**

- `ts`: New timepoint vector

```
double t0 () const
```

Get simulation start time.

**Return** 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.

**Return** 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.

*ModelState* **const** &**getModelState** () **const**  
Get the current model state.

**Return** Current model state

void **setModelState** (*ModelState* **const** &*state*)  
Set the current model state.

**Parameters**

- *state*: *ModelState* state

std::vector<int> **const** &**getParameterList** () **const**  
Get the list of parameters for which sensitivities are computed.

**Return** List of parameter indices

int **plist** (int *pos*) **const**  
Get entry in parameter list by index.

**Return** Index in parameter list

**Parameters**

- *pos*: Index in sensitivity 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.

**Return** 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.

**Return** true if has custom initial states, otherwise false

std::vector<*realtype*> **getInitialStateSensitivities** ()

Get the initial states sensitivities.

**Return** 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.

**Return** 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.

**Return** 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.

**Return** 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

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: Timepoint
- 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, 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)
- 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  $J_y$ .

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  $J_y$ .

Partial derivative (to be used with adjoint sensitivities).

#### Parameters

- `sllh`: First order buffer (shape `nplist`)

- `s2llh`: Second order buffer (shape  $(nJ-1) \times nplist$ , 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  $nJ \times nx\_solver$ , 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 real-  
                                type t, const AmiVector &x_old, const AmiVec-  
                                tor &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 real-  
                                   type 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*, **const** char *\*fun*) **const**

Check if the given array has only finite elements.

If not, try to give hints by which other fields this could be caused.

**Return** `amici::AMICI_RECOVERABLE_ERROR` if a NaN/Inf value was found, `amici::AMICI_SUCCESS` otherwise

### Parameters

- `array`: Array to check
- `fun`: Name of the function that generated the values (for more informative messages).

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.

**Return** 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

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`)

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)

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.

**Return** Those indices.

const *AmiVectorArray* &**get\_dxdotdp** () const

getter for `dxdotdp` (matlab generated)

**Return** `dxdotdp`

const *SUNMatrixWrapper* &**get\_dxdotdp\_full** () const

getter for `dxdotdp` (python generated)

**Return** `dxdotdp`

## 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

*AmiciApplication* \***app** = &*defaultContext*

AMICI application context

## 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*> &*s1lh*, 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
- *s1lh*: First order buffer
- *s2llh*: Second order buffer

void **checkLLHBufferSize** (const std::vector<*realtype*> &*s1lh*, const std::vector<*realtype*> &*s2llh*) const

Verify that the provided buffers have the expected size.

### Parameters

- *s1lh*: 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 **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  $J_y$ .

#### 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  $J_y$  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  $J_y$  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  $J_y$  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 **fdrrzdp** (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 **fdrrzdx** (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 **fsmgma** (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 **fdsgmazdp** (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 sigma<sub>z</sub>.

#### 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

void **fx\_rdata**(*realtype* `*x_rdata`, **const** *realtype* `*x_solver`, **const** *realtype* `*tcl`)

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

void **fsx\_rdata**(*realtype* `*sx_rdata`, **const** *realtype* `*sx_solver`, **const** *realtype* `*stcl`, 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
- `ip`: Sensitivity index

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

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

void **ftotal\_cl** (*realtype* \*`total_cl`, **const** *realtype* \*`x_rdata`)

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

void **fstotal\_cl** (*realtype* \*`stotal_cl`, **const** *realtype* \*`sx_rdata`, int `ip`)

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

*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.

**Return** State vector with negative values replaced by 0 according to `stateIsNonNegative`

**Parameters**

- `x`: State vector possibly containing negative values

## 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

`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

## 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.

**Return** Equality

**Parameters**

- `a`: First model instance

- b: Second model instance

## Class Model\_DAE

- Defined in file\_include\_amici\_model\_dae.h

## Inheritance Relationships

### Base Type

- public amici::Model (*Class Model*)

## Class Documentation

**class** amici::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

**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

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

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

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

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

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

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)

```
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

```
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

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

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

void **writeSteadystateJB** (**const** *realtype* *t*, *realtype* *cj*, **const** *AmiVector* &*x*, **const** *AmiVec-*  
*tor* &*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

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)

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

std::unique\_ptr<*Solver*> **getSolver** () **override**  
Retrieves the solver object.

**Return** The *Solver* instance

## Protected Functions

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*

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

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

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

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** amici::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

**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

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

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

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

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

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

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

```
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

```
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

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

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

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 **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

std::unique\_ptr<*Solver*> **getSolver** () **override**  
Retrieves the solver object.

**Return** The *Solver* instance

## Protected Functions

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*

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*

void **fJSparse\_colptrs** (*SUNMatrixWrapper* &*JSparse*)

*Model* specific implementation for fJSparse, column pointers.

### Parameters

- *JSparse*: sparse matrix to which colptrs will be written

void **fJSparse\_rowvals** (*SUNMatrixWrapper* &*JSparse*)

*Model* specific implementation for fJSparse, row values.

### Parameters

- *JSparse*: sparse matrix to which rowvals will be written

```
void froot (realtype *root, realtype t, const realtype *x, const realtype *p, const realtype *k,  
          const realtype *h)  
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

```
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

```
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

```
void fdxdotdp_explicit (realtype *dxdotdp_explicit, realtype t, const realtype *x, const real-  
                       type *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  $\dot{x}$  wrt  $p$
- `t`: timepoint
- `x`: Vector with the states
- `p`: parameter vector
- `k`: constants vector
- `h`: Heaviside vector
- `w`: vector with helper variables

void **fdxdotdp\_explicit\_colptrs** (*SUNMatrixWrapper* &*dxdotdp*)  
*Model* specific implementation of `fdxdotdp_explicit`, `colptrs` part.

**Parameters**

- `dxdotdp`: sparse matrix to which `colptrs` will be written

void **fdxdotdp\_explicit\_rowvals** (*SUNMatrixWrapper* &*dxdotdp*)  
*Model* specific implementation of `fdxdotdp_explicit`, `rowvals` part.

**Parameters**

- `dxdotdp`: sparse matrix to which `rowvals` will be written

void **fdxdotdx\_explicit** (*realtype* \**dxdotdx\_explicit*, *realtype* *t*, **const** *realtype* \**x*, **const** *real-*  
*type* \**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  $\dot{x}$  wrt  $x$
- `t`: timepoint
- `x`: Vector with the states
- `p`: parameter vector
- `k`: constants vector
- `h`: heavyside vector
- `w`: vector with helper variables

void **fdxdotdx\_explicit\_colptrs** (*SUNMatrixWrapper* &*dxdotdx*)  
*Model* specific implementation of `fdxdotdx_explicit`, `colptrs` part.

**Parameters**

- `dxdotdx`: sparse matrix to which `colptrs` will be written

void **fdxdotdx\_explicit\_rowvals** (*SUNMatrixWrapper* &*dxdotdx*)  
*Model* specific implementation of `fdxdotdx_explicit`, `rowvals` part.

**Parameters**

- `dxdotdx`: sparse matrix to which `rowvals` will be written

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

void **fdxdotdw\_colptrs** (*SUNMatrixWrapper* &dxdotdw)  
*Model* specific implementation of fdxdotdw, colptrs part.

**Parameters**

- dxdotdw: sparse matrix to which colptrs will be written

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

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** `amici::ModelContext` : **public** `amici::ContextManager`

The *ModelContext* temporarily stores `amici::Model::state` and restores it when going out of scope.

### Public Functions

**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** amici::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*, const char \**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::NewtonSolverIterative (*Class NewtonSolverIterative*)
- public amici::NewtonSolverSparse (*Class NewtonSolverSparse*)

## Class Documentation

**class** amici::NewtonSolver

The *NewtonSolver* class sets up the linear solver for the Newton method.

Subclassed by *amici::NewtonSolverDense*, *amici::NewtonSolverIterative*, *amici::NewtonSolverSparse*

## Public Functions

**NewtonSolver** (*realtype* \**t*, *AmiVector* \**x*, *Model* \**model*)

Initializes all members with the provided objects.

### Parameters

- *t*: pointer to time variable
- *x*: pointer to state variables
- *model*: pointer to the model object

void **getStep** (int *ntry*, int *nnewt*, *AmiVector* &*delta*)

Computes the solution of one Newton iteration.

### Parameters

- *ntry*: integer *newton\_try* integer start number of Newton solver (1 or 2)
- *nnewt*: integer number of current Newton step
- *delta*: containing the RHS of the linear system, will be overwritten by solution to the linear system

void **computeNewtonSensis** (*AmiVectorArray* &*sx*)

Computes steady state sensitivities.

### Parameters

- *sx*: pointer to state variable sensitivities

**const** std::vector<int> &**getNumLinSteps** () **const**

Accessor for numlinsteps.

**Return** numlinsteps

void **prepareLinearSystem** (int *ntry*, int *nnewt*) = 0

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

### Parameters

- *ntry*: integer *newton\_try* integer start number of Newton solver (1 or 2)
- *nnewt*: integer number of current Newton step

void **prepareLinearSystemB** (int *ntry*, int *nnewt*) = 0

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

### Parameters

- *ntry*: integer *newton\_try* integer start number of Newton solver (1 or 2)
- *nnewt*: integer number of current Newton step

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

`~NewtonSolver()` = default

## Public Members

int `max_lin_steps_` = {0}

maximum number of allowed linear steps per Newton step for steady state computation

int `max_steps` = {0}

maximum number of allowed Newton steps for steady state computation

*realtype* `atol_` = {1e-16}

absolute tolerance

*realtype* `rtol_` = {1e-8}

relative tolerance

*NewtonDampingFactorMode* `damping_factor_mode_` = {*NewtonDampingFactorMode::on*}

damping factor flag

*realtype* `damping_factor_lower_bound` = {1e-8}

damping factor lower bound

## Public Static Functions

`std::unique_ptr<NewtonSolver> getSolver(realtype *t, AmiVector *x, Solver &simulationSolver, Model *model)`

Factory method to create a *NewtonSolver* based on `linsolType`.

**Return** solver *NewtonSolver* according to the specified `linsolType`

### Parameters

- `t`: pointer to time variable
- `x`: pointer to state variables
- `simulationSolver`: solver with settings
- `model`: pointer to the model object

## Protected Attributes

*realtype* \*`t_`

time variable

*Model* \*`model_`

pointer to the model object

*AmiVector* `xdot_`

right hand side *AmiVector*

*AmiVector* \*`x_`

current state

*AmiVector* `dx_`

current state time derivative (DAE)

`std::vector<int> num_lin_steps_`  
history of number of linear steps

*AmiVector* `xB_`  
current adjoint state

*AmiVector* `dxB_`  
current adjoint state time derivative (DAE)

## Class NewtonSolverDense

- Defined in file `include_amici_newton_solver.h`

## Inheritance Relationships

### Base Type

- `public amici::NewtonSolver` (*Class NewtonSolver*)

## Class Documentation

**class** `amici::NewtonSolverDense` : **public** `amici::NewtonSolver`

The *NewtonSolverDense* provides access to the dense linear solver for the Newton method.

### Public Functions

**NewtonSolverDense** (*realtype* \**t*, *AmiVector* \**x*, *Model* \**model*)

Constructor, initializes all members with the provided objects and initializes temporary storage objects.

#### Parameters

- *t*: pointer to time variable
- *x*: pointer to state variables
- *model*: pointer to the model object

**NewtonSolverDense** (**const** *NewtonSolverDense*&) = delete

*NewtonSolverDense* &**operator=** (**const** *NewtonSolverDense* &*other*) = delete

**~NewtonSolverDense** () **override**

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

void **prepareLinearSystem** (int *ntry*, int *nnewt*) **override**

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

#### Parameters

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

void **prepareLinearSystemB** (int *ntry*, int *nnewt*) **override**  
Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

#### Parameters

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

## Class `NewtonSolverIterative`

- Defined in `file_include_amici_newton_solver.h`

## Inheritance Relationships

### Base Type

- `public amici::NewtonSolver` (*Class `NewtonSolver`*)

## Class Documentation

**class** `amici::NewtonSolverIterative` : **public** `amici::NewtonSolver`  
The *NewtonSolverIterative* provides access to the iterative linear solver for the Newton method.

### Public Functions

**NewtonSolverIterative** (*realtype* \**t*, *AmiVector* \**x*, *Model* \**model*)  
Constructor, initializes all members with the provided objects.

#### Parameters

- `t`: pointer to time variable
- `x`: pointer to state variables
- `model`: pointer to the model object

**~NewtonSolverIterative** () **override** = default

void **solveLinearSystem** (*AmiVector* &*rhs*) **override**  
Solves the linear system for the Newton step by passing it to `linsolveSPBCG`.

#### Parameters

- `rhs`: containing the RHS of the linear system, will be overwritten by solution to the linear system

void **prepareLinearSystem** (int *ntry*, int *nnewt*) **override**  
Writes the Jacobian (J) for the Newton iteration and passes it to the linear solver. Also wraps around `getSensis` for iterative linear solver.

**Parameters**

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

void **prepareLinearSystemB** (int *ntry*, int *nnewt*) **override**

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver. Also wraps around `getSensis` for iterative linear solver.

**Parameters**

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

void **linsolveSPBCG** (int *ntry*, int *nnewt*, *AmiVector* &*ns\_delta*)

Iterative linear solver created from SPILS BiCG-Stab. Solves the linear system within each Newton step if iterative solver is chosen.

**Parameters**

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step
- `ns_delta`: Newton step

**Class NewtonSolverSparse**

- Defined in `file_include_amici_newton_solver.h`

**Inheritance Relationships****Base Type**

- `public amici::NewtonSolver` (*Class NewtonSolver*)

**Class Documentation**

**class** `amici::NewtonSolverSparse` : **public** `amici::NewtonSolver`

The *NewtonSolverSparse* provides access to the sparse linear solver for the Newton method.

**Public Functions**

**NewtonSolverSparse** (*realtype* \**t*, *AmiVector* \**x*, *Model* \**model*)

Constructor, initializes all members with the provided objects, initializes temporary storage objects and the `klu` solver.

**Parameters**

- `t`: pointer to time variable
- `x`: pointer to state variables

- `model`: pointer to the model object

**NewtonSolverSparse** (`const` *NewtonSolverSparse*&) = delete

*NewtonSolverSparse* &`operator=` (`const` *NewtonSolverSparse* &*other*) = delete

**~NewtonSolverSparse** () **override**

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

void **prepareLinearSystem** (int *ntry*, int *nnewt*) **override**

Writes the Jacobian for the Newton iteration and passes it to the linear solver.

#### Parameters

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

void **prepareLinearSystemB** (int *ntry*, int *nnewt*) **override**

Writes the Jacobian (JB) for the Newton iteration and passes it to the linear solver

#### Parameters

- `ntry`: integer `newton_try` integer start number of Newton solver (1 or 2)
- `nnewt`: integer number of current Newton step

## Class ReturnData

- Defined in `file_include_amici_rdata.h`

## Inheritance Relationships

### Base Type

- `public` `amici::ModelDimensions` (*Struct ModelDimensions*)

## Class Documentation

**class** `amici::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)

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

**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::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 nz`, row-major)

`std::vector<realtype> ssigmaz`  
parameter derivative of event output standard deviation (shape `nmaxevent 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 nz x nplist`, 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]

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)

std::vector<int> preeq_numlinsteps
    number of linear steps by Newton step for steady state problem. this will only be filled for iterative solvers
    (preequilibration) (shape newton_maxsteps * 2)

```

`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)

`std::vector<int> posteq_numlinsteps`  
number of linear steps by Newton step for steady state problem. this will only be filled for iterative solvers (postequilibration) (shape `newton_maxsteps * 2`)

`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`  
 $\chi^2$  value

`std::vector<realtype> s11h`  
parameter derivative of log-likelihood (shape `nplist`)

`std::vector<realtype> s211h`  
second-order parameter derivative of log-likelihood (shape `nJ-1 x nplist`, row-major)

`int status = 0`  
status code

`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

```

## 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>

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*)

Chi-squared function.

**Parameters**

- `it`: time index

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

bool **computingFSA** () const  
Checks whether forward sensitivity analysis is performed.

**Return** 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, *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
- 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* **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

## Class SimulationParameters

- Defined in file\_include\_amici\_simulation\_parameters.h

## Inheritance Relationships

### Derived Type

- `public amici::ExpData (Class ExpData)`

## Class Documentation

**class amici::SimulationParameters**

Container for various simulation parameters.

Subclassed by *amici::ExpData*

### Public Functions

**SimulationParameters** () = default

**SimulationParameters** (std::vector<realtype> timepoints)

Constructor.

#### Parameters

- `timepoints`: Timepoints for which simulation results are requested

**SimulationParameters** (std::vector<realtype> fixedParameters, std::vector<realtype> parameters)

Constructor.

**Parameters**

- fixedParameters: *Model* constants
- parameters: *Model* parameters

**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

**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\_idxes\_presim and reinitialization\_state\_idxes\_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\_idxes\_presim and reinitialization\_state\_idxes\_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\_idxes\_presim and reinitialization\_state\_idxes\_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. Overrides `Solver::newton_preeq`

`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 fixed-Parameters 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_idxsim
```

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** amici::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::readSolverSettingsFromHDF5` in `hdf5.cpp`.

Subclassed by *amici::CvodeSolver*, *amici::IDASolver*

## Public Functions

**Solver** () = default

**Solver** (*AmiciApplication* \*app)  
Constructor.

### Parameters

- app: AMICI application context

**Solver** (const *Solver* &other)  
*Solver* copy constructor.

### Parameters

- other:

**~Solver** () = default

*Solver* \***clone** () const = 0  
Clone this instance.

**Return** The clone

int **run** (*realtype* tout) **const**  
runs a forward simulation until the specified timepoint

**Return** status flag

**Parameters**

- tout: next timepoint

int **step** (*realtype* tout) **const**  
makes a single step in the simulation

**Return** status flag

**Parameters**

- tout: next timepoint

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)
```

Reinitializes state and respective sensitivities (if necessary) according to changes in fixedParameters.

#### Parameters

- `model`: pointer to the model instance

```
void getRootInfo (int *rootsfound) const = 0
```

`getRootInfo` extracts information which event occurred

#### Parameters

- `rootsfound`: array with flags indicating whether the respective event occurred

```
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)

```
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)

```
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`

void **turnOffRootFinding** () **const** = 0  
Disable rootfinding.

*SensitivityMethod* **getSensitivityMethod** () **const**  
Return current sensitivity method.

**Return** method enum

void **setSensitivityMethod** (*SensitivityMethod* sensi\_meth)  
Set sensitivity method.

**Parameters**

- sensi\_meth:

*SensitivityMethod* **getSensitivityMethodPreequilibration** () **const**  
Return current sensitivity method during preequilibration.

**Return** 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.

**Return**

void **setNewtonMaxSteps** (int newton\_maxsteps)  
Set maximum number of allowed Newton steps for steady state computation.

**Parameters**

- newton\_maxsteps:

bool **getPreequilibration** () **const**  
Get if model preequilibration is enabled.

**Return**

void **setPreequilibration** (bool require\_preequilibration)  
Enable/disable model preequilibration.

**Parameters**

- require\_preequilibration:

int **getNewtonMaxLinearSteps** () **const**  
Get maximum number of allowed linear steps per Newton step for steady state computation.



**Return**

void **setNewtonMaxLinearSteps** (int *newton\_maxlinsteps*)  
 Set maximum number of allowed linear steps per Newton step for steady state computation.

**Parameters**

- *newton\_maxlinsteps*:

*NewtonDampingFactorMode* **getNewtonDampingFactorMode** () **const**

Get a state of the damping factor used in the Newton solver.

**Return**

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.

**Return**

void **setNewtonDampingFactorLowerBound** (double *dampingFactorLowerBound*)  
 Set a lower bound of the damping factor in the Newton solver.

**Parameters**

- *dampingFactorLowerBound*:

*SensitivityOrder* **getSensitivityOrder** () **const**

Get sensitivity order.

**Return** 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`.

**Return** 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`.

**Return** 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.

**Return** 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.

**Return** 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.

**Return** 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.

**Return** 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.

**Return** 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

**Return** absolute tolerance

void **setAbsoluteToleranceQuadratures** (double *atol*)

sets the absolute tolerance for the quadrature problem

**Parameters**

- *atol*: absolute tolerance (non-negative number)

double **getRelativeToleranceSteadyState** () **const**

returns the relative tolerance for the steady state problem

**Return** 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

**Return** absolute tolerance

void **setAbsoluteToleranceSteadyState** (double *atol*)

sets the absolute tolerance for the steady state problem

**Parameters**

- `atol`: absolute tolerance (non-negative number)

double **getRelativeToleranceSteadyStateSensi** () **const**  
returns the relative tolerance for the sensitivities of the steady state problem

**Return** 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

**Return** 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

**Return** 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)

long int **getMaxStepsBackwardProblem** () **const**  
returns the maximum number of solver steps for the backward problem

**Return** 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

**Return** 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

**Return**

void **setNonlinearSolverIteration** (*NonlinearSolverIteration* *iter*)  
sets the nonlinear system solution method

**Parameters**

- *iter*: nonlinear system solution method

*InterpolationType* **getInterpolationType** () **const**  
getInterpolationType

**Return**

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.

**Return** State-ordering as integer according to *SUNLinSolKLU::StateOrdering* or *SUNLinSolSuperLUMT::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

**Return** *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

**Return**

void **setLinearSolver** (*LinearSolver* linsol)  
setLinearSolver

**Parameters**

- `linsol`:

*InternalSensitivityMethod* **getInternalSensitivityMethod**() **const**  
returns the internal sensitivity method

**Return** 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

**Return** *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 forward 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`.

**Return** `x` or interpolated solution `dky`

**Parameters**

- `t`: time

**const** *AmiVector* &getDerivativeState (*realtype t*) **const**

Access derivative state solution at time `t`.

**Return** `dx` or interpolated solution `dky`

**Parameters**

- `t`: time

**const** *AmiVectorArray* &getStateSensitivity (*realtype t*) **const**

Access state sensitivity solution at time `t`.

**Return** (interpolated) solution `sx`

**Parameters**

- `t`: time

**const** *AmiVector* &getAdjointState (int *which*, *realtype t*) **const**

Access adjoint solution at time `t`.

**Return** (interpolated) solution `xB`

**Parameters**

- `which`: adjoint problem index
- `t`: time

**const** *AmiVector* &getAdjointDerivativeState (int *which*, *realtype t*) **const**

Access adjoint derivative solution at time `t`.

**Return** (interpolated) solution `dxB`

**Parameters**

- `which`: adjoint problem index
- `t`: time

**const** *AmiVector* &**getAdjointQuadrature** (int *which*, *realtype* *t*) **const**  
Access adjoint quadrature solution at time *t*.

**Return** (interpolated) solution *xQB*

**Parameters**

- *which*: adjoint problem index
- *t*: time

**const** *AmiVector* &**getQuadrature** (*realtype* *t*) **const**  
Access quadrature solution at time *t*.

**Return** (interpolated) solution *xQ*

**Parameters**

- *t*: time

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)

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)

void **sensToggleOff** () **const** = 0  
Switches off computation of state sensitivities without deallocating the memory for sensitivities.

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

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

**Return** *t*

*realtype* **getCpuTime () const**  
Reads out the CPU time needed for forward solve.

**Return** *cpu\_time*

*realtype* **getCpuTimeB () const**  
Reads out the CPU time needed for backward solve.

**Return** *cpu\_timeB*

**int nx () const**  
number of states with which the solver was initialized

**Return** *x.getLength()*

**int nplist () const**  
number of parameters with which the solver was initialized

**Return** *sx.getLength()*

**int nquad () const**  
number of quadratures with which the solver was initialized

**Return** *xQB.getLength()*

**bool computingFSA () const**  
check if FSA is being computed

**Return** *flag*

**bool computingASA () const**  
check if ASA is being computed

**Return** *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

`std::vector<int> const &getNumSteps () const`  
Accessor ns.

**Return** ns

`std::vector<int> const &getNumStepsB () const`  
Accessor nsB.

**Return** nsB

`std::vector<int> const &getNumRhsEvals () const`  
Accessor nrhs.

**Return** nrhs

`std::vector<int> const &getNumRhsEvalsB () const`  
Accessor nrhsB.

**Return** nrhsB

`std::vector<int> const &getNumErrTestFails () const`  
Accessor netf.

**Return** netf

`std::vector<int> const &getNumErrTestFailsB () const`  
Accessor netfB.

**Return** netfB

`std::vector<int> const &getNumNonlinSolvConvFails () const`  
Accessor nnlsf.

**Return** nnlsf

`std::vector<int> const &getNumNonlinSolvConvFailsB () const`  
Accessor nnlsfB.

**Return** nnlsfB

`std::vector<int> const &getLastOrder () const`  
Accessor order.

**Return** order

## Public Members

*AmiciApplication* \*app = &defaultContext  
AMICI context

## Protected Functions

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

int **solve** (*realtype* tout, int itask) **const** = 0  
Solves the forward problem until a predefined timepoint.

**Return** status flag indicating success of execution

### Parameters

- tout: timepoint until which simulation should be performed
- itask: task identifier, can be CV\_NORMAL or CV\_ONE\_STEP

int **solveF** (*realtype* tout, int itask, int \*ncheckPtr) **const** = 0  
Solves the forward problem until a predefined timepoint (adjoint only)

**Return** status flag indicating success of execution

### 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

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)

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)

void **getSens** () **const** = 0  
extracts the state sensitivity at the current timepoint from solver memory and writes it to the sx member variable

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

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

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

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

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

void **sensInit1** (**const** *AmiVectorArray* &*sx0*, **const** *AmiVectorArray* &*sdx0*) **const** = 0

Initializes the forward sensitivities.

**Parameters**

- `sx0`: initial states sensitivities
- `sdx0`: initial derivative states sensitivities

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

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

void **rootInit** (int *ne*) const = 0  
Initializes the rootfinding for events.

#### Parameters

- *ne*: number of different events

void **initializeNonLinearSolverSens** (const *Model* \**model*) const  
Initialize non-linear solver for sensitivities.

#### Parameters

- *model*: *Model* instance

void **setDenseJacFn** () const = 0  
Set the dense Jacobian function.

void **setSparseJacFn** () const = 0  
sets the sparse Jacobian function

void **setBandJacFn** () const = 0  
sets the banded Jacobian function

void **setJacTimesVecFn** () const = 0  
sets the Jacobian vector multiplication function

void **setDenseJacFnB** (int *which*) const = 0  
sets the dense Jacobian function

#### Parameters

- *which*: identifier of the backwards problem

void **setSparseJacFnB** (int *which*) const = 0  
sets the sparse Jacobian function

#### Parameters

- *which*: identifier of the backwards problem

void **setBandJacFnB** (int *which*) const = 0  
sets the banded Jacobian function

#### Parameters

- *which*: identifier of the backwards problem

void **setJacTimesVecFnB** (int *which*) **const** = 0  
sets the Jacobian vector multiplication function

**Parameters**

- *which*: identifier of the backwards problem

void **setSparseJacFn\_ss** () **const** = 0  
sets the sparse Jacobian function for backward steady state case

void **allocateSolver** () **const** = 0  
Create specifies solver method and initializes solver memory for the forward problem.

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

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

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

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

void **setQuadErrCon** (bool *flag*) **const** = 0  
Specifies whether error control is also enforced for the forward quadrature problem.

**Parameters**

- *flag*: activation flag

void **setErrorHandlerFn** () **const** = 0  
Attaches the error handler function (*errMsgIdAndTxt*) to the solver.

void **setUserData** (*Model* *\*model*) **const** = 0  
Attaches the user data instance (here this is a *Model*) to the forward problem.

**Parameters**

- `model`: *Model* instance

void **setUserDataB** (int *which*, *Model* \**model*) **const** = 0  
 attaches the user data instance (here this is a *Model*) to the backward problem

**Parameters**

- *which*: identifier of the backwards problem
- `model`: *Model* instance

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

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

void **setStabLimDet** (int *stldet*) **const** = 0  
 activates stability limit detection for the forward problem

**Parameters**

- *stldet*: flag for stability limit detection (TRUE or FALSE)

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)

void **setId** (**const** *Model* \**model*) **const** = 0  
 specify algebraic/differential components (DAE only)

**Parameters**

- `model`: model specification

void **setSuppressAlg** (bool *flag*) **const** = 0  
 deactivates error control for algebraic components (DAE only)

**Parameters**

- `flag`: deactivation flag

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

void **getDky** (*realtype* *t*, int *k*) **const** = 0  
interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- *t*: timepoint
- *k*: derivative order

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

void **getSensDky** (*realtype* *t*, int *k*) **const** = 0  
interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- *t*: timepoint
- *k*: derivative order

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

void **getQuadDky** (*realtype* *t*, int *k*) **const** = 0  
interpolates the (derivative of the) solution at the requested timepoint

**Parameters**

- *t*: timepoint



- `k`: derivative order

void **adjInit** () **const** = 0  
initializes the adjoint problem

void **quadInit** (**const** *AmiVector* &*xQ0*) **const** = 0  
initializes the quadratures

#### Parameters

- `xQ0`: vector with initial values for `xQ`

void **allocateSolverB** (int \**which*) **const** = 0  
Specifies solver method and initializes solver memory for the backward problem.

#### Parameters

- `which`: identifier of the backwards problem

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

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

void **quadSStolerances** (*realtype* *reltolQB*, *realtype* *abstolQB*) **const** = 0  
sets relative and absolute tolerances for the quadrature problem

#### Parameters

- `reltolQB`: relative tolerances
- `abstolQB`: absolute tolerances

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

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

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

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

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.

void **setLinearSolver** () **const** = 0  
Sets the linear solver for the forward problem.

void **setLinearSolverB** (int *which*) **const** = 0  
Sets the linear solver for the backward problem.

**Parameters**

- *which*: index of the backward problem

void **setNonLinearSolver** () **const** = 0  
Set the non-linear solver for the forward problem.

void **setNonLinearSolverB** (int *which*) **const** = 0  
Set the non-linear solver for the backward problem.

**Parameters**

- *which*: index of the backward problem

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

**const** *Model* \***getModel** () **const** = 0  
Accessor function to the model stored in the user data

**Return** user data model

bool **getInitDone** () **const**  
checks whether memory for the forward problem has been allocated

**Return** proxy for solverMemory->(cvlida)\_MallocDone

bool **getSensInitDone** () **const**  
checks whether memory for forward sensitivities has been allocated

**Return** proxy for solverMemory->(cvlida)\_SensMallocDone

bool **getAdjInitDone** () **const**  
checks whether memory for forward interpolation has been allocated

**Return** proxy for solverMemory->(cvlida)\_adjMallocDone

bool **getInitDoneB** (int *which*) **const**  
checks whether memory for the backward problem has been allocated

**Return** proxy for solverMemoryB->(cvlida)\_MallocDone

**Parameters**

- *which*: adjoint problem index

bool **getQuadInitDoneB** (int *which*) **const**  
checks whether memory for backward quadratures has been allocated

**Return** proxy for solverMemoryB->(cvlida)\_QuadMallocDone

**Parameters**

- *which*: adjoint problem index

bool **getQuadInitDone** () **const**  
checks whether memory for quadratures has been allocated

**Return** proxy for solverMemory->(cvlida)\_QuadMallocDone

void **diag** () **const** = 0  
attaches a diagonal linear solver to the forward problem

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)

void **\*getAdjBmem** (void *\*ami\_mem*, int *which*) **const** = 0  
Retrieves the solver memory instance for the backward problem.

**Return** A (void \*) pointer to the CVODES memory allocated for the backward problem.

**Parameters**

- *which*: identifier of the backwards problem
- *ami\_mem*: pointer to the forward solver memory instance

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*[\_preeq]
- *preequilibration*: flag indicating preequilibration or simulation

**Protected Attributes**

std::unique\_ptr<void, std::function<void (void\*) >> **solver\_memory\_**  
pointer to solver memory block

std::vector<std::unique\_ptr<void, std::function<void (void\*) >>> **solver\_memory\_B\_**  
pointer to solver memory block

*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::unique\_ptr<*SUNLinSolWrapper*> **linear\_solver\_**  
linear solver for the forward problem

std::unique\_ptr<*SUNLinSolWrapper*> **linear\_solver\_B\_**  
linear solver for the backward problem

std::unique\_ptr<*SUNNonLinSolWrapper*> **non\_linear\_solver\_**  
non-linear solver for the forward problem

std::unique\_ptr<*SUNNonLinSolWrapper*> **non\_linear\_solver\_B\_**  
non-linear solver for the backward problem

std::unique\_ptr<*SUNNonLinSolWrapper*> **non\_linear\_solver\_sens\_**  
non-linear solver for the sensitivities

bool **solver\_was\_called\_F\_** = {false}  
flag indicating whether the forward solver has been called

bool **solver\_was\_called\_B\_** = {false}  
flag indicating whether the backward solver has been called

*AmiVector* **x\_** = {0}  
state (dimension: nx\_solver)

*AmiVector* **dky\_** = {0}  
state interface variable (dimension: nx\_solver)

*AmiVector* **dx\_** = {0}  
state derivative dummy (dimension: nx\_solver)

*AmiVectorArray* **sx\_** = {0, 0}  
state sensitivities interface variable (dimension: nx\_solver x nplist)

*AmiVectorArray* **sdx\_** = {0, 0}  
state derivative sensitivities dummy (dimension: nx\_solver x nplist)

*AmiVector* **xB\_** = {0}  
adjoint state interface variable (dimension: nx\_solver)

*AmiVector* **dxB\_** = {0}  
adjoint derivative dummy variable (dimension: nx\_solver)

*AmiVector* **xQB\_** = {0}  
adjoint quadrature interface variable (dimension: nJ x nplist)

*AmiVector* **xQ\_** = {0}  
forward quadrature interface variable (dimension: nx\_solver)

*realtype* **t\_** = {std::nan("")}  
integration time of the forward problem

```
bool force_reinit_postprocess_F_ = {false}
    flag to force reInitPostProcessF before next call to solve
bool force_reinit_postprocess_B_ = {false}
    flag to force reInitPostProcessB before next call to solveB
```

## 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.
```

### Return

### Parameters

- a:
- b:

## Class SteadystateProblem

- Defined in file\_include\_amici\_steadystateproblem.h

## Class Documentation

```
class amici::SteadystateProblem
```

The *SteadystateProblem* class solves a steady-state problem using Newton's method and falls back to integration on failure.

## Public Functions

```
SteadystateProblem (const Solver &solver, const Model &model)
    constructor
```

### Parameters

- solver: *Solver* instance
- model: *Model* instance

```
void workSteadyStateProblem (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** (*Solver* \*`solver`, *Model* \*`model`, const *Backward-Problem* \*`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

void **findSteadyState** (*Solver* \*`solver`, *NewtonSolver* \*`newtonSolver`, *Model* \*`model`, int `it`)

Handles the computation of the steady state, throws an *AmiException*, if no steady state was found.

**Parameters**

- `solver`: pointer to the solver object
- `newtonSolver`: pointer to the newtonSolver solver object
- `model`: pointer to the model object
- `it`: integer with the index of the current time step

void **findSteadyStateByNewtonsMethod** (*NewtonSolver* \*`newtonSolver`, *Model* \*`model`, bool `newton_retry`)

Tries to determine the steady state by using Newton's method.

**Parameters**

- `newtonSolver`: pointer to the newtonSolver solver object
- `model`: pointer to the model object
- `newton_retry`: bool flag indicating whether being relaunched

void **findSteadyStateBySimulation** (const *Solver* \*`solver`, *Model* \*`model`, int `it`)

Tries to determine the steady state by using forward simulation.

**Parameters**

- `solver`: pointer to the solver object
- `model`: pointer to the model object
- `it`: integer with the index of the current time step

void **computeSteadyStateQuadrature** (*NewtonSolver* \*`newtonSolver`, const *Solver* \*`solver`, *Model* \*`model`)

Handles the computation of quadratures in adjoint mode.

**Parameters**

- `newtonSolver`: pointer to the newtonSolver solver object



- `solver`: pointer to the solver object
- `model`: pointer to the model object

void **getQuadratureByLinSolve** (*NewtonSolver* \**newtonSolver*, *Model* \**model*)

Computes the quadrature in steady state backward mode by solving the linear system defined by the backward Jacobian.

#### Parameters

- `newtonSolver`: pointer to the newtonSolver solver object
- `model`: pointer to the model object

void **getQuadratureBySimulation** (**const** *Solver* \**solver*, *Model* \**model*)

Computes the quadrature in steady state backward mode by numerical integration of xB forward in time.

#### Parameters

- `solver`: pointer to the solver object
- `model`: pointer to the model object

void **handleSteadyStateFailure** (**const** *Solver* \**solver*, *Model* \**model*)

Stores state and throws an exception if equilibration failed.

#### Parameters

- `solver`: pointer to the solver object
- `model`: pointer to the model object

void **writeErrorString** (std::string \**errorString*, *SteadyStateStatus* *status*) **const**

Assembles the error message to be thrown.

#### Parameters

- `errorString`: const pointer to string with error message
- `status`: Entry of `steady_state_status` to be processed

bool **getSensitivityFlag** (**const** *Model* \**model*, **const** *Solver* \**solver*, int *it*, *SteadyStateContext* *context*)

Checks depending on the status of the Newton solver, solver settings, and the model, whether state sensitivities still need to be computed via a linear system solve or stored.

**Return** flag telling how to process state sensitivities

#### Parameters

- `model`: pointer to the model object
- `solver`: pointer to the solver object
- `it`: integer with the index of the current time step
- `context`: `SteadyStateContext` giving the situation for the flag

*realtype* **getWrmsNorm** (*AmiVector* const &x, *AmiVector* const &xdot, *realtype* atol, *realtype* rtol, *AmiVector* &ewt) const

Computes the weighted root mean square of xdot the weights are computed according to x:  $w_i = 1 / (rtol * x_i + atol)$

**Return** root-mean-square norm

**Parameters**

- x: current state
- xdot: current rhs
- atol: absolute tolerance
- rtol: relative tolerance
- ewt: error weight vector

bool **checkConvergence** (const *Solver* \*solver, *Model* \*model, *SensitivityMethod* checkSensitivities)

Checks convergence for state and respective sensitivities.

**Return** boolean indicating convergence

**Parameters**

- solver: *Solver* instance
- model: instance
- checkSensitivities: flag whether sensitivities should be checked

void **applyNewtonsMethod** (*Model* \*model, *NewtonSolver* \*newtonSolver, bool newton\_retry)

Runs the Newton solver iterations and checks for convergence to steady state.

**Parameters**

- model: pointer to the model object
- newtonSolver: pointer to the *NewtonSolver* object
- newton\_retry: flag indicating if Newton solver is rerun

void **runSteadystateSimulation** (const *Solver* \*solver, *Model* \*model, bool backward)

Simulation is launched, if Newton solver or linear system solve fails.

**Parameters**

- solver: pointer to the solver object
- model: pointer to the model object
- backward: flag indicating adjoint mode (including quadrature)

std::unique\_ptr<*Solver*> **createSteadystateSimSolver** (const *Solver* \*solver, *Model* \*model, bool forwardSensis, bool backward) const

Initialize *CVodeSolver* instance for preequilibration simulation.

**Return** solver instance

**Parameters**

- `solver`: pointer to the solver object
- `model`: pointer to the model object
- `forwardSensis`: flag switching on integration with FSA
- `backward`: flag switching on quadratures computation

bool **initializeBackwardProblem** (*Solver* \*`solver`, *Model* \*`model`, const *BackwardProblem* \*`bwd`)

Initialize backward computation by setting state, time, adjoint state and checking for preequilibration mode.

**Return** flag indicating whether backward computation to be carried out

#### Parameters

- `solver`: pointer to the solver object
- `model`: pointer to the model object
- `bwd`: pointer to backward problem

void **computeQBfromQ** (*Model* \*`model`, const *AmiVector* &`yQ`, *AmiVector* &`yQB`) const

Compute the backward quadratures, which contribute to the gradient (xQB) from the quadrature over the backward state itself (xQ)

#### Parameters

- `model`: pointer to the model object
- `yQ`: vector to be multiplied with `dxdotdp`
- `yQB`: resulting vector after multiplication

void **storeSimulationState** (*Model* \*`model`, bool `storesensi`)

Store carbon copy of current simulation state variables as *SimulationState*.

#### Parameters

- `model`: model carrying the *ModelState* to be used
- `storesensi`: flag to enable storage of sensitivities

const *SimulationState* &**getFinalSimulationState** () const

Returns the stored *SimulationState*.

**Return** stored *SimulationState*

const *AmiVector* &**getEquilibrationQuadratures** () const

Returns the quadratures from pre- or postequilibration.

**Return** xQB Vector with quadratures

const *AmiVector* &**getState** () const

Returns state at steadystate.

**Return** x

**const** *AmiVectorArray* &getStateSensitivity () **const**

Returns state sensitivity at steadystate.

**Return** *sx*

std::vector<*realtype*> **const** &getDJydx () **const**

Accessor for dJydx.

**Return** *dJydx*

double **getCPUTime** () **const**

Accessor for run\_time of the forward problem.

**Return** *run\_time*

double **getCPUTimeB** () **const**

Accessor for run\_time of the backward problem.

**Return** *run\_time*

std::vector<*SteadyStateStatus*> **const** &getSteadyStateStatus () **const**

Accessor for steady\_state\_status.

**Return** *steady\_state\_status*

*realtype* **getSteadyStateTime** () **const**

Accessor for t.

**Return** *t*

*realtype* **getResidualNorm** () **const**

Accessor for wrms.

**Return** *wrms*

**const** std::vector<int> &getNumSteps () **const**

Accessor for numsteps.

**Return** *numsteps*

int **getNumStepsB** () **const**

Accessor for numstepsB.

**Return** *numstepsB*

**const** std::vector<int> &getNumLinSteps () **const**

Accessor for numlinsteps.

**Return** *numlinsteps*

void **getAdjointUpdates** (*Model* &model, **const** *ExpData* &edata)

computes adjoint updates dJydx according to provided model and expdata

**Parameters**

- `model`: *Model* instance
- `edata`: experimental data

*AmiVector* **const &getAdjointQuadrature () const**

Accessor for xQB.

**Return** xQB

bool **hasQuadrature () const**

Accessor for hasQuadrature\_.

**Return** hasQuadrature\_

bool **checkSteadyStateSuccess () const**

computes adjoint updates dJydx according to provided model and expdata

**Return** 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** `amici::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

SUNMatrix **getMatrix()** **const override**  
Get the matrix A (matrix solvers only).

**Return** A

## Class SUNLinSolDense

- Defined in `file_include_amici_sundials_linsol_wrapper.h`

## Inheritance Relationships

### Base Type

- `public amici::SUNLinSolWrapper` (*Class SUNLinSolWrapper*)

## Class Documentation

**class** `amici::SUNLinSolDense` : **public** `amici::SUNLinSolWrapper`  
SUNDIALS dense direct solver.

### Public Functions

**SUNLinSolDense** (*AmiVector* **const** &x)  
Create dense solver.

#### Parameters

- `x`: A template for cloning vectors needed within the solver.

SUNMatrix **getMatrix()** **const override**  
Get the matrix A (matrix solvers only).

**Return** A

## Class SUNLinSolKLU

- Defined in `file_include_amici_sundials_linsol_wrapper.h`

## Inheritance Relationships

### Base Type

- `public amici::SUNLinSolWrapper (Class SUNLinSolWrapper)`

### Class Documentation

**class** `amici::SUNLinSolKLU`: **public** `amici::SUNLinSolWrapper`  
 SUNDIALS KLU sparse direct solver.

### Public Types

**enum** `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*:

`SUNMatrix` **getMatrix** () **const override**

Get the matrix *A* (matrix solvers only).

**Return** *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** amici::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).

#### Return

#### Parameters

- *A\_data*:



- `ATimes:`

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).

**Return**

**Parameters**

- `P_data:`
- `Pset:`
- `Psol:`

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).

**Return**

**Parameters**

- `s:`
- `nul:`

int **getNumIters** () **const**

Returns the number of linear iterations performed in the last ‘Solve’ call.

**Return** Number of iterations

*realtype* **getResNorm** () **const**

Returns the final residual norm from the last ‘Solve’ call.

**Return** residual norm

N\_Vector **getResid** () **const**

Get preconditioned initial residual (see sundials/sundials\_linearsolver.h).

**Return**

## Class SUNLinSolSPBCGS

- Defined in file\_include\_amici\_sundials\_linsol\_wrapper.h

## Inheritance Relationships

### Base Type

- `public amici::SUNLinSolWrapper (Class SUNLinSolWrapper)`

### Class Documentation

**class** `amici::SUNLinSolSPBCGS` : **public** `amici::SUNLinSolWrapper`  
SUNDIALS scaled preconditioned Bi-CGStab (Bi-Conjugate Gradient Stable method) (SPBCGS) solver.

#### Public Functions

**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

**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).

##### Return

##### Parameters

- *A\_data*:
- *ATimes*:

`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).

##### Return

##### Parameters

- *P\_data*:

- Pset:
- Psol:

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).

**Return**

**Parameters**

- *s*:
- *nul*:

int **getNumIters** () **const**

Returns the number of linear iterations performed in the last ‘Solve’ call.

**Return** Number of iterations

*realtype* **getResNorm** () **const**

Returns the final residual norm from the last ‘Solve’ call.

**Return** residual norm

N\_Vector **getResid** () **const**

Get preconditioned initial residual (see sundials/sundials\_linearsolver.h).

**Return**

## Class SUNLinSolSPFGMR

- Defined in file\_include\_amici\_sundials\_linsol\_wrapper.h

## Inheritance Relationships

### Base Type

- public amici::SUNLinSolWrapper (*Class SUNLinSolWrapper*)

## Class Documentation

**class** amici::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).

### Return

#### Parameters

- A\_data:
- ATimes:

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).

### Return

#### Parameters

- P\_data:
- Pset:
- Psol:

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).

### Return

#### Parameters

- s:
- nul:

int **getNumIters** () **const**  
Returns the number of linear iterations performed in the last ‘Solve’ call.

**Return** Number of iterations

*realtype* **getResNorm** () **const**  
Returns the final residual norm from the last ‘Solve’ call.

**Return** residual norm

N\_Vector **getResid()** **const**

Get preconditioned initial residual (see sundials/sundials\_linearsolver.h).

**Return**

## Class SUNLinSolSPGMR

- Defined in file\_include\_amici\_sundials\_linsol\_wrapper.h

## Inheritance Relationships

### Base Type

- public amici::SUNLinSolWrapper (*Class SUNLinSolWrapper*)

## Class Documentation

**class** amici::SUNLinSolSPGMR : public amici::SUNLinSolWrapper

SUNDIALS scaled preconditioned GMRES (Generalized Minimal Residual method) solver (SPGMR).

### Public Functions

**SUNLinSolSPGMR**(*AmiVector* const &x, int pretype = PREC\_NONE, int maxl = SUN-  
SPGMR\_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).

#### Return

#### Parameters

- A\_data:
- ATimes:

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).

#### Return

#### Parameters

- P\_data:

- Pset:
- Psol:

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).

**Return**

**Parameters**

- *s*:
- *nul*:

int **getNumIters** () **const**

Returns the number of linear iterations performed in the last ‘Solve’ call.

**Return** Number of iterations

*realtype* **getResNorm** () **const**

Returns the final residual norm from the last ‘Solve’ call.

**Return** residual norm

N\_Vector **getResid** () **const**

Get preconditioned initial residual (see sundials/sundials\_linearsolver.h).

**Return**

## Class SUNLinSolSPTFQMR

- Defined in file\_include\_amici\_sundials\_linsol\_wrapper.h

## Inheritance Relationships

### Base Type

- public amici::SUNLinSolWrapper (*Class SUNLinSolWrapper*)

## Class Documentation

**class** amici::SUNLinSolSPTFQMR : public amici::SUNLinSolWrapper

SUNDIALS scaled preconditioned TFQMR (Transpose-Free Quasi-Minimal Residual method) (SPTFQMR) solver.

## Public Functions

**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

**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).

### Return

### Parameters

- *A\_data*:
- *ATimes*:

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).

### Return

### Parameters

- *P\_data*:
- *Pset*:
- *Psol*:

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).

### Return

### Parameters

- *s*:
- *nul*:

`int getNumIters () const`

Returns the number of linear iterations performed in the last ‘Solve’ call.

**Return** Number of iterations

*realtype* `getResNorm () const`

Returns the final residual norm from the last ‘Solve’ call.

**Return** residual norm

`N_Vector getResid () const`

Get preconditioned initial residual (see sundials/sundials\_linearsolver.h).

**Return**

## 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** `amici::SUNLinSolWrapper`

A RAI 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

**SUNLinSolWrapper** (SUNLinearSolver *linsol*)  
Wrap existing SUNLinearSolver.

### Parameters

- *linsol*:

**~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.

### Return

### Parameters

- *other*:

*SUNLinSolWrapper* &**operator=** (*SUNLinSolWrapper* &&*other*) **noexcept**  
Move assignment.

### Return

### Parameters

- *other*:

SUNLinearSolver **get** () **const**  
Returns the wrapped SUNLinSol.

**Return** SUNLinearSolver

SUNLinearSolver\_Type **getType** () **const**  
Returns an identifier for the linear solver type.

### Return

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  $A \cdot x = b$ .

**Return** error flag

**Parameters**

- A:
- x: A template for cloning vectors needed within the solver.
- b:
- tol: Tolerance (weighted 2-norm), iterative solvers only

long int **getLastFlag** () **const**

Returns the last error flag encountered within the linear solver.

**Return** error flag

int **space** (long int \*lenrwLS, long int \*leniwLS) **const**

Returns the integer and real workspace sizes for the linear solver.

**Return** workspace size

**Parameters**

- lenrwLS: output argument for size of real workspace
- leniwLS: output argument for size of integer workspace

SUNMatrix **getMatrix** () **const**

Get the matrix A (matrix solvers only).

**Return** A

**Protected Functions**

int **initialize** ()

Performs linear solver initialization (assumes that all solver-specific options have been set).

**Return** error code

## Protected Attributes

SUNLinearSolver **solver\_** = {nullptr}  
 Wrapped solver

## Class SUNMatrixWrapper

- Defined in file\_include\_amici\_sundials\_matrix\_wrapper.h

## Class Documentation

**class** amici::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

**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.

**Return****Parameters**

- other:

*SUNMatrixWrapper* &**operator=** (*SUNMatrixWrapper &&other*)  
Move assignment.

**Return****Parameters**

- other:

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.

**Return** raw SunMatrix object

**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.

sunindextype **rows () const**

Get the number of rows.

**Return** number of rows

sunindextype **columns () const**

Get the number of columns.

**Return** 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.

**Return** number of nonzero entries

sunindextype **num\_indexptrs () const**

Get the number of `indexptrs` that can be specified (sparse matrices only)

**Return** number of `indexptrs`

sunindextype **capacity () const**

Get the number of allocated data elements.

**Return** number of allocated entries

*realtype* **\*data ()**

Get raw data of a sparse matrix.

**Return** pointer to first data entry

**const** *realtype* **\*data () const**

Get const raw data of a sparse matrix.

**Return** pointer to first data entry

*realtype* **get\_data (sunindextype idx) const**

Get data of a sparse matrix.

**Return** `idx`-th data entry

**Parameters**

- `idx`: data index

*realtype* **get\_data** (sunindextype *irow*, sunindextype *icol*) **const**  
Get data entry for a dense matrix.

**Return** A(*irow*,*icol*)

**Parameters**

- *irow*: row
- *icol*: col

void **set\_data** (sunindextype *idx*, *realtype* *data*)  
Set data entry for a sparse matrix.

**Parameters**

- *idx*: data index
- *data*: data for *idx*-th entry

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

sunindextype **get\_indexval** (sunindextype *idx*) **const**  
Get the index value of a sparse matrix.

**Return** row (CSC) or column (CSR) for *idx*-th data entry

**Parameters**

- *idx*: data index

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

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

sunindextype **get\_indexptr** (sunindextype *ptr\_idx*) **const**  
Get the index pointer of a sparse matrix.

**Return** index where the *ptr\_idx*-th column (CSC) or row (CSR) starts

**Parameters**

- `ptr_idx`: pointer index

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

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.

**Return** 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

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  $A = \text{sum}(\text{mats}(0) \dots \text{mats}(\text{len}(\text{mats})))$

**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  $\text{beta}=B(k,j)$ ,  $x=C(:,j)$ ,  $C=C$ ,  $w=\text{sparsity of } C(:,j)$  from  $B(k,0 \dots j-1)$ ,  $\text{nnz}=\text{nnz}(C(:,0 \dots j-1))$
- add the k-th column of the sparse matrix A multiplied by beta to the dense vector x. enabled by passing  $\text{beta}^*$ ,  $x=x$ ,  $C=\text{nullptr}$ ,  $w=\text{nullptr}$ ,  $\text{nnz}^*$

**Return** updated number of nonzeros in C

#### Parameters

- k: column index
- beta: scaling factor
- w: index workspace, ( $w[i]<\text{mark}$ ) indicates non-zerosness 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

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 output vector

void **zero** ()  
Set to 0.0, for sparse matrices also resets indexptr/indexvals.

SUNMatrix\_ID **matrix\_id** () const  
Get matrix id.

**Return** 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** `amici::SUNNonLinSolFixedPoint` : **public** `amici::SUNNonLinSolWrapper`  
SUNDIALS Fixed point non-linear solver to solve  $G(y) = y$ .

### Public Functions

**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  $N_s$  sensitivities the value of `count` is:
  - $N_s+1$  if using a simultaneous corrector approach.
  - $N_s$  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  $G(y) = y$ .

#### Return

#### Parameters

- `SysFn`:

## Class SUNNonLinSolNewton

- Defined in file\_include\_amici\_sundials\_linsol\_wrapper.h

## Inheritance Relationships

### Base Type

- `public amici::SUNNonLinSolWrapper` (*Class SUNNonLinSolWrapper*)

## Class Documentation

**class** `amici::SUNNonLinSolNewton` : **public** `amici::SUNNonLinSolWrapper`  
 SUNDIALS Newton non-linear solver to solve  $F(y) = 0$ .

### Public Functions

**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  $F(y) = 0$ .

#### Return

#### Parameters

- `SysFn`:

## 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** `amici::SUNNonLinSolWrapper`

A RAI wrapper for SUNNonLinearSolver structs which solve the nonlinear system  $F(y) = 0$  or  $G(y) = y$ .

Subclassed by *`amici::SUNNonLinSolFixedPoint`*, *`amici::SUNNonLinSolNewton`*

### Public Functions

**SUNNonLinSolWrapper** (SUNNonlinearSolver *sol*)  
*SUNNonLinSolWrapper* from existing SUNNonlinearSolver.

#### Parameters

- *sol*:

**~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.

#### Return

#### Parameters

- *other*:

*SUNNonLinSolWrapper* &**operator=** (*SUNNonLinSolWrapper* &&*other*) **noexcept**  
Move assignment.

**Return****Parameters**

- other:

SUNNonlinearSolver **get** () **const**

Get the wrapped SUNNonlinearSolver.

**Return** SUNNonlinearSolver

SUNNonlinearSolver\_Type **getType** () **const**

Get type ID of the solver.

**Return**

int **setup** (N\_Vector y, void \*mem)

Setup solver.

**Return****Parameters**

- y: the initial iteration passed to the nonlinear solver.
- mem: the sundials integrator memory structure.

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  $G(y) = y$ .

**Return****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.

int **setSysFn** (SUNNonlinSolSysFn SysFn)

Set function to evaluate the nonlinear residual function  $F(y) = 0$  or the fixed point function  $G(y) = y$ .

**Return****Parameters**

- SysFn:

int **setLSetupFn** (SUNNonlinSolLSetupFn SetupFn)

Set linear solver setup function.

**Return**

### Parameters

- SetupFn:

int **setLSolveFn** (SUNNonlinSolLSolveFn *SolveFn*)  
 Set linear solver solve function.

### Return

### Parameters

- SolveFn:

int **setConvTestFn** (SUNNonlinSolConvTestFn *CTestFn*, void *\*ctest\_data*)  
 Set function to test for convergence.

### Return

### Parameters

- CTestFn:
- ctest\_data:

int **setMaxIters** (int *maxiters*)  
 Set maximum number of non-linear iterations.

### Return

### Parameters

- maxiters:

long int **getNumIters** () **const**  
 getNumIters

### Return

int **getCurIter** () **const**  
 getCurIter

### Return

long int **getNumConvFails** () **const**  
 getNumConvFails

### Return

## 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 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 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 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 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 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 amici::LinearMultistepMethod**  
CVODES/IDAS linear multistep method

*Values:*

**enumerator adams**

**enumerator BDF**

### Enum LinearSolver

- Defined in file\_include\_amici\_defines.h

### Enum Documentation

**enum 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 NewtonDampingFactorMode

- Defined in file\_include\_amici\_defines.h

## Enum Documentation

**enum** 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** amici::NonlinearSolverIteration

CVODES/IDAS Nonlinear Iteration method

*Values:*

**enumerator** functional

**enumerator** fixedpoint  
deprecated

**enumerator** newton

## Enum ParameterScaling

- Defined in file\_include\_amici\_defines.h

## Enum Documentation

**enum** 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** amici::RDataReporting

*Values:*

**enumerator** full

**enumerator** residuals

**enumerator** likelihood

## Enum SecondOrderMode

- Defined in file\_include\_amici\_defines.h

## Enum Documentation

**enum** 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** 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** 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** 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** amici::SteadyStateSensitivityMode  
Sensitivity computation mode in steadyStateProblem

*Values:*

**enumerator** newtonOnly

**enumerator** simulationFSA

## Enum SteadyStateStatus

- Defined in file\_include\_amici\_defines.h

## Enum Documentation

**enum** 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

### 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*)

amici\_dgemm provides an interface to the CBlas matrix matrix multiplication routine 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*)

amici\_dgemm provides an interface to the CBlas matrix vector multiplication routine dgemv. This routines computes  $y = \alpha * A * x + \beta * y$  with A: [MxN] x:[Nx1] y:[Mx1]

#### Parameters

- layout: always needs to be AMICI\_BLAS\_ColMajor.
- 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 of `A` (m or n)
- `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*)  
Returns the current backtrace as `std::string`.

**Return** Backtrace

#### Parameters

- `maxFrames`: Number of frames to include

### 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::checkFieldNames

- Defined in file\_include\_amici\_returndata\_matlab.h

### Function Documentation

void amici::checkFieldNames (const char \*\*fieldNames, const int fieldCount)  
checks whether fieldNames was properly allocated

#### Parameters

- fieldNames: array of field names
- fieldCount: expected number of fields in fieldNames

## Function amici::checkSigmaPositivity(std::vector<realtype> const &sigmaVector, const char \*vectorName)

- Defined in file\_include\_amici\_edata.h

### Function Documentation

void amici::checkSigmaPositivity (std::vector<realtype> const &sigmaVector, const char \*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 sigma, const char \*sigmaName)

- Defined in file\_include\_amici\_edata.h

### Function Documentation

void amici::checkSigmaPositivity (realtype sigma, const char \*sigmaName)  
checks input scalar sigma for not strictly positive value

#### Parameters

- sigma: input to be checked
- sigmaName: name of the input



### Function amici::DDspline

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::DDspline (int *id1*, int *id2*, double *t*, int *num*, ...)

### Function amici::DDspline\_pos

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::DDspline\_pos (int *id1*, int *id2*, double *t*, int *num*, ...)

### 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.

**Return** The deserialized object

#### Parameters

- *buffer*: serialized object
- *size*: length of buffer

### 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.

**Return** The deserialized object

#### Parameters

- *serialized*: serialized object

### Function amici::dirac

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::dirac (double *x*)

### Function amici::Dmax

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::Dmax (int *id*, double *a*, double *b*, double *c*)

### Function amici::Dmin

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::Dmin (int *id*, double *a*, double *b*, double *c*)

### Function amici::Dspline

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::Dspline (int *id*, double *t*, int *num*, ...)

### Function amici::Dspline\_pos

- Defined in file\_include\_amici\_symbolic\_functions.h

## Function Documentation

double amici::Dspline\_pos (int *id*, double *t*, int *num*, ...)

## Function amici::expDataFromMatlabCall

- Defined in file\_include\_amici\_interface\_matlab.h

## Function Documentation

std::unique\_ptr<ExpData> amici::expDataFromMatlabCall (const mxArray \*prhs[], const *Model* &model)  
expDataFromMatlabCall parses the experimental data from the matlab call and writes it to an *ExpData* class object

**Return** edata pointer to experimental data object

### Parameters

- prhs: pointer to the array of input arguments
- model: pointer to the model object, this is necessary to perform dimension checks

## Function amici::generic\_model::getModel

- Defined in file\_include\_amici\_interface\_matlab.h

## Function Documentation

std::unique\_ptr<amici::Model> amici::generic\_model::getModel ()

## Function amici::getNaN

- Defined in file\_include\_amici\_symbolic\_functions.h

## Function Documentation

double amici::getNaN ()

## Function amici::getReturnDataMatlabFromAmiciCall

- Defined in file\_include\_amici\_returndata\_matlab.h

## Function Documentation

`mxArray *amici::getReturnDataMatlabFromAmiciCall (ReturnData const *rdata)`  
 generates matlab mxArray from a *ReturnData* object

**Return** rdatamatlab ReturnDataObject stored as matlab compatible data

### Parameters

- `rdata`: ReturnDataObject

## 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`

**Return** Scaled parameter

### Parameters

- `unscaledParameter`:
- `scaling`: parameter scaling

## 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`

**Return** Unscaled parameter

### Parameters

- `scaledParameter`: scaled parameter
- `scaling`: parameter scaling

**Function amici::hdf5::attributeExists(H5::H5File const &file, const std::string &optionsObject, const std::string &attributeName)**

- 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.

**Return** true if attribute exists, false otherwise

#### Parameters

- file: The HDF5 file object
- optionsObject: Dataset of which attributes should be checked
- attributeName: Name of the attribute of interest

**Function amici::hdf5::attributeExists(H5::H5Object const &object, const std::string &attributeName)**

- 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.

**Return** true if attribute exists, false otherwise

#### Parameters

- object: An HDF5 object
- attributeName: Name of the attribute of interest

**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.

**Return** File object

#### Parameters

- `hdf5filename`: File to open

## 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.

**Return** Data read

#### Parameters

- `file`: HDF5 file object
- `name`: Name of dataset to 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.

**Return** Flattened data (row-major)

##### 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

### 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.

**Return** Flattened data (row-major)

##### 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

## 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 &at-
                                             tributeName)
```

Read scalar native double attribute from HDF5 object.

**Return** Attribute value

#### Parameters

- file: HDF5 file
- optionsObject: Object to read attribute from
- attributeName: Name of attribute to read

## 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.

**Return** Data read

#### Parameters

- file: HDF5 file object
- name: Name of dataset to 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 &option-
                                       sObject, const std::string &attributeName)
```

Read scalar native integer attribute from HDF5 object.

**Return** Attribute value

#### Parameters

- file: HDF5 file

- `optionsObject`: Object to read attribute from
- `attributeName`: Name of attribute to read

### Function `amici::hdf5::locationExists(std::string const &filename, std::string const &location)`

- 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.

**Return** `true` if exists, `false` otherwise

#### Parameters

- `filename`: HDF5 filename
- `location`: Location to test for

### Function `amici::hdf5::locationExists(H5::H5File const &file, std::string const &location)`

- 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.

**Return** `true` if exists, `false` otherwise

#### Parameters

- `file`: HDF5 file object
- `location`: Location to test for

### Function `amici::hdf5::readModelDataFromHDF5(std::string const &hdfFile, Model &model, std::string const &datasetPath)`

- 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 &file, *Model* &model, std::string const &datasetPath)**

- 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 &hdf5Filename, std::string const &hdf5Root, const *Model* &model)

Read AMICI *ExpData* data from HDF5 file.

**Return** *ExpData* created from data in the given location

### 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

**Function** `amici::hdf5::readSolverSettingsFromHDF5(const H5::H5File &file, Solver &solver, std::string const &datasetPath)`

- 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 &hdffile, Solver &solver, std::string const &datasetPath)`

- 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 &rdata, H5::H5File const &file, const std::string &hdf5Location)`

- 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 &rdata, std::string const &hdf5Filename, const std::string &hdf5Location)`**

- 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 &solver, std::string const &hdf5Filename, std::string const &hdf5Location)

- 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 &solver, H5::H5File const &file, std::string const &hdf5Location)

- 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

## Function `amici::heaviside`

- Defined in `file_include_amici_symbolic_functions.h`

## Function Documentation

`double amici::heaviside(double x)`

## Function `amici::initAndAttachArray`

- Defined in `file_include_amici_returndata_matlab.h`

## Function Documentation

`double *amici::initAndAttachArray(mxArray *matlabStruct, const char *fieldName, std::vector<mwSize> dim)`  
initializes the field `fieldName` in `matlabStruct` with dimension `dim`

**Return** pointer to field data

### Parameters

- `matlabStruct`: Pointer to the matlab structure
- `fieldName`: Name of the field to which the tensor will be attached
- `dim`: vector of field dimensions

## Function `amici::initMatlabDiagnosisFields`

- Defined in `file_include_amici_returndata_matlab.h`



## Function Documentation

`mxArray *amici::initMatlabDiagnosisFields (ReturnData const *rdata)`  
allocates and initializes diagnosis mxArray with the corresponding fields

**Return** Diagnosis mxArray

**Parameters**

- `rdata`: ReturnDataObject

## Function `amici::initMatlabReturnFields`

- Defined in `file_include_amici_returndata_matlab.h`

## Function Documentation

`mxArray *amici::initMatlabReturnFields (ReturnData const *rdata)`  
allocates and initializes solution mxArray with the corresponding fields

**Return** Solution mxArray

**Parameters**

- `rdata`: ReturnDataObject

## Function `amici::isInf`

- Defined in `file_include_amici_symbolic_functions.h`

## Function Documentation

`int amici::isInf (double what)`

## Function `amici::isNaN`

- Defined in `file_include_amici_symbolic_functions.h`

## Function Documentation

`int amici::isNaN (double what)`

### Function amici::log

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::log(double *x*)

### Function amici::max

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::max(double *a*, double *b*, double *c*)

### Function amici::min

- Defined in file\_include\_amici\_symbolic\_functions.h

### Function Documentation

double amici::min(double *a*, double *b*, double *c*)

### Function amici::N\_VGetArrayPointerConst

- Defined in file\_include\_amici\_vector.h

### Function Documentation

const *realtype*\* amici::N\_VGetArrayPointerConst(const *N\_Vector* *x*)

### Function amici::operator==(const Model &a, const Model &b)

- Defined in file\_include\_amici\_model.h

## Function Documentation

`bool amici::operator==(const Model &a, const Model &b)`

**Return** Equality

**Parameters**

- a: First model instance
- b: Second model instance

**Function `amici::operator==(const ModelDimensions &a, const ModelDimensions &b)`**

- Defined in `file_include_amici_model.h`

## Function Documentation

`bool amici::operator==(const ModelDimensions &a, const ModelDimensions &b)`

**Function `amici::operator==(const SimulationParameters &a, const SimulationParameters &b)`**

- Defined in `file_include_amici_simulation_parameters.h`

## Function Documentation

`bool amici::operator==(const SimulationParameters &a, const SimulationParameters &b)`

**Function `amici::operator==(const Solver &a, const Solver &b)`**

- Defined in `file_include_amici_solver.h`

## Function Documentation

`bool amici::operator==(const Solver &a, const Solver &b)`

**Return**

**Parameters**

- a:
- b:

## Function amici::pos\_pow

- Defined in file\_include\_amici\_symbolic\_functions.h

## Function Documentation

double amici::pos\_pow (double *base*, double *exponent*)

## Function amici::printErrMsgIdAndTxt

- Defined in file\_include\_amici\_amici.h

## Function Documentation

void amici::printErrMsgIdAndTxt (std::string **const** &*id*, std::string **const** &*message*)  
Prints a specified error message associated with the specified identifier.

### Parameters

- *id*: error identifier
- *message*: error message

## Function amici::printfToString

- Defined in file\_include\_amici\_misc.h

## Function Documentation

std::string amici::printfToString (const char \**fmt*, va\_list *ap*)  
Format printf-style arguments to std::string.

**Return** Formatted String

### Parameters

- *fmt*: Format string
- *ap*: Argument list pointer

## Function amici::printWarnMsgIdAndTxt

- Defined in file\_include\_amici\_amici.h

## Function Documentation

void amici::printWarnMsgIdAndTxt (std::string const &id, std::string const &message)  
Prints a specified warning message associated with the specified identifier.

### Parameters

- id: warning identifier
- message: warning message

## 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.

**Return** Error type as string

### Parameters

- err\_type: error type

## Template Function amici::reorder

- Defined in file\_include\_amici\_returndata\_matlab.h

## Function Documentation

template<typename T>  
std::vector<T> amici::reorder (std::vector<T> const &input, std::vector<int> const &order)  
template function that reorders elements in a std::vector

**Return** Reordered vector

### Parameters

- input: unordered vector
- order: dimension permutation

## Function `amici::runAmiciSimulation`

- Defined in `file_include_amici_amici.h`

### Function Documentation

`std::unique_ptr<ReturnData> amici::runAmiciSimulation (Solver &solver, const ExpData *edata, Model &model, bool rethrow = false)`

Core integration routine. Initializes the solver and runs the forward and backward problem.

**Return** rdata pointer to return data object

#### Parameters

- `solver`: *Solver* instance
- `edata`: pointer to experimental data object
- `model`: model specification object
- `rethrow`: rethrow integration exceptions?

## Function `amici::runAmiciSimulations`

- Defined in `file_include_amici_amici.h`

### Function Documentation

`std::vector<std::unique_ptr<ReturnData>> amici::runAmiciSimulations (Solver &solver, const std::vector<ExpData*> &edatas, Model &model, const bool failfast, int num_threads)`

Same as `runAmiciSimulation`, but for multiple *ExpData* instances. When compiled with OpenMP support, this function runs multi-threaded.

**Return** vector of pointers to return data objects

#### 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

## Function amici::scaleParameters

- Defined in file\_include\_amici\_misc.h

### Function Documentation

void amici::scaleParameters (gsl::span<const *realtype*> *bufferUnscaled*, gsl::span<const *ParameterScaling*> *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 obj to char array.

**Return** The object serialized as char

#### Parameters

- *data*: input object
- *size*: maximum char length

## 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>

**Return** The object serialized as std::vector<char>

#### Parameters

- *data*: input object

## 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.
```

**Return** The object serialized as string

#### Parameters

- data: input object

## Function amici::setModelData

- Defined in file\_include\_amici\_interface\_matlab.h

### Function Documentation

```
void amici::setModelData(const mxArray *prhs[], int nrhs, Model &model)
    setModelData sets data from the matlab call to the model object
```

#### Parameters

- prhs: pointer to the array of input arguments
- nrhs: number of elements in prhs
- model: model to update

## Function amici::setSolverOptions

- Defined in file\_include\_amici\_interface\_matlab.h

### Function Documentation

```
void amici::setSolverOptions(const mxArray *prhs[], int nrhs, Solver &solver)
    setSolverOptions solver options from the matlab call to a solver object
```

#### Parameters

- prhs: pointer to the array of input arguments
- nrhs: number of elements in prhs
- solver: solver to update



### Function amici::setupReturnData

- Defined in file\_include\_amici\_interface\_matlab.h

#### Function Documentation

ReturnDataMatlab \*amici::setupReturnData (mxArray \*plhs[], int nlhs)  
setupReturnData initialises the return data struct

**Return** rdata: return data struct

#### Parameters

- plhs: user input
- nlhs: number of elements in plhs

### Function amici::seval

- Defined in file\_include\_amici\_spline.h

#### Function Documentation

double amici::seval (int n, double u, double x[], double y[], double b[], double c[], double d[])

### Function amici::sign

- Defined in file\_include\_amici\_symbolic\_functions.h

#### Function Documentation

double amici::sign (double x)

### Function amici::sinteg

- Defined in file\_include\_amici\_spline.h

#### Function Documentation

double amici::sinteg (int n, double u, double x[], double y[], double b[], double c[], double d[])

## Template Function amici::slice(std::vector<T> &data, int index, unsigned size)

- 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

**Return** span of the slice

#### Parameters

- data: to be sliced
- index: slice index
- size: slice size

## Template Function amici::slice(const std::vector<T> &data, int index, unsigned size)

- Defined in file\_include\_amici\_misc.h

### Function Documentation

template<class **T**>  
 gsl::span<const *T*> amici::slice (const std::vector<*T*> &data, int *index*, unsigned *size*)  
 creates a constant slice from existing constant data

**Return** span of the slice

#### Parameters

- data: to be sliced
- index: slice index
- size: slice size

## Function amici::spline

- Defined in file\_include\_amici\_symbolic\_functions.h

## Function Documentation

double amici::spline (double *t*, int *num*, ...)

### Function amici::spline\_pos

- Defined in file\_include\_amici\_symbolic\_functions.h

## Function Documentation

double amici::spline\_pos (double *t*, int *num*, ...)

### Function amici::unscaleParameters

- Defined in file\_include\_amici\_misc.h

## Function Documentation

void amici::unscaleParameters (gsl::span<const *realtype*> *bufferScaled*, gsl::span<const *ParameterScaling*> *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::wrapErrorHandlerFn

- Defined in file\_include\_amici\_solver.h

## Function Documentation

void amici::wrapErrorHandlerFn (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::writeMatlabField0

- Defined in file\_include\_amici\_returndata\_matlab.h

#### Function Documentation

```
template<typename T>  
void amici::writeMatlabField0 (mxArray *matlabStruct, const char *fieldName, T fieldData)  
    initialize vector and attach to the field
```

##### Parameters

- matlabStruct: pointer of the field to which the vector will be attached
- fieldName: Name of the field to which the vector will be attached
- fieldData: Data which will be stored in the field

### Template Function amici::writeMatlabField1

- Defined in file\_include\_amici\_returndata\_matlab.h

#### Function Documentation

```
template<typename T>  
void amici::writeMatlabField1 (mxArray *matlabStruct, const char *fieldName, gsl::span<const  
                                T> const &fieldData, const int dim0)  
    initialize vector and attach to the field
```

##### Parameters

- matlabStruct: pointer of the field to which the vector will be attached
- fieldName: Name of the field to which the vector will be attached
- fieldData: Data which will be stored in the field
- dim0: Number of elements in the vector

### Template Function amici::writeMatlabField2

- Defined in file\_include\_amici\_returndata\_matlab.h

## Function Documentation

```
template<typename T>
void amici::writeMatlabField2 (mxArray *matlabStruct, const char *fieldName, std::vector<T>
                             const &fieldData, int dim0, int dim1, std::vector<int> perm)
    initialize matrix, attach to the field and write data
```

### Parameters

- matlabStruct: Pointer to the matlab structure
- fieldName: Name of the field to which the tensor will be attached
- fieldData: Data which will be stored in the field
- dim0: Number of rows in the tensor
- dim1: Number of columns in the tensor
- perm: reordering of dimensions (i.e., transposition)

## Template Function amici::writeMatlabField3

- Defined in file\_include\_amici\_returndata\_matlab.h

## Function Documentation

```
template<typename T>
void amici::writeMatlabField3 (mxArray *matlabStruct, const char *fieldName, std::vector<T>
                             const &fieldData, int dim0, int dim1, int dim2, std::vector<int>
                             perm)
    initialize 3D tensor, attach to the field and write data
```

### Parameters

- matlabStruct: Pointer to the matlab structure
- fieldName: Name of the field to which the tensor will be attached
- fieldData: Data which will be stored in the field
- dim0: number of rows in the tensor
- dim1: number of columns in the tensor
- dim2: number of elements in the third dimension of the tensor
- perm: reordering of dimensions

## Template Function amici::writeMatlabField4

- Defined in file\_include\_amici\_returndata\_matlab.h

### Function Documentation

```
template<typename T>
void amici::writeMatlabField4 (mxArray *matlabStruct, const char *fieldName, std::vector<T>
                             const &fieldData, int dim0, int dim1, int dim2, int dim3,
                             std::vector<int> perm)
    initialize 4D tensor, attach to the field and write data
```

#### Parameters

- matlabStruct: Pointer to the matlab structure
- fieldName: Name of the field to which the tensor will be attached
- fieldData: Data which will be stored in the field
- dim0: number of rows in the tensor
- dim1: number of columns in the tensor
- dim2: number of elements in the third dimension of the tensor
- dim3: number of elements in the fourth dimension of the tensor
- perm: reordering of dimensions

## Template Function amici::writeSlice(const gsl::span<const T> slice, gsl::span<T> buffer)

- Defined in file\_include\_amici\_misc.h

### Function Documentation

```
template<class T>
void amici::writeSlice (const gsl::span<const T> 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(const std::vector<T> &s, std::vector<T> &b)

- Defined in file\_include\_amici\_misc.h

#### Function Documentation

```
template<class T>
void amici::writeSlice (const std::vector<T> &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(const std::vector<T> &s, gsl::span<T> b)

- Defined in file\_include\_amici\_misc.h

#### Function Documentation

```
template<class T>
void amici::writeSlice (const std::vector<T> &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(const AmiVector &s, gsl::span<realtype> b)

- Defined in file\_include\_amici\_misc.h

#### Function Documentation

```
void amici::writeSlice (const AmiVector &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 &ar, amici::Model &m, unsigned int version)`

- 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 &ar, amici::ReturnData &r, unsigned int version)`

- 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 &ar, amici::Solver &s, unsigned int version)`

- 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 &ar, amici::CVodeSolver &s, unsigned int version)`

- 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 &ar, amici::IDASolver &s, unsigned int version)`

- 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 m)`

- Defined in file\_include\_amici\_sundials\_matrix\_wrapper.h

## Function Documentation

```
span<realtype> gsl::make_span (SUNMatrix m)
    Create span from SUNMatrix.
```

**Return** Created span

**Parameters**

- *m*: SUNMatrix

## Function `gsl::make_span(N_Vector nv)`

- Defined in `file_include_amici_vector.h`

## Function Documentation

`span<realtype> gsl::make_span(N_Vector nv)`  
Create span from `N_Vector`.

### Return

### Parameters

- `nv`:

## 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\_ILL\_INPUT**

- Defined in file\_include\_amici\_defines.h

**Variable Documentation**

```
constexpr int amici::AMICI_ILL_INPUT = -22
```

**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\_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::defaultContext

- Defined in file\_include\_amici\_model.h

### Variable Documentation

*AmiciApplication* amici::defaultContext

### 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::outputFunctionType

- Defined in file\_include\_amici\_defines.h

### Typedef Documentation

```
using amici::outputFunctionType = std::function<void (std::string const &identifier, std::string  
                                                    const &message) >
```

Type for function to process warnings or error messages.

### 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)
```



## 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](https://doi.org/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

$$J = 1/2 * \text{sum}(((y_i(t) - my_{ti})/\sigma_{y_i})^2 + \log(2 * \pi * \sigma_{y_i}^2))$$

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 import 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 occurred 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

#### 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 initialised 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

### Parameters

- 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.

#### Parameters

- `trigger`: trigger function, the event will be triggered on at all roots of this function
- `bolus`: the bolus that will be added to all states on every occurrence of the event
- `z`: the event output that will be reported on every occurrence of the event

**`mlhsInnerSubst<::amievent> amievent::setHflag(matlabtypesubstitute hflag)`**  
setHflag sets the hflag property.

#### Parameters

- `hflag`: value for the hflag property, type double

#### Return Value

- `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 occurrence

**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.

#### Parameters

- `funstr`: name of the requested function
- `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

#### Parameters

- `model`: model definition object
- `fid`: file id in which the final expression is written

#### Return Value

- `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

#### Parameters

- `model`: model definition object
- `fid`: file id in which the final expression is written

#### Return Value

- `fid`: void

**noret::substitute amifun::writeMcode(:: amimodel model)**  
`writeMcode` generates matlab evaluable code for specific model functions

#### Parameters

- `model`: model definition object

**Return Value**

- `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

**Parameters**

- `model`: model definition object
- `fid`: file id in which the expression should be written

**Return Value**

- `this`: function definition object

**`mlhsInnerSubst<::amifun > amifun::getDeps(:: amimodel model)`**  
getDeps populates the sensiflag for the requested function

**Parameters**

- `model`: model definition object

**Return Value**

- `this`: updated function definition object

**`mlhsInnerSubst<::amifun > amifun::getArgs(:: amimodel model)`**  
getArgs 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.

**Parameters**

- `model`: model definition object

**Return Value**

- `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

**Return Value**

- `this`: updated function definition object

**`mlhsInnerSubst<::amifun > amifun::getCVar()`**  
getCVar populates the cvar property

**Return Value**

- `this`: updated function definition object

**`mlhsInnerSubst<::amifun > amifun::getSensiFlag()`**  
getSensiFlag populates the sensiflag property

**Return Value**

- `this`: updated function definition object

```
mlhsSubst< mlhsInnerSubst<::amifun >,mlhsInnerSubst<::amimodel > > amifun::getSyms (::
```

getSyms computes the symbolic expression for the requested function

#### Parameters

- `model`: model definition object

#### Return Value

- `this`: updated function definition object
- `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

##### Parameters

- symfun: this is the string to the function which generates the modelstruct. You can also directly pass the struct here
- 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)

##### Parameters

- xdot: new right hand side of the differential equation

##### Return Value

- xdot: void

#### **noret::substitute amimodel::updateModelName (matlabtypesubstitute modelname)**

updateModelName updates the modelname

##### Parameters

- modelname: new modelname

##### Return Value

- modelname: void

#### **noret::substitute amimodel::updateWrapPath (matlabtypesubstitute wrap\_path)**

updateModelName updates the modelname

##### Parameters

- wrap\_path: new wrap\_path

##### Return Value

- `wrap_path`: void

**`noret::substitute amimodel::parseModel()`**

`parseModel` parses the model definition and computes all necessary symbolic expressions.

#### Return Value

- void:

**`noret::substitute amimodel::generateC()`**

`generateC` generates the c files which will be used in the compilation.

#### Return Value

- void:

**`noret::substitute amimodel::generateRebuildM()`**

`generateRebuildM` generates a Matlab script for recompilation of this model

#### Return Value

- void:

**`noret::substitute amimodel::compileC()`**

`compileC` compiles the mex simulation file

#### Return Value

- void:

**`noret::substitute amimodel::generateM(:: amimodel amimodelo2)`**

`generateM` generates the matlab wrapper for the compiled C files.

#### Parameters

- `amimodelo2`: this struct must contain all necessary symbolic definitions for second order sensitivities

#### Return Value

- `amimodelo2`: void

**`noret::substitute amimodel::getFun(::struct HTable, ::string funstr)`**

`getFun` generates symbolic expressions for the requested function.

#### Parameters

- `HTable`: struct with hashes of symbolic definition from the previous compilation
- `funstr`: function for which symbolic expressions should be computed

#### Return Value

- `funstr`: void

**`noret::substitute amimodel::makeEvents()`**

`makeEvents` extracts discontinuities from the model right hand side and converts them into events

#### Return Value

- void:

**`noret::substitute amimodel::makeSyms()`**

`makeSyms` extracts symbolic definition from the user provided model and checks them for consistency

**Return Value**

- `void`:

**`mlhsInnerSubst<::bool> amimodel::checkDeps (::struct HTable, ::cell deps)`**  
`checkDeps` checks the dependencies of functions and populates `sym` fields if necessary

**Parameters**

- `HTable`: struct with reference hashes of functions in its fields
- `deps`: cell array with containing a list of dependencies

**Return Value**

- `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.

**Return Value**

- `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 approach later on.

**Return Value**

- `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 approach later on.

**Return Value**

- `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 (cvides/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 mfun = 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 occurring 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()*.

### Parameters

- **modelname**: name of the model as specified for *amiwrap()*
- **modelSourceFolder**: path to model source directory
- **coptim**: optimization flags
- **debug**: enable debugging
- **funs**: array with names of the model functions, will be guessed from source files if left empty
- **cfun**: struct indicating which files should be recompiled

### Return Value

- **cfun**: void

**noret::substitute amimodel::generateMatlabWrapper(matlabtypesubstitute nx, matlabtypesubstitute ny, matlabtypesubstitute np, matlabtypesubstitute nk, matlabtypesubstitute nz, matlabtypesubstitute o2flag, matlabtypesubstitute amimodelo2, matlabtypesubstitute wrapperFilename, matlabtypesubstitute modelname)**  
generateMatlabWrapper generates the matlab wrapper for the compiled C files.

### Parameters

- **nx**: number of states
- **ny**: number of observables
- **np**: number of parameters
- **nk**: number of fixed parameters
- **nz**: number of events
- **o2flag**: o2flag
- **amimodelo2**: this struct must contain all necessary symbolic definitions for second order sensitivities
- **wrapperFilename**: output filename
- **modelname**: name of the model

- `pscale`: default parameter scaling
- `forward`: has forward sensitivity equations
- `adjoint`: has adjoint sensitivity equations

#### Return Value

- `adjoint`: void

### Class `amioption`

- Defined in file `_matlab_@amioption_amioption.m`

### Inheritance Relationships

#### Base Type

- `public CustomDisplay`

### Class Documentation

#### **`amioption` : `public 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`

#### Parameters

- `varargin`: input to construct `amioption` object, see function description

#### Public Members

**`matlabtypesubstitute atol`**     **= 1e-16**  
absolute integration tolerance

**Default:** 1e-16

**`matlabtypesubstitute rtol`**     **= 1e-8**  
relative integration tolerance

**Default:** 1e-8

**matlabtypesubstitute maxsteps** = 1e4  
maximum number of integration steps  
**Default:** 1e4

**matlabtypesubstitute quad\_atol** = 1e-12  
absolute quadrature tolerance  
**Default:** 1e-12

**matlabtypesubstitute quad\_rtol** = 1e-8  
relative quadrature tolerance  
**Default:** 1e-8

**matlabtypesubstitute maxstepsB** = 0  
maximum number of integration steps  
**Default:** 0

**matlabtypesubstitute ss\_atol** = 1e-16  
absolute steady state tolerance  
**Default:** 1e-16

**matlabtypesubstitute ss\_rtol** = 1e-8  
relative steady state tolerance  
**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 newton\_maxlinsteps** = 100  
newton solver: maximum linear steps

**Default:** 100

**Note** This property has custom functionality when its value is changed.

**matlabtypesubstitute newton\_preeq** = **false**  
preequilibration of system via newton solver

**Default:** false

**Note** This property has custom functionality when its value is changed.

**matlabtypesubstitute z2event** = **double.empty("")**  
mapping of event outputs 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 computing sensitivities ({0: Newton}, 1: Simulation)

**Default:** 0

## Public Static Functions

**mlhsInnerSubst< matlabtypesubstitute > amioption::getIntegerPScale(matlabtypesubstitute)**  
pscaleInt converts a parameter scaling string into the corresponding integer representation

### Parameters

- pscaleString: parameter scaling string

### Return Value

- 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

#### Parameters

- 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

#### Parameters

- `symbol`: symbolic object

`mlhsInnerSubst<:sym> optsym::getoptimized()`  
getoptimized calls `symobj::optimize` on the optsym object

#### Return Value

- `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

#### Parameters

- a: first input parameter
- b: second input parameter

#### Return Value

- 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

#### Parameters

- varargin: elements for chain of equalities

#### Return Value

- 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}, ...)`

#### Parameters

- varargin: chain of input parameters

### Return Value

- fun:  $a \geq b$  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  $\text{and}(\text{varargin}\{1\} > \text{varargin}\{2\}, \text{varargin}\{2\} > \text{varargin}\{3\}, \dots)$

### Parameters

- varargin: chain of input parameters

### Return Value

- 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

### Parameters

- condition: logical value
- truepart: value if condition is true
- falsepart: value if condition is false

### Return Value

- 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},...)

### Parameters

- varargin: chain of input parameters

### Return Value

- 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},...)

### Parameters

- varargin: chain of input parameters

### Return Value

- 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

### Parameters

- a: first input parameter
- b: second input parameter

### Return Value

- 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

### Parameters

- a: first input parameter
- b: second input parameter

### Return Value

- 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

### Parameters

- a: first input parameter
- b: second input parameter

### Return Value

- 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

### Parameters

- piece: value if condition is true
- condition: logical value
- default: value if condition is false

**Return Value**

- `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, mat  
am_stepfun is the amici implementation of the step function
```

**Parameters**

- `t`: input variable
- `tstart`: input variable value at which the step starts
- `vstart`: value during the step
- `tend`: input variable value at which the step end
- `vend`: value after the step

**Return Value**

- `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

### Parameters

- `a`: first input parameter
- `b`: second input parameter

### Return Value

- `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.

### Parameters

- `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

**Return Value**

- 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.

**Parameters**

- filename: name of the SBML file (withouth extension)
- modelname: name of the model, this will define the name of the output file (default: input filename)

**Return Value**

- modelname: void



## 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-doxxygen.sh` to check completeness of your documentation
- Make sure your code is compatible with C++11, `gcc` and `clang` (our CI pipeline will do this for you)

- When adding new functionality, please also provide test cases (see `tests/cpputest/` and [documentation/CI.md](#))
- Write meaningful commit messages
- Run all tests to ensure nothing was broken ([more details](#))
  - Run `scripts/buildAll.sh && scripts/run-cpputest.sh`.
  - If you made changes to the Matlab or C++ code and have a Matlab license, please also run `tests/cpputest/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 3.7
- 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 doxypy 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-cpputest.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/cpptest/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/cpptest/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/cpptest/expectedResults.h5` by `tests/cpptest/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/cpptest/expectedResults.h5 tests/cpptest/
↪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).

## 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](https://doi.org/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  $Heaviside(\dots)$  and one to the value of  $1-Heaviside(\dots)$ . To ensure that Heaviside functions are correctly evaluated at the beginning of the simulation, Heaviside functions are implemented 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 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.

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### a

- `amici`, [125](#)
- `amici.amici`, [128](#)
- `amici.gradient_check`, [246](#)
- `amici.import_utils`, [211](#)
- `amici.logging`, [245](#)
- `amici.ode_export`, [214](#)
- `amici.pandas`, [243](#)
- `amici.parameter_mapping`, [249](#)
- `amici.petab_import`, [197](#)
- `amici.petab_import_pysb`, [201](#)
- `amici.petab_objective`, [207](#)
- `amici.plotting`, [242](#)
- `amici.pysb_import`, [195](#)
- `amici.sbml_import`, [189](#)





## Symbols

\_\_USE\_MATH\_DEFINES (C macro), 513  
 \_\_init\_\_() (amici.add\_path method), 126  
 \_\_init\_\_() (amici.amici.ExpData method), 134  
 \_\_init\_\_() (amici.amici.Model method), 148  
 \_\_init\_\_() (amici.amici.ModelDimensions method), 161  
 \_\_init\_\_() (amici.amici.ReturnData method), 168  
 \_\_init\_\_() (amici.amici.SimulationParameters method), 173  
 \_\_init\_\_() (amici.amici.Solver method), 177  
 \_\_init\_\_() (amici.ode\_export.ConservationLaw method), 215  
 \_\_init\_\_() (amici.ode\_export.Constant method), 216  
 \_\_init\_\_() (amici.ode\_export.Event method), 217  
 \_\_init\_\_() (amici.ode\_export.Expression method), 219  
 \_\_init\_\_() (amici.ode\_export.LogLikelihood method), 220  
 \_\_init\_\_() (amici.ode\_export.ModelQuantity method), 221  
 \_\_init\_\_() (amici.ode\_export.ODEExporter method), 223  
 \_\_init\_\_() (amici.ode\_export.ODEModel method), 227  
 \_\_init\_\_() (amici.ode\_export.Observable method), 232  
 \_\_init\_\_() (amici.ode\_export.Parameter method), 233  
 \_\_init\_\_() (amici.ode\_export.SigmaY method), 234  
 \_\_init\_\_() (amici.ode\_export.State method), 235  
 \_\_init\_\_() (amici.ode\_export.TemplateAmici method), 237  
 \_\_init\_\_() (amici.parameter\_mapping.ParameterMapping method), 249  
 \_\_init\_\_() (amici.parameter\_mapping.ParameterMappingForConditioning method), 251  
 \_\_init\_\_() (amici.petab\_import\_pysb.PysbPetabProblem method), 203  
 \_\_init\_\_() (amici.sbml\_import.SbmlImporter method), 191

## A

add\_component() (amici.ode\_export.ODEModel method), 227  
 add\_conservation\_law() (amici.ode\_export.ODEModel method), 227  
 add\_d\_dt() (amici.sbml\_import.SbmlImporter method), 191  
 add\_local\_symbol() (amici.sbml\_import.SbmlImporter method), 191  
 add\_path (class in amici), 126  
 aggregate\_sllh() (in module amici.petab\_objective), 207  
 amici  
   module, 125  
 amici.amici  
   module, 128  
 amici.gradient\_check  
   module, 246  
 amici.import\_utils  
   module, 211  
 amici.logging  
   module, 245  
 amici.ode\_export  
   module, 214  
 amici.pandas  
   module, 243  
 amici.parameter\_mapping  
   module, 249  
 amici.petab\_import  
   module, 197  
 amici.petab\_import\_pysb  
   module, 201  
 amici.petab\_objective  
   module, 207  
 amici.petab\_objective\_for\_conditioning  
   module, 242  
 amici.pysb\_import  
   module, 195  
 amici.sbml\_import  
   module, 189  
 amici::AbstractModel (C++ class), 273

`amici::AbstractModel::~~AbstractModel`  
(C++ function), 273

`amici::AbstractModel::fdeltaqB` (C++ function), 283

`amici::AbstractModel::fdeltasx` (C++ function), 282

`amici::AbstractModel::fdeltax` (C++ function), 282

`amici::AbstractModel::fdeltaxB` (C++ function), 283

`amici::AbstractModel::fdJrzdsigma` (C++ function), 287

`amici::AbstractModel::fdJrzdz` (C++ function), 287

`amici::AbstractModel::fdJydsigma` (C++ function), 286

`amici::AbstractModel::fdJydy` (C++ function), 285

`amici::AbstractModel::fdJydy_colptrs`  
(C++ function), 286

`amici::AbstractModel::fdJydy_rowvals`  
(C++ function), 286

`amici::AbstractModel::fdJzdsigma` (C++ function), 287

`amici::AbstractModel::fdJzdz` (C++ function), 286

`amici::AbstractModel::fdrzdp` (C++ function), 281

`amici::AbstractModel::fdrzdx` (C++ function), 282

`amici::AbstractModel::fdsigmaydp` (C++ function), 284

`amici::AbstractModel::fdsigmazdp` (C++ function), 284

`amici::AbstractModel::fdwdp` (C++ function), 288

`amici::AbstractModel::fdwdp_colptrs`  
(C++ function), 288

`amici::AbstractModel::fdwdp_rowvals`  
(C++ function), 288

`amici::AbstractModel::fdwdw` (C++ function), 289

`amici::AbstractModel::fdwdw_colptrs`  
(C++ function), 290

`amici::AbstractModel::fdwdw_rowvals`  
(C++ function), 290

`amici::AbstractModel::fdwdx` (C++ function), 289

`amici::AbstractModel::fdwdx_colptrs`  
(C++ function), 289

`amici::AbstractModel::fdwdx_rowvals`  
(C++ function), 289

`amici::AbstractModel::fdx0` (C++ function), 278

`amici::AbstractModel::fdxdotdp` (C++ function), 276

`amici::AbstractModel::fdydp` (C++ function), 279

`amici::AbstractModel::fdydx` (C++ function), 279

`amici::AbstractModel::fdzdp` (C++ function), 281

`amici::AbstractModel::fdzdx` (C++ function), 281

`amici::AbstractModel::fJ` (C++ function), 274

`amici::AbstractModel::fJB` (C++ function), 275

`amici::AbstractModel::fJDiag` (C++ function), 276

`amici::AbstractModel::fJrz` (C++ function), 285

`amici::AbstractModel::fJSparse` (C++ function), 275

`amici::AbstractModel::fJSparseB` (C++ function), 275

`amici::AbstractModel::fJSparseB_ss` (C++ function), 274

`amici::AbstractModel::fJv` (C++ function), 276

`amici::AbstractModel::fJy` (C++ function), 285

`amici::AbstractModel::fJz` (C++ function), 285

`amici::AbstractModel::froot` (C++ function), 273

`amici::AbstractModel::frz` (C++ function), 280

`amici::AbstractModel::fsigmay` (C++ function), 284

`amici::AbstractModel::fsigmaz` (C++ function), 284

`amici::AbstractModel::fsrz` (C++ function), 280

`amici::AbstractModel::fstau` (C++ function), 278

`amici::AbstractModel::fsx0` (C++ function), 278

`amici::AbstractModel::fsx0_fixedParameters`  
(C++ function), 277

`amici::AbstractModel::fsxdot` (C++ function), 273

`amici::AbstractModel::fsz` (C++ function), 280

`amici::AbstractModel::fw` (C++ function), 287

`amici::AbstractModel::fx0` (C++ function), 277

`amici::AbstractModel::fx0_fixedParameters`  
(C++ function), 277

amici::AbstractModel::fxBdot\_ss (C++ function), 274  
 amici::AbstractModel::fxdot (C++ function), 273  
 amici::AbstractModel::fy (C++ function), 278  
 amici::AbstractModel::fz (C++ function), 279  
 amici::AbstractModel::getAmiciCommit (C++ function), 277  
 amici::AbstractModel::getAmiciVersion (C++ function), 276  
 amici::AbstractModel::getSolver (C++ function), 273  
 amici::AbstractModel::isFixedParameterState (C++ function), 277  
 amici::AbstractModel::writeSteadystateJBamici (C++ function), 274  
 amici::AMICI\_CONV\_FAILURE (C++ member), 508  
 amici::AMICI\_DAMPING\_FACTOR\_ERROR (C++ member), 508  
 amici::AMICI\_DATA\_RETURN (C++ member), 508  
 amici::amici\_daxpy (C++ function), 471  
 amici::amici\_dgemm (C++ function), 472  
 amici::amici\_dgemv (C++ function), 472  
 amici::AMICI\_ERR\_FAILURE (C++ member), 509  
 amici::AMICI\_ERROR (C++ member), 509  
 amici::AMICI\_ILL\_INPUT (C++ member), 509  
 amici::AMICI\_NO\_STEADY\_STATE (C++ member), 509  
 amici::AMICI\_NORMAL (C++ member), 510  
 amici::AMICI\_NOT\_IMPLEMENTED (C++ member), 510  
 amici::AMICI\_ONE\_STEP (C++ member), 510  
 amici::AMICI\_ONEOUTPUT (C++ member), 510  
 amici::AMICI\_PREEQUILIBRATE (C++ member), 510  
 amici::AMICI\_RECOVERABLE\_ERROR (C++ member), 511  
 amici::AMICI\_ROOT\_RETURN (C++ member), 511  
 amici::AMICI\_SINGULAR\_JACOBIAN (C++ member), 511  
 amici::AMICI\_SUCCESS (C++ member), 511  
 amici::AMICI\_TOO\_MUCH\_ACC (C++ member), 512  
 amici::AMICI\_TOO\_MUCH\_WORK (C++ member), 512  
 amici::AMICI\_UNRECOVERABLE\_ERROR (C++ member), 512  
 amici::AmiciApplication (C++ class), 290  
 amici::AmiciApplication::AmiciApplication (C++ function), 290  
 amici::AmiciApplication::checkFinite (C++ function), 291  
 amici::AmiciApplication::error (C++ member), 292  
 amici::AmiciApplication::errorF (C++ function), 291  
 amici::AmiciApplication::runAmiciSimulation (C++ function), 290  
 amici::AmiciApplication::runAmiciSimulations (C++ function), 291  
 amici::AmiciApplication::warning (C++ member), 292  
 amici::AmiciApplication::warningF (C++ function), 291  
 amici::AmiException (C++ class), 292  
 amici::AmiException::AmiException (C++ function), 292  
 amici::AmiException::getBacktrace (C++ function), 293  
 amici::AmiException::storeBacktrace (C++ function), 293  
 amici::AmiException::what (C++ function), 292  
 amici::AmiVector (C++ class), 293  
 amici::AmiVector::~~AmiVector (C++ function), 294  
 amici::AmiVector::AmiVector (C++ function), 293, 294  
 amici::AmiVector::at (C++ function), 295  
 amici::AmiVector::copy (C++ function), 295  
 amici::AmiVector::data (C++ function), 294  
 amici::AmiVector::getLength (C++ function), 294  
 amici::AmiVector::getNVector (C++ function), 294  
 amici::AmiVector::getVector (C++ function), 294  
 amici::AmiVector::minus (C++ function), 295  
 amici::AmiVector::operator= (C++ function), 294  
 amici::AmiVector::operator[] (C++ function), 295  
 amici::AmiVector::set (C++ function), 295  
 amici::AmiVector::zero (C++ function), 295  
 amici::AmiVectorArray (C++ class), 296  
 amici::AmiVectorArray::~~AmiVectorArray (C++ function), 296  
 amici::AmiVectorArray::AmiVectorArray (C++ function), 296  
 amici::AmiVectorArray::at (C++ function), 296, 297  
 amici::AmiVectorArray::copy (C++ function), 298  
 amici::AmiVectorArray::data (C++ function), 296  
 amici::AmiVectorArray::flatten\_to\_vector (C++ function), 298

`amici::AmiVectorArray::getLength` (C++ function), 298

`amici::AmiVectorArray::getNVector` (C++ function), 297

`amici::AmiVectorArray::getNVectorArray` (C++ function), 297

`amici::AmiVectorArray::operator=` (C++ function), 296

`amici::AmiVectorArray::operator[]` (C++ function), 297

`amici::AmiVectorArray::zero` (C++ function), 298

`amici::backtraceString` (C++ function), 473

`amici::BackwardProblem` (C++ class), 298

`amici::BackwardProblem::BackwardProblem` (C++ function), 298

`amici::BackwardProblem::getAdjointQuadrature` (C++ function), 299

`amici::BackwardProblem::getAdjointState` (C++ function), 299

`amici::BackwardProblem::getdJydx` (C++ function), 299

`amici::BackwardProblem::gett` (C++ function), 298

`amici::BackwardProblem::getwhich` (C++ function), 299

`amici::BackwardProblem::getwhichptr` (C++ function), 299

`amici::BackwardProblem::workBackwardProblem` (C++ function), 298

`amici::BLASLayout` (C++ enum), 465

`amici::BLASLayout::colMajor` (C++ enumerator), 465

`amici::BLASLayout::rowMajor` (C++ enumerator), 465

`amici::BLASTranspose` (C++ enum), 465

`amici::BLASTranspose::conjTrans` (C++ enumerator), 465

`amici::BLASTranspose::noTrans` (C++ enumerator), 465

`amici::BLASTranspose::trans` (C++ enumerator), 465

`amici::checkBufferSize` (C++ function), 473

`amici::checkFieldNames` (C++ function), 474

`amici::checkSigmaPositivity` (C++ function), 474

`amici::ConditionContext` (C++ class), 299

`amici::ConditionContext::~~ConditionContext` (C++ function), 300

`amici::ConditionContext::applyCondition` (C++ function), 300

`amici::ConditionContext::ConditionContext` (C++ function), 300

`amici::ConditionContext::operator=` (C++ function), 300

`amici::function`, 300

`amici::ConditionContext::restore` (C++ function), 300

`amici::const_N_Vector` (C++ type), 516

`amici::ContextManager` (C++ class), 301

`amici::ContextManager::ContextManager` (C++ function), 301

`amici::CvodeException` (C++ class), 301

`amici::CvodeException::CvodeException` (C++ function), 301

`amici::CvodeSolver` (C++ class), 302

`amici::CvodeSolver::~~CvodeSolver` (C++ function), 302

`amici::CvodeSolver::adjInit` (C++ function), 308

`amici::CvodeSolver::allocateSolver` (C++ function), 306

`amici::CvodeSolver::allocateSolverB` (C++ function), 308

`amici::CvodeSolver::binit` (C++ function), 311

`amici::CvodeSolver::boost::serialization::serialize` (C++ function), 312

`amici::CvodeSolver::calcIC` (C++ function), 305

`amici::CvodeSolver::calcICB` (C++ function), 305

`amici::CvodeSolver::clone` (C++ function), 302

`amici::CvodeSolver::diag` (C++ function), 309

`amici::CvodeSolver::diagB` (C++ function), 309

`amici::CvodeSolver::getAdjBmem` (C++ function), 310

`amici::CvodeSolver::getB` (C++ function), 305

`amici::CvodeSolver::getDky` (C++ function), 303

`amici::CvodeSolver::getDkyB` (C++ function), 304

`amici::CvodeSolver::getLastOrder` (C++ function), 310

`amici::CvodeSolver::getModel` (C++ function), 304

`amici::CvodeSolver::getNumErrTestFails` (C++ function), 310

`amici::CvodeSolver::getNumNonlinSolvConvFails` (C++ function), 310

`amici::CvodeSolver::getNumRhsEvals` (C++ function), 309

`amici::CvodeSolver::getNumSteps` (C++ function), 309

`amici::CvodeSolver::getQuad` (C++ function), 305

`amici::CvodeSolver::getQuadB` (C++ function), 305

tion), 305  
 amici::CNodeSolver::getQuadDky (C++ function), 305  
 amici::CNodeSolver::getQuadDkyB (C++ function), 303  
 amici::CNodeSolver::getRootInfo (C++ function), 304  
 amici::CNodeSolver::getSens (C++ function), 305  
 amici::CNodeSolver::getSensDky (C++ function), 303  
 amici::CNodeSolver::init (C++ function), 310  
 amici::CNodeSolver::initSteadystate (C++ function), 310  
 amici::CNodeSolver::operator== (C++ function), 312  
 amici::CNodeSolver::qbinit (C++ function), 311  
 amici::CNodeSolver::quadInit (C++ function), 308  
 amici::CNodeSolver::quadReInitB (C++ function), 302  
 amici::CNodeSolver::quadSStolerances (C++ function), 309  
 amici::CNodeSolver::quadSStolerancesB (C++ function), 309  
 amici::CNodeSolver::reInit (C++ function), 302  
 amici::CNodeSolver::reInitB (C++ function), 302  
 amici::CNodeSolver::reInitPostProcess (C++ function), 306  
 amici::CNodeSolver::reInitPostProcessB (C++ function), 306  
 amici::CNodeSolver::reInitPostProcessF (C++ function), 306  
 amici::CNodeSolver::resetState (C++ function), 308  
 amici::CNodeSolver::rootInit (C++ function), 311  
 amici::CNodeSolver::sensInit1 (C++ function), 310  
 amici::CNodeSolver::sensReInit (C++ function), 302  
 amici::CNodeSolver::sensToggleOff (C++ function), 302  
 amici::CNodeSolver::setBandJacFn (C++ function), 311  
 amici::CNodeSolver::setBandJacFnB (C++ function), 312  
 amici::CNodeSolver::setDenseJacFn (C++ function), 311  
 amici::CNodeSolver::setDenseJacFnB (C++ function), 311  
 amici::CNodeSolver::setErrHandlerFn (C++ function), 307  
 amici::CNodeSolver::setId (C++ function), 308  
 amici::CNodeSolver::setJacTimesVecFn (C++ function), 311  
 amici::CNodeSolver::setJacTimesVecFnB (C++ function), 312  
 amici::CNodeSolver::setLinearSolver (C++ function), 304  
 amici::CNodeSolver::setLinearSolverB (C++ function), 304  
 amici::CNodeSolver::setMaxNumSteps (C++ function), 307  
 amici::CNodeSolver::setMaxNumStepsB (C++ function), 309  
 amici::CNodeSolver::setNonLinearSolver (C++ function), 304  
 amici::CNodeSolver::setNonLinearSolverB (C++ function), 304  
 amici::CNodeSolver::setNonLinearSolverSens (C++ function), 304  
 amici::CNodeSolver::setQuadErrCon (C++ function), 307  
 amici::CNodeSolver::setQuadErrConB (C++ function), 306  
 amici::CNodeSolver::setSensErrCon (C++ function), 306  
 amici::CNodeSolver::setSensParams (C++ function), 308  
 amici::CNodeSolver::setSensSStolerances (C++ function), 306  
 amici::CNodeSolver::setSparseJacFn (C++ function), 311  
 amici::CNodeSolver::setSparseJacFn\_ss (C++ function), 312  
 amici::CNodeSolver::setSparseJacFnB (C++ function), 311  
 amici::CNodeSolver::setSStolerances (C++ function), 306  
 amici::CNodeSolver::setSStolerancesB (C++ function), 308  
 amici::CNodeSolver::setStabLimDet (C++ function), 307  
 amici::CNodeSolver::setStabLimDetB (C++ function), 307  
 amici::CNodeSolver::setStopTime (C++ function), 304  
 amici::CNodeSolver::setSuppressAlg (C++ function), 308  
 amici::CNodeSolver::setUserData (C++ function), 307  
 amici::CNodeSolver::setUserDataB (C++ function), 307

`amici::CNodeSolver::solve (C++ function), 303`  
`amici::CNodeSolver::solveB (C++ function), 303`  
`amici::CNodeSolver::solveF (C++ function), 303`  
`amici::CNodeSolver::turnOffRootFinding (C++ function), 304`  
`amici::DDspline (C++ function), 475`  
`amici::DDspline_pos (C++ function), 475`  
`amici::defaultContext (C++ member), 512`  
`amici::deserializeFromChar (C++ function), 475`  
`amici::deserializeFromString (C++ function), 475`  
`amici::dirac (C++ function), 476`  
`amici::Dmax (C++ function), 476`  
`amici::Dmin (C++ function), 476`  
`amici::Dspline (C++ function), 476`  
`amici::Dspline_pos (C++ function), 477`  
`amici::ExpData (C++ class), 313`  
`amici::ExpData::~~ExpData (C++ function), 314`  
`amici::ExpData::applyDataDimension (C++ function), 319`  
`amici::ExpData::applyDimensions (C++ function), 319`  
`amici::ExpData::applyEventDimension (C++ function), 319`  
`amici::ExpData::checkDataDimension (C++ function), 319`  
`amici::ExpData::checkEventsDimension (C++ function), 319`  
`amici::ExpData::ExpData (C++ function), 313, 314`  
`amici::ExpData::getObservedData (C++ function), 316`  
`amici::ExpData::getObservedDataPtr (C++ function), 316`  
`amici::ExpData::getObservedDataStdDev (C++ function), 317`  
`amici::ExpData::getObservedDataStdDevPtr (C++ function), 317`  
`amici::ExpData::getObservedEvents (C++ function), 317`  
`amici::ExpData::getObservedEventsPtr (C++ function), 317`  
`amici::ExpData::getObservedEventsStdDev (C++ function), 318`  
`amici::ExpData::getObservedEventsStdDevPtr (C++ function), 318`  
`amici::ExpData::getTimepoint (C++ function), 315`  
`amici::ExpData::getTimepoints (C++ function), 315`  
`amici::ExpData::isSetObservedData (C++ function), 315`  
`amici::ExpData::isSetObservedDataStdDev (C++ function), 316`  
`amici::ExpData::isSetObservedEvents (C++ function), 317`  
`amici::ExpData::isSetObservedEventsStdDev (C++ function), 318`  
`amici::ExpData::nmaxevent (C++ function), 315`  
`amici::ExpData::nmaxevent_ (C++ member), 319`  
`amici::ExpData::nt (C++ function), 315`  
`amici::ExpData::nytrue (C++ function), 314`  
`amici::ExpData::nytrue_ (C++ member), 319`  
`amici::ExpData::nztrue (C++ function), 315`  
`amici::ExpData::nztrue_ (C++ member), 319`  
`amici::ExpData::observed_data_ (C++ member), 319`  
`amici::ExpData::observed_data_std_dev_ (C++ member), 319`  
`amici::ExpData::observed_events_ (C++ member), 319`  
`amici::ExpData::observed_events_std_dev_ (C++ member), 319`  
`amici::ExpData::setObservedData (C++ function), 315`  
`amici::ExpData::setObservedDataStdDev (C++ function), 316`  
`amici::ExpData::setObservedEvents (C++ function), 317`  
`amici::ExpData::setObservedEventsStdDev (C++ function), 317, 318`  
`amici::ExpData::setTimepoints (C++ function), 315`  
`amici::expDataFromMatlabCall (C++ function), 477`  
`amici::FinalStateStorer (C++ class), 320`  
`amici::FinalStateStorer::~~FinalStateStorer (C++ function), 320`  
`amici::FinalStateStorer::FinalStateStorer (C++ function), 320`  
`amici::FinalStateStorer::operator= (C++ function), 320`  
`amici::FixedParameterContext (C++ enum), 466`  
`amici::FixedParameterContext::preequilibration (C++ enumerator), 466`  
`amici::FixedParameterContext::presimulation (C++ enumerator), 466`  
`amici::FixedParameterContext::simulation (C++ enumerator), 466`  
`amici::ForwardProblem (C++ class), 320`  
`amici::ForwardProblem::~~ForwardProblem`



(C++ function), 321

amici::ForwardProblem::edata (C++ member), 323

amici::ForwardProblem::ForwardProblem (C++ function), 320

amici::ForwardProblem::getAdjointUpdates (C++ function), 321

amici::ForwardProblem::getCurrentTimeIteration (C++ function), 322

amici::ForwardProblem::getDiscontinuities (C++ function), 322

amici::ForwardProblem::getDJydx (C++ function), 322

amici::ForwardProblem::getDJzdx (C++ function), 322

amici::ForwardProblem::getEventCounter (C++ function), 322

amici::ForwardProblem::getFinalSimulationState (C++ function), 323

amici::ForwardProblem::getFinalTime (C++ function), 322

amici::ForwardProblem::getInitialSimulationState (C++ function), 323

amici::ForwardProblem::getNumberOfRoots (C++ function), 321

amici::ForwardProblem::getRHSAtDiscontinuities (C++ function), 321

amici::ForwardProblem::getRHSBeforeDiscontinuity (C++ function), 321

amici::ForwardProblem::getRootCounter (C++ function), 323

amici::ForwardProblem::getRootIndexes (C++ function), 322

amici::ForwardProblem::getSimulationStateEvent (C++ function), 323

amici::ForwardProblem::getSimulationStateTimepoint (C++ function), 323

amici::ForwardProblem::getState (C++ function), 321

amici::ForwardProblem::getStateDerivative (C++ function), 321

amici::ForwardProblem::getStateDerivativePointer (C++ function), 322

amici::ForwardProblem::getStateDerivativeSensitivity (C++ function), 322

amici::ForwardProblem::getStatePointer (C++ function), 322

amici::ForwardProblem::getStatesAtDiscontinuity (C++ function), 321

amici::ForwardProblem::getStateSensitivity (C++ function), 321

amici::ForwardProblem::getStateSensitivityPointer (C++ function), 322

amici::ForwardProblem::getTime (C++ function), 321

amici::ForwardProblem::model (C++ member), 323

amici::ForwardProblem::solver (C++ member), 323

amici::ForwardProblem::workForwardProblem (C++ function), 321

amicon::generic\_model::getModel (C++ function), 477

amici::getNaN (C++ function), 477

amici::getReturnDataMatlabFromAmiciCall (C++ function), 478

amici::getScaledParameter (C++ function), 478

amici::getUnscaledParameter (C++ function), 478

amici::hdf5::attributeExists (C++ function), 479

amici::hdf5::createAndWriteDouble1DDataset (C++ function), 480

amici::hdf5::createAndWriteDouble2DDataset (C++ function), 480

amici::hdf5::createAndWriteDouble3DDataset (C++ function), 480

amici::hdf5::createAndWriteInt1DDataset (C++ function), 481

amici::hdf5::createAndWriteInt2DDataset (C++ function), 481

amici::hdf5::createGroup (C++ function), 482

amici::hdf5::createOrOpenForWriting (C++ function), 482

amici::hdf5::getDoubleDataset1D (C++ function), 482

amici::hdf5::getDoubleDataset2D (C++ function), 483

amici::hdf5::getDoubleDataset3D (C++ function), 483

amici::hdf5::getDoubleScalarAttribute (C++ function), 484

amici::hdf5::getIntDataset1D (C++ function), 484

amici::hdf5::getIntScalarAttribute (C++ function), 484

amici::hdf5::parameterExists (C++ function), 485

amici::hdf5::readModelDataFromHDF5 (C++ function), 486

amici::hdf5::readSimulationExpData (C++ function), 486

amici::hdf5::readSolverSettingsFromHDF5 (C++ function), 487

amici::hdf5::writeReturnData (C++ function), 488

amici::hdf5::writeReturnDataDiagnosis

(C++ function), 488

amici::hdf5::writeSimulationExpData (C++ function), 489

amici::hdf5::writeSolverSettingsToHDF5 (C++ function), 489, 490

amici::heaviside (C++ function), 490

amici::IDAException (C++ class), 324

amici::IDAException::IDAException (C++ function), 324

amici::IDASolver (C++ class), 324

amici::IDASolver::~~IDASolver (C++ function), 325

amici::IDASolver::adjInit (C++ function), 331

amici::IDASolver::allocateSolver (C++ function), 329

amici::IDASolver::allocateSolverB (C++ function), 331

amici::IDASolver::binit (C++ function), 333

amici::IDASolver::calcIC (C++ function), 328

amici::IDASolver::calcICB (C++ function), 328

amici::IDASolver::clone (C++ function), 325

amici::IDASolver::diag (C++ function), 332

amici::IDASolver::diagB (C++ function), 332

amici::IDASolver::getAdjBmem (C++ function), 333

amici::IDASolver::getB (C++ function), 327

amici::IDASolver::getDky (C++ function), 327

amici::IDASolver::getDkyB (C++ function), 327

amici::IDASolver::getLastOrder (C++ function), 333

amici::IDASolver::getModel (C++ function), 328

amici::IDASolver::getNumErrTestFails (C++ function), 332

amici::IDASolver::getNumNonlinSolvConvFails (C++ function), 333

amici::IDASolver::getNumRhsEvals (C++ function), 332

amici::IDASolver::getNumSteps (C++ function), 332

amici::IDASolver::getQuad (C++ function), 328

amici::IDASolver::getQuadB (C++ function), 327

amici::IDASolver::getQuadDky (C++ function), 328

amici::IDASolver::getQuadDkyB (C++ function), 327

amici::IDASolver::getRootInfo (C++ function), 326

amici::IDASolver::getSens (C++ function), 327

amici::IDASolver::getSensDky (C++ function), 327

amici::IDASolver::init (C++ function), 333

amici::IDASolver::initSteadystate (C++ function), 333

amici::IDASolver::qbinit (C++ function), 334

amici::IDASolver::quadInit (C++ function), 331

amici::IDASolver::quadReInitB (C++ function), 325

amici::IDASolver::quadSStolerances (C++ function), 326

amici::IDASolver::quadSStolerancesB (C++ function), 326

amici::IDASolver::reInit (C++ function), 325

amici::IDASolver::reInitB (C++ function), 325

amici::IDASolver::reInitPostProcess (C++ function), 329

amici::IDASolver::reInitPostProcessB (C++ function), 325

amici::IDASolver::reInitPostProcessF (C++ function), 325

amici::IDASolver::resetState (C++ function), 331

amici::IDASolver::rootInit (C++ function), 334

amici::IDASolver::sensInit1 (C++ function), 333

amici::IDASolver::sensReInit (C++ function), 325

amici::IDASolver::sensToggleOff (C++ function), 325

amici::IDASolver::setBandJacFn (C++ function), 334

amici::IDASolver::setBandJacFnB (C++ function), 334

amici::IDASolver::setDenseJacFn (C++ function), 334

amici::IDASolver::setDenseJacFnB (C++ function), 334

amici::IDASolver::setErrHandlerFn (C++ function), 330

amici::IDASolver::setId (C++ function), 331

amici::IDASolver::setJacTimesVecFn (C++ function), 334

amici::IDASolver::setJacTimesVecFnB (C++ function), 335

amici::IDASolver::setLinearSolver (C++ function), 328

amici::IDASolver::setLinearSolverB (C++ function), 328

amici::IDASolver::setMaxNumSteps (C++



*function*), 330  
 amici::IDASolver::setMaxNumStepsB (C++ *function*), 332  
 amici::IDASolver::setNonLinearSolver (C++ *function*), 329  
 amici::IDASolver::setNonLinearSolverB (C++ *function*), 329  
 amici::IDASolver::setNonLinearSolverSens (C++ *function*), 329  
 amici::IDASolver::setQuadErrCon (C++ *function*), 330  
 amici::IDASolver::setQuadErrConB (C++ *function*), 330  
 amici::IDASolver::setSensErrCon (C++ *function*), 329  
 amici::IDASolver::setSensParams (C++ *function*), 331  
 amici::IDASolver::setSensSStolerances (C++ *function*), 329  
 amici::IDASolver::setSparseJacFn (C++ *function*), 334  
 amici::IDASolver::setSparseJacFn\_ss (C++ *function*), 335  
 amici::IDASolver::setSparseJacFnB (C++ *function*), 334  
 amici::IDASolver::setSStolerances (C++ *function*), 329  
 amici::IDASolver::setSStolerancesB (C++ *function*), 332  
 amici::IDASolver::setStabLimDet (C++ *function*), 330  
 amici::IDASolver::setStabLimDetB (C++ *function*), 330  
 amici::IDASolver::setStopTime (C++ *function*), 328  
 amici::IDASolver::setSuppressAlg (C++ *function*), 331  
 amici::IDASolver::setUserData (C++ *function*), 330  
 amici::IDASolver::setUserDataB (C++ *function*), 330  
 amici::IDASolver::solve (C++ *function*), 326  
 amici::IDASolver::solveB (C++ *function*), 326  
 amici::IDASolver::solveF (C++ *function*), 326  
 amici::IDASolver::turnOffRootFinding (C++ *function*), 328  
 amici::initAndAttachArray (C++ *function*), 490  
 amici::initMatlabDiagnosisFields (C++ *function*), 491  
 amici::initMatlabReturnFields (C++ *function*), 491  
 amici::IntegrationFailure (C++ *class*), 335  
 amici::IntegrationFailure::error\_code (C++ *member*), 335  
 amici::IntegrationFailure::IntegrationFailure (C++ *function*), 335  
 amici::IntegrationFailure::time (C++ *member*), 335  
 amici::IntegrationFailureB (C++ *class*), 336  
 amici::IntegrationFailureB::error\_code (C++ *member*), 336  
 amici::IntegrationFailureB::IntegrationFailureB (C++ *function*), 336  
 amici::IntegrationFailureB::time (C++ *member*), 336  
 amici::InternalSensitivityMethod (C++ *enum*), 466  
 amici::InternalSensitivityMethod::simultaneous (C++ *enumerator*), 466  
 amici::InternalSensitivityMethod::staggered (C++ *enumerator*), 466  
 amici::InternalSensitivityMethod::staggered1 (C++ *enumerator*), 466  
 amici::InterpolationType (C++ *enum*), 466  
 amici::InterpolationType::hermite (C++ *enumerator*), 466  
 amici::InterpolationType::polynomial (C++ *enumerator*), 466  
 amici::isInf (C++ *function*), 491  
 amici::isNaN (C++ *function*), 491  
 amici::LinearMultistepMethod (C++ *enum*), 467  
 amici::LinearMultistepMethod::adams (C++ *enumerator*), 467  
 amici::LinearMultistepMethod::BDF (C++ *enumerator*), 467  
 amici::LinearSolver (C++ *enum*), 467  
 amici::LinearSolver::band (C++ *enumerator*), 467  
 amici::LinearSolver::dense (C++ *enumerator*), 467  
 amici::LinearSolver::diag (C++ *enumerator*), 467  
 amici::LinearSolver::KLU (C++ *enumerator*), 467  
 amici::LinearSolver::LAPACKBand (C++ *enumerator*), 467  
 amici::LinearSolver::LAPACKDense (C++ *enumerator*), 467  
 amici::LinearSolver::SPBCG (C++ *enumerator*), 467  
 amici::LinearSolver::SPGMR (C++ *enumerator*), 467  
 amici::LinearSolver::SPTFQMR (C++ *enumerator*), 467  
 amici::LinearSolver::SuperLUMT (C++ *enumerator*), 467

amici::log (C++ function), 492  
 amici::max (C++ function), 492  
 amici::min (C++ function), 492  
 amici::Model (C++ class), 337  
 amici::Model::~~Model (C++ function), 337  
 amici::Model::addAdjointQuadratureEventUpdate (C++ function), 354  
 amici::Model::addAdjointStateEventUpdate (C++ function), 354  
 amici::Model::addEventObjective (C++ function), 351  
 amici::Model::addEventObjectiveRegularization (C++ function), 352  
 amici::Model::addEventObjectiveSensitivity (C++ function), 352  
 amici::Model::addObservableObjective (C++ function), 349  
 amici::Model::addObservableObjectiveSensitivity (C++ function), 349  
 amici::Model::addPartialEventObjectiveSensitivity (C++ function), 352  
 amici::Model::addPartialObservableObjectiveSensitivity (C++ function), 349  
 amici::Model::addStateEventUpdate (C++ function), 353  
 amici::Model::addStateSensitivityEventUpdate (C++ function), 354  
 amici::Model::always\_check\_finite\_ (C++ member), 365  
 amici::Model::any\_state\_non\_negative\_ (C++ member), 365  
 amici::Model::app (C++ member), 357  
 amici::Model::boost::serialization::serialize (C++ function), 365  
 amici::Model::checkFinite (C++ function), 355  
 amici::Model::checkLLHBufferSize (C++ function), 357  
 amici::Model::clone (C++ function), 338  
 amici::Model::computeX\_pos (C++ function), 364  
 amici::Model::derived\_state\_ (C++ member), 365  
 amici::Model::fdJrzdsigma (C++ function), 362  
 amici::Model::fdJrzd (C++ function), 362  
 amici::Model::fdJydp (C++ function), 359  
 amici::Model::fdJydsigma (C++ function), 359  
 amici::Model::fdJydx (C++ function), 359  
 amici::Model::fdJydy (C++ function), 358  
 amici::Model::fdJzdp (C++ function), 361  
 amici::Model::fdJzdsigma (C++ function), 361  
 amici::Model::fdJzdx (C++ function), 362  
 amici::Model::fdJzdz (C++ function), 361  
 amici::Model::fdrzdp (C++ function), 360  
 amici::Model::fdrzdx (C++ function), 360  
 amici::Model::fdsigma ydp (C++ function), 358  
 amici::Model::fdsigmazdp (C++ function), 360  
 amici::Model::fdwdp (C++ function), 363  
 amici::Model::fdwdw (C++ function), 363  
 amici::Model::fdwdx (C++ function), 363  
 amici::Model::fdydp (C++ function), 358  
 amici::Model::fdydx (C++ function), 358  
 amici::Model::fdzdp (C++ function), 359  
 amici::Model::fdzdx (C++ function), 360  
 amici::Model::fJrz (C++ function), 362  
 amici::Model::fJy (C++ function), 358  
 amici::Model::fJz (C++ function), 361  
 amici::Model::frz (C++ function), 360  
 amici::Model::fsdx0 (C++ function), 356  
 amici::Model::fsigma y (C++ function), 358  
 amici::Model::fsigmaz (C++ function), 360  
 amici::Model::fstotal\_cl (C++ function), 364  
 amici::Model::fsx0 (C++ function), 355  
 amici::Model::fsx0\_fixedParameters (C++ function), 356  
 amici::Model::fsx\_rdata (C++ function), 356, 363  
 amici::Model::fsx\_solver (C++ function), 364  
 amici::Model::ftotal\_cl (C++ function), 364  
 amici::Model::fw (C++ function), 363  
 amici::Model::fx0 (C++ function), 355  
 amici::Model::fx0\_fixedParameters (C++ function), 355  
 amici::Model::fx\_rdata (C++ function), 356, 363  
 amici::Model::fx\_solver (C++ function), 364  
 amici::Model::fy (C++ function), 358  
 amici::Model::fz (C++ function), 359  
 amici::Model::get\_dxdotdp (C++ function), 356  
 amici::Model::get\_dxdotdp\_full (C++ function), 356  
 amici::Model::getAdjointStateEventUpdate (C++ function), 353  
 amici::Model::getAdjointStateObservableUpdate (C++ function), 349  
 amici::Model::getAlwaysCheckFinite (C++ function), 355  
 amici::Model::getEvent (C++ function), 350  
 amici::Model::getEventRegularization (C++ function), 350  
 amici::Model::getEventRegularizationSensitivity (C++ function), 350  
 amici::Model::getEventSensitivity (C++ function), 350  
 amici::Model::getEventSigma (C++ function), 351

```

amici::Model::getEventSigmaSensitivity (C++ function), 351
amici::Model::getEventTimeSensitivity (C++ function), 353
amici::Model::getExpression (C++ function), 348
amici::Model::getExpressionIds (C++ function), 345
amici::Model::getExpressionNames (C++ function), 344
amici::Model::getFixedParameterById (C++ function), 342
amici::Model::getFixedParameterByName (C++ function), 342
amici::Model::getFixedParameterIds (C++ function), 344
amici::Model::getFixedParameterNames (C++ function), 343
amici::Model::getFixedParameters (C++ function), 341
amici::Model::getInitialStates (C++ function), 346
amici::Model::getInitialStateSensitivities (C++ function), 347
amici::Model::getModelState (C++ function), 346
amici::Model::getName (C++ function), 343
amici::Model::getObservable (C++ function), 348
amici::Model::getObservableIds (C++ function), 345
amici::Model::getObservableNames (C++ function), 343
amici::Model::getObservableSensitivity (C++ function), 348
amici::Model::getObservableSigma (C++ function), 348
amici::Model::getObservableSigmaSensitivity (C++ function), 348
amici::Model::getParameterById (C++ function), 340
amici::Model::getParameterByName (C++ function), 340
amici::Model::getParameterIds (C++ function), 344
amici::Model::getParameterList (C++ function), 346
amici::Model::getParameterNames (C++ function), 343
amici::Model::getParameters (C++ function), 340
amici::Model::getParameterScale (C++ function), 340
amici::Model::getReinitializationStateIds (C++ function), 356
amici::Model::getReinitializeFixedParameterInitialValues (C++ function), 347
amici::Model::getStateIds (C++ function), 344
amici::Model::getStateIsNonNegative (C++ function), 345
amici::Model::getStateNames (C++ function), 343
amici::Model::getSteadyStateSensitivityMode (C++ function), 347
amici::Model::getTimepoint (C++ function), 345
amici::Model::getTimepoints (C++ function), 345
amici::Model::getUnobservedEventSensitivity (C++ function), 350
amici::Model::getUnscaledParameters (C++ function), 340
amici::Model::hasCustomInitialStates (C++ function), 347
amici::Model::hasCustomInitialStateSensitivities (C++ function), 347
amici::Model::hasExpressionIds (C++ function), 345
amici::Model::hasExpressionNames (C++ function), 344
amici::Model::hasFixedParameterIds (C++ function), 344
amici::Model::hasFixedParameterNames (C++ function), 343
amici::Model::hasObservableIds (C++ function), 344
amici::Model::hasObservableNames (C++ function), 343
amici::Model::hasParameterIds (C++ function), 344
amici::Model::hasParameterNames (C++ function), 343
amici::Model::hasQuadraticLLH (C++ function), 345
amici::Model::hasStateIds (C++ function), 344
amici::Model::hasStateNames (C++ function), 343
amici::Model::idlist (C++ member), 357
amici::Model::initHeaviside (C++ function), 338
amici::Model::initialize (C++ function), 338
amici::Model::initializeB (C++ function), 338
amici::Model::initializeStates (C++ function), 338
amici::Model::initializeStateSensitivities (C++ function), 347

```

(C++ function), 338

amici::Model::initializeVectors (C++ function), 358

amici::Model::k (C++ function), 339

amici::Model::Model (C++ function), 337

amici::Model::ncl (C++ function), 339

amici::Model::nk (C++ function), 339

amici::Model::nMaxEvent (C++ function), 339

amici::Model::nmaxevent\_ (C++ member), 365

amici::Model::np (C++ function), 339

amici::Model::nplist (C++ function), 339

amici::Model::nt (C++ function), 339

amici::Model::nx\_reinit (C++ function), 339

amici::Model::o2mode (C++ member), 357

amici::Model::operator= (C++ function), 337

amici::Model::operator== (C++ function), 365

amici::Model::plist (C++ function), 346

amici::Model::pythonGenerated (C++ member), 357

amici::Model::requireSensitivitiesForAllParameters (C++ function), 348

amici::Model::setAllStatesNonNegative (C++ function), 346

amici::Model::setAlwaysCheckFinite (C++ function), 355

amici::Model::setFixedParameterById (C++ function), 342

amici::Model::setFixedParameterByName (C++ function), 342

amici::Model::setFixedParameters (C++ function), 342

amici::Model::setFixedParametersByIdRegex (C++ function), 342

amici::Model::setFixedParametersByNameRegex (C++ function), 342

amici::Model::setInitialStates (C++ function), 346

amici::Model::setInitialStateSensitivities (C++ function), 347

amici::Model::setModelState (C++ function), 346

amici::Model::setNMaxEvent (C++ function), 339

amici::Model::setParameterById (C++ function), 341

amici::Model::setParameterByName (C++ function), 341

amici::Model::setParameterList (C++ function), 346

amici::Model::setParameters (C++ function), 340

amici::Model::setParametersByIdRegex (C++ function), 341

amici::Model::setParametersByNameRegex (C++ function), 341

amici::Model::setParameterScale (C++ function), 340

amici::Model::setReinitializationStateIdxs (C++ function), 356

amici::Model::setReinitializeFixedParameterInitialStates (C++ function), 347

amici::Model::setStateIsNonNegative (C++ function), 346

amici::Model::setSteadyStateSensitivityMode (C++ function), 347

amici::Model::setT0 (C++ function), 345

amici::Model::setTimepoints (C++ function), 345

amici::Model::setUnscaledInitialStateSensitivities (C++ function), 347

amici::Model::state\_ (C++ member), 365

amici::Model::state\_is\_non\_negative\_ (C++ member), 365

amici::Model::steadystate\_sensitivity\_mode\_ (C++ member), 365

amici::Model::sx0data\_ (C++ member), 365

amici::Model::t0 (C++ function), 345

amici::Model::updateHeaviside (C++ function), 354

amici::Model::updateHeavisideB (C++ function), 355

amici::Model::writeLLHSensitivitySlice (C++ function), 357

amici::Model::writeSensitivitySliceEvent (C++ function), 357

amici::Model::writeSliceEvent (C++ function), 357

amici::Model::x0data\_ (C++ member), 365

amici::Model::z2event\_ (C++ member), 365

amici::Model\_DAE (C++ class), 366

amici::Model\_DAE::fdxdotdp (C++ function), 372, 375

amici::Model\_DAE::fJ (C++ function), 366, 367

amici::Model\_DAE::fJB (C++ function), 367

amici::Model\_DAE::fJDiag (C++ function), 369

amici::Model\_DAE::fJSparse (C++ function), 368, 374

amici::Model\_DAE::fJSparseB (C++ function), 368

amici::Model\_DAE::fJSparseB\_ss (C++ function), 372

amici::Model\_DAE::fJv (C++ function), 369

amici::Model\_DAE::fJvB (C++ function), 369

amici::Model\_DAE::fM (C++ function), 373, 375

amici::Model\_DAE::fqBdot (C++ function), 371

amici::Model\_DAE::fqBdot\_ss (C++ function), 372

amici::Model\_DAE::froot (C++ function), 370,

374  
 amici::Model\_DAE::fsxdot (C++ function), 373  
 amici::Model\_DAE::fxBdot (C++ function), 371  
 amici::Model\_DAE::fxBdot\_ss (C++ function), 371  
 amici::Model\_DAE::fxdot (C++ function), 370, 374  
 amici::Model\_DAE::getSolver (C++ function), 373  
 amici::Model\_DAE::Model\_DAE (C++ function), 366  
 amici::Model\_DAE::writeSteadystateJB (C++ function), 372  
 amici::Model\_ODE (C++ class), 376  
 amici::Model\_ODE::fdxdotdp (C++ function), 384, 386  
 amici::Model\_ODE::fdxdotdp\_explicit (C++ function), 384  
 amici::Model\_ODE::fdxdotdp\_explicit\_colptrs (C++ function), 385  
 amici::Model\_ODE::fdxdotdp\_explicit\_rowvals (C++ function), 385  
 amici::Model\_ODE::fdxdotdw (C++ function), 386  
 amici::Model\_ODE::fdxdotdw\_colptrs (C++ function), 386  
 amici::Model\_ODE::fdxdotdw\_rowvals (C++ function), 386  
 amici::Model\_ODE::fdxdotdx\_explicit (C++ function), 385  
 amici::Model\_ODE::fdxdotdx\_explicit\_colptrs (C++ function), 385  
 amici::Model\_ODE::fdxdotdx\_explicit\_rowvals (C++ function), 385  
 amici::Model\_ODE::fJ (C++ function), 376  
 amici::Model\_ODE::fJB (C++ function), 377  
 amici::Model\_ODE::fJDiag (C++ function), 378, 379  
 amici::Model\_ODE::fJSparse (C++ function), 377, 378, 383  
 amici::Model\_ODE::fJSparse\_colptrs (C++ function), 383  
 amici::Model\_ODE::fJSparse\_rowvals (C++ function), 383  
 amici::Model\_ODE::fJSparseB (C++ function), 378  
 amici::Model\_ODE::fJSparseB\_ss (C++ function), 381  
 amici::Model\_ODE::fJv (C++ function), 379  
 amici::Model\_ODE::fJvB (C++ function), 379  
 amici::Model\_ODE::fqBdot (C++ function), 380  
 amici::Model\_ODE::fqBdot\_ss (C++ function), 381  
 amici::Model\_ODE::froot (C++ function), 380, 383  
 amici::Model\_ODE::fsxdot (C++ function), 382  
 amici::Model\_ODE::fxBdot (C++ function), 380  
 amici::Model\_ODE::fxBdot\_ss (C++ function), 381  
 amici::Model\_ODE::fxdot (C++ function), 380, 384  
 amici::Model\_ODE::getSolver (C++ function), 382  
 amici::Model\_ODE::Model\_ODE (C++ function), 376  
 amici::Model\_ODE::writeSteadystateJB (C++ function), 381  
 amici::ModelContext (C++ class), 387  
 amici::ModelContext::~ModelContext (C++ function), 387  
 amici::ModelContext::ModelContext (C++ function), 387  
 amici::ModelContext::operator= (C++ function), 387  
 amici::ModelContext::restore (C++ function), 387  
 amici::ModelDimensions (C++ struct), 266  
 amici::ModelDimensions::lbw (C++ member), 268  
 amici::ModelDimensions::ModelDimensions (C++ function), 267  
 amici::ModelDimensions::ndJydy (C++ member), 268  
 amici::ModelDimensions::ndwdp (C++ member), 268  
 amici::ModelDimensions::ndwdw (C++ member), 268  
 amici::ModelDimensions::ndwdx (C++ member), 268  
 amici::ModelDimensions::ndxdotdw (C++ member), 268  
 amici::ModelDimensions::ne (C++ member), 268  
 amici::ModelDimensions::nJ (C++ member), 268  
 amici::ModelDimensions::nk (C++ member), 268  
 amici::ModelDimensions::nnz (C++ member), 268  
 amici::ModelDimensions::np (C++ member), 268  
 amici::ModelDimensions::nw (C++ member), 268  
 amici::ModelDimensions::nx\_rdata (C++ member), 268  
 amici::ModelDimensions::nx\_solver (C++ member), 268  
 amici::ModelDimensions::nx\_solver\_reinit



(C++ member), 268

amici::ModelDimensions::nxtrue\_rdata (C++ member), 268

amici::ModelDimensions::nxtrue\_solver (C++ member), 268

amici::ModelDimensions::ny (C++ member), 268

amici::ModelDimensions::nytrue (C++ member), 268

amici::ModelDimensions::nz (C++ member), 268

amici::ModelDimensions::nztrue (C++ member), 268

amici::ModelDimensions::ubw (C++ member), 268

amici::ModelState (C++ struct), 269

amici::ModelState::fixedParameters (C++ member), 269

amici::ModelState::h (C++ member), 269

amici::ModelState::plist (C++ member), 269

amici::ModelState::stotal\_cl (C++ member), 269

amici::ModelState::total\_cl (C++ member), 269

amici::ModelState::unscaledParameters (C++ member), 269

amici::ModelStateDerived (C++ struct), 269

amici::ModelStateDerived::deltaqB\_ (C++ member), 272

amici::ModelStateDerived::deltasx\_ (C++ member), 272

amici::ModelStateDerived::deltax\_ (C++ member), 272

amici::ModelStateDerived::deltaxB\_ (C++ member), 272

amici::ModelStateDerived::dJrzdsigma\_ (C++ member), 271

amici::ModelStateDerived::dJrzdz\_ (C++ member), 271

amici::ModelStateDerived::dJydp\_ (C++ member), 271

amici::ModelStateDerived::dJydsigma\_ (C++ member), 271

amici::ModelStateDerived::dJydx\_ (C++ member), 271

amici::ModelStateDerived::dJydy\_ (C++ member), 270

amici::ModelStateDerived::dJydy\_matlab\_ (C++ member), 270

amici::ModelStateDerived::dJzdp\_ (C++ member), 271

amici::ModelStateDerived::dJzdsigma\_ (C++ member), 271

amici::ModelStateDerived::dJzdx\_ (C++ member), 271

amici::ModelStateDerived::dJzdz\_ (C++ member), 271

amici::ModelStateDerived::drzdp\_ (C++ member), 271

amici::ModelStateDerived::drzdx\_ (C++ member), 271

amici::ModelStateDerived::dsigmaydp\_ (C++ member), 272

amici::ModelStateDerived::dsigmazdp\_ (C++ member), 272

amici::ModelStateDerived::dwdp\_ (C++ member), 270

amici::ModelStateDerived::dwdx\_ (C++ member), 270

amici::ModelStateDerived::dxdotdp (C++ member), 270

amici::ModelStateDerived::dxdotdp\_explicit (C++ member), 270

amici::ModelStateDerived::dxdotdp\_full (C++ member), 270

amici::ModelStateDerived::dxdotdp\_implicit (C++ member), 270

amici::ModelStateDerived::dxdotdw\_ (C++ member), 270

amici::ModelStateDerived::dxdotdx\_explicit (C++ member), 270

amici::ModelStateDerived::dxdotdx\_implicit (C++ member), 270

amici::ModelStateDerived::dydp\_ (C++ member), 271

amici::ModelStateDerived::dydx\_ (C++ member), 271

amici::ModelStateDerived::dzdp\_ (C++ member), 271

amici::ModelStateDerived::dzdx\_ (C++ member), 271

amici::ModelStateDerived::J\_ (C++ member), 270

amici::ModelStateDerived::JB\_ (C++ member), 270

amici::ModelStateDerived::M\_ (C++ member), 270

amici::ModelStateDerived::ModelStateDerived (C++ function), 270

amici::ModelStateDerived::rz\_ (C++ member), 272

amici::ModelStateDerived::sigmay\_ (C++ member), 271

amici::ModelStateDerived::sigmaz\_ (C++ member), 272

amici::ModelStateDerived::sx\_ (C++ member), 271

amici::ModelStateDerived::sx\_rdata\_

(C++ member), 271  
 amici::ModelStateDerived::w\_ (C++ member), 271  
 amici::ModelStateDerived::x\_pos\_tmp\_ (C++ member), 272  
 amici::ModelStateDerived::x\_rdata\_ (C++ member), 271  
 amici::ModelStateDerived::y\_ (C++ member), 271  
 amici::ModelStateDerived::z\_ (C++ member), 272  
 amici::N\_VGetArrayPointerConst (C++ function), 492  
 amici::NewtonDampingFactorMode (C++ enum), 468  
 amici::NewtonDampingFactorMode::off (C++ enumerator), 468  
 amici::NewtonDampingFactorMode::on (C++ enumerator), 468  
 amici::NewtonFailure (C++ class), 388  
 amici::NewtonFailure::error\_code (C++ member), 388  
 amici::NewtonFailure::NewtonFailure (C++ function), 388  
 amici::NewtonSolver (C++ class), 388  
 amici::NewtonSolver::~~NewtonSolver (C++ function), 390  
 amici::NewtonSolver::atol\_ (C++ member), 390  
 amici::NewtonSolver::computeNewtonSensis (C++ function), 389  
 amici::NewtonSolver::damping\_factor\_lower\_bound (C++ member), 390  
 amici::NewtonSolver::damping\_factor\_mode\_ (C++ member), 390  
 amici::NewtonSolver::dx\_ (C++ member), 390  
 amici::NewtonSolver::dxB\_ (C++ member), 391  
 amici::NewtonSolver::getNumLinSteps (C++ function), 389  
 amici::NewtonSolver::getSolver (C++ function), 390  
 amici::NewtonSolver::getStep (C++ function), 389  
 amici::NewtonSolver::max\_lin\_steps\_ (C++ member), 390  
 amici::NewtonSolver::max\_steps (C++ member), 390  
 amici::NewtonSolver::model\_ (C++ member), 390  
 amici::NewtonSolver::NewtonSolver (C++ function), 389  
 amici::NewtonSolver::num\_lin\_steps\_ (C++ member), 390  
 amici::NewtonSolver::prepareLinearSystem (C++ function), 389  
 amici::NewtonSolver::prepareLinearSystemB (C++ function), 389  
 amici::NewtonSolver::rtol\_ (C++ member), 390  
 amici::NewtonSolver::solveLinearSystem (C++ function), 389  
 amici::NewtonSolver::t\_ (C++ member), 390  
 amici::NewtonSolver::x\_ (C++ member), 390  
 amici::NewtonSolver::xB\_ (C++ member), 391  
 amici::NewtonSolver::xdot\_ (C++ member), 390  
 amici::NewtonSolverDense (C++ class), 391  
 amici::NewtonSolverDense::~~NewtonSolverDense (C++ function), 391  
 amici::NewtonSolverDense::NewtonSolverDense (C++ function), 391  
 amici::NewtonSolverDense::operator= (C++ function), 391  
 amici::NewtonSolverDense::prepareLinearSystem (C++ function), 391  
 amici::NewtonSolverDense::prepareLinearSystemB (C++ function), 392  
 amici::NewtonSolverDense::solveLinearSystem (C++ function), 391  
 amici::NewtonSolverIterative (C++ class), 392  
 amici::NewtonSolverIterative::~~NewtonSolverIterative (C++ function), 392  
 amici::NewtonSolverIterative::linsolveSPBCG (C++ function), 393  
 amici::NewtonSolverIterative::NewtonSolverIterative (C++ function), 392  
 amici::NewtonSolverIterative::prepareLinearSystem (C++ function), 392  
 amici::NewtonSolverIterative::prepareLinearSystemB (C++ function), 393  
 amici::NewtonSolverIterative::solveLinearSystem (C++ function), 392  
 amici::NewtonSolverSparse (C++ class), 393  
 amici::NewtonSolverSparse::~~NewtonSolverSparse (C++ function), 394  
 amici::NewtonSolverSparse::NewtonSolverSparse (C++ function), 393, 394  
 amici::NewtonSolverSparse::operator= (C++ function), 394  
 amici::NewtonSolverSparse::prepareLinearSystem (C++ function), 394  
 amici::NewtonSolverSparse::prepareLinearSystemB (C++ function), 394  
 amici::NewtonSolverSparse::solveLinearSystem (C++ function), 394  
 amici::NonlinearSolverIteration (C++

*enum*), 468  
amici::NonlinearSolverIteration::fixedpoint (C++ enumerator), 468  
amici::NonlinearSolverIteration::function (C++ enumerator), 468  
amici::NonlinearSolverIteration::newton (C++ enumerator), 468  
amici::operator== (C++ function), 493  
amici::outputFunctionType (C++ type), 517  
amici::ParameterScaling (C++ enum), 468  
amici::ParameterScaling::ln (C++ enumerator), 468  
amici::ParameterScaling::log10 (C++ enumerator), 468  
amici::ParameterScaling::none (C++ enumerator), 468  
amici::pi (C++ member), 512  
amici::pos\_pow (C++ function), 494  
amici::printErrMsgIdAndTxt (C++ function), 494  
amici::printfToString (C++ function), 494  
amici::printWarnMsgIdAndTxt (C++ function), 495  
amici::RDataReporting (C++ enum), 469  
amici::RDataReporting::full (C++ enumerator), 469  
amici::RDataReporting::likelihood (C++ enumerator), 469  
amici::RDataReporting::residuals (C++ enumerator), 469  
amici::realtype (C++ type), 517  
amici::regexErrorToString (C++ function), 495  
amici::reorder (C++ function), 495  
amici::ReturnData (C++ class), 394  
amici::ReturnData::~~ReturnData (C++ function), 395  
amici::ReturnData::applyChainRuleFactorToSimilarResults (C++ function), 401  
amici::ReturnData::boost::serialization::serialize (C++ function), 403  
amici::ReturnData::chi2 (C++ member), 398  
amici::ReturnData::computingFSA (C++ function), 402  
amici::ReturnData::cpu\_time (C++ member), 397  
amici::ReturnData::cpu\_timeB (C++ member), 397  
amici::ReturnData::dx\_solver\_ (C++ member), 403  
amici::ReturnData::fchi2 (C++ function), 401  
amici::ReturnData::fFIM (C++ function), 401  
amici::ReturnData::FIM (C++ member), 397  
amici::ReturnData::fres (C++ function), 401  
amici::ReturnData::fsres (C++ function), 401  
amici::ReturnData::getDataOutput (C++ function), 402  
amici::ReturnData::getDataSensisFSA (C++ function), 402  
amici::ReturnData::getEventOutput (C++ function), 402  
amici::ReturnData::getEventSensisFSA (C++ function), 402  
amici::ReturnData::handleSx0Backward (C++ function), 403  
amici::ReturnData::handleSx0Forward (C++ function), 403  
amici::ReturnData::initializeFullReporting (C++ function), 399  
amici::ReturnData::initializeLikelihoodReporting (C++ function), 399  
amici::ReturnData::initializeObjectiveFunction (C++ function), 399  
amici::ReturnData::initializeResidualReporting (C++ function), 399  
amici::ReturnData::invalidate (C++ function), 401  
amici::ReturnData::invalidateLLH (C++ function), 401  
amici::ReturnData::invalidateSLLH (C++ function), 401  
amici::ReturnData::J (C++ member), 396  
amici::ReturnData::llh (C++ member), 398  
amici::ReturnData::newton\_maxsteps (C++ member), 399  
amici::ReturnData::nmaxevent (C++ member), 398  
amici::ReturnData::nplist (C++ member), 398  
amici::ReturnData::nroots\_ (C++ member), 403  
amici::ReturnData::nroots (C++ member), 399  
amici::ReturnData::numerrtestfails (C++ member), 397  
amici::ReturnData::numerrtestfailsB (C++ member), 397  
amici::ReturnData::numnonlinsolvconvfails (C++ member), 397  
amici::ReturnData::numnonlinsolvconvfailsB (C++ member), 397  
amici::ReturnData::numrhsevals (C++ member), 397  
amici::ReturnData::numrhsevalsB (C++ member), 397  
amici::ReturnData::numsteps (C++ member), 397  
amici::ReturnData::numstepsB (C++ member), 397



---

`amici::ReturnData::nx (C++ member), 398`  
`amici::ReturnData::nxtrue (C++ member), 398`  
`amici::ReturnData::o2mode (C++ member), 399`  
`amici::ReturnData::order (C++ member), 397`  
`amici::ReturnData::posteq_cpu_time (C++ member), 397`  
`amici::ReturnData::posteq_cpu_timeB (C++ member), 397`  
`amici::ReturnData::posteq_numlinsteps (C++ member), 398`  
`amici::ReturnData::posteq_numsteps (C++ member), 398`  
`amici::ReturnData::posteq_numstepsB (C++ member), 398`  
`amici::ReturnData::posteq_status (C++ member), 397`  
`amici::ReturnData::posteq_t (C++ member), 398`  
`amici::ReturnData::posteq_wrms (C++ member), 398`  
`amici::ReturnData::preeq_cpu_time (C++ member), 397`  
`amici::ReturnData::preeq_cpu_timeB (C++ member), 397`  
`amici::ReturnData::preeq_numlinsteps (C++ member), 397`  
`amici::ReturnData::preeq_numsteps (C++ member), 397`  
`amici::ReturnData::preeq_numstepsB (C++ member), 397`  
`amici::ReturnData::preeq_status (C++ member), 397`  
`amici::ReturnData::preeq_t (C++ member), 398`  
`amici::ReturnData::preeq_wrms (C++ member), 398`  
`amici::ReturnData::processBackwardProblem (C++ function), 400`  
`amici::ReturnData::processForwardProblem (C++ function), 400`  
`amici::ReturnData::processPostEquilibration (C++ function), 400`  
`amici::ReturnData::processPreEquilibration (C++ function), 399`  
`amici::ReturnData::processSimulationObjects (C++ function), 395`  
`amici::ReturnData::processSolver (C++ function), 400`  
`amici::ReturnData::pscale (C++ member), 399`  
`amici::ReturnData::rdata_reporting (C++ member), 399`  
`amici::ReturnData::readSimulationState (C++ function), 400`  
`amici::ReturnData::res (C++ member), 396`  
`amici::ReturnData::ReturnData (C++ function), 395`  
`amici::ReturnData::rz (C++ member), 396`  
`amici::ReturnData::s2llh (C++ member), 398`  
`amici::ReturnData::s2rz (C++ member), 396`  
`amici::ReturnData::sensi (C++ member), 399`  
`amici::ReturnData::sensi_meth (C++ member), 399`  
`amici::ReturnData::sigmay (C++ member), 396`  
`amici::ReturnData::sigmaz (C++ member), 396`  
`amici::ReturnData::sllh (C++ member), 398`  
`amici::ReturnData::sres (C++ member), 396`  
`amici::ReturnData::srz (C++ member), 396`  
`amici::ReturnData::ssigmay (C++ member), 396`  
`amici::ReturnData::ssigmaz (C++ member), 396`  
`amici::ReturnData::status (C++ member), 398`  
`amici::ReturnData::storeJacobianAndDerivativeInReturn (C++ function), 400`  
`amici::ReturnData::sx (C++ member), 396`  
`amici::ReturnData::sx0 (C++ member), 398`  
`amici::ReturnData::sx_rdata_ (C++ member), 403`  
`amici::ReturnData::sx_solver_ (C++ member), 403`  
`amici::ReturnData::sx_ss (C++ member), 398`  
`amici::ReturnData::sy (C++ member), 396`  
`amici::ReturnData::sz (C++ member), 396`  
`amici::ReturnData::t_ (C++ member), 403`  
`amici::ReturnData::ts (C++ member), 396`  
`amici::ReturnData::w (C++ member), 396`  
`amici::ReturnData::x (C++ member), 396`  
`amici::ReturnData::x0 (C++ member), 398`  
`amici::ReturnData::x_rdata_ (C++ member), 403`  
`amici::ReturnData::x_solver_ (C++ member), 403`  
`amici::ReturnData::x_ss (C++ member), 398`  
`amici::ReturnData::xdot (C++ member), 396`  
`amici::ReturnData::y (C++ member), 396`  
`amici::ReturnData::z (C++ member), 396`  
`amici::runAmiciSimulation (C++ function), 496`  
`amici::runAmiciSimulations (C++ function), 496`  
`amici::scaleParameters (C++ function), 497`  
`amici::SecondOrderMode (C++ enum), 469`

`amici::SecondOrderMode::directional` (*C++ member*), 406  
`(C++ enumerator)`, 469  
`amici::SecondOrderMode::full` (*C++ enumerator*), 469  
`amici::SecondOrderMode::none` (*C++ enumerator*), 469  
`amici::SensitivityMethod` (*C++ enum*), 469  
`amici::SensitivityMethod::adjoint` (*C++ enumerator*), 469  
`amici::SensitivityMethod::forward` (*C++ enumerator*), 469  
`amici::SensitivityMethod::none` (*C++ enumerator*), 469  
`amici::SensitivityOrder` (*C++ enum*), 470  
`amici::SensitivityOrder::first` (*C++ enumerator*), 470  
`amici::SensitivityOrder::none` (*C++ enumerator*), 470  
`amici::SensitivityOrder::second` (*C++ enumerator*), 470  
`amici::serializeToChar` (*C++ function*), 497  
`amici::serializeToStdVec` (*C++ function*), 497  
`amici::serializeToString` (*C++ function*), 498  
`amici::setModelData` (*C++ function*), 498  
`amici::setSolverOptions` (*C++ function*), 498  
`amici::SetupFailure` (*C++ class*), 404  
`amici::setupReturnData` (*C++ function*), 499  
`amici::seval` (*C++ function*), 499  
`amici::sign` (*C++ function*), 499  
`amici::SimulationParameters` (*C++ class*), 404  
`amici::SimulationParameters::fixedParameters` (*C++ member*), 406  
`amici::SimulationParameters::fixedParametersPresim` (*C++ member*), 406  
`amici::SimulationParameters::fixedParametersPresim_t30n` (*C++ member*), 406  
`amici::SimulationParameters::parameters` (*C++ member*), 406  
`amici::SimulationParameters::plist` (*C++ member*), 406  
`amici::SimulationParameters::pscale` (*C++ member*), 406  
`amici::SimulationParameters::reinitialization_function` (*C++ member*), 406  
`amici::SimulationParameters::reinitialization_sim` (*C++ member*), 406  
`amici::SimulationParameters::reinitializeFixedParametersDependentInitialStates` (*C++ function*), 405  
`amici::SimulationParameters::reinitializeFixedParametersDependentInitialStatesForPresimulation` (*C++ function*), 405  
`amici::SimulationParameters::reinitializeFixedParametersDependentlyMethods` (*C++ function*), 405  
`amici::SimulationParameters::reinitializeFixedParametersNone` (*C++ function*), 407  
`(C++ member)`, 406  
`amici::SimulationParameters::SimulationParameters` (*C++ function*), 404, 405  
`amici::SimulationParameters::sx0` (*C++ member*), 406  
`amici::SimulationParameters::t_presim` (*C++ member*), 406  
`amici::SimulationParameters::ts_` (*C++ member*), 406  
`amici::SimulationParameters::tstart_` (*C++ member*), 406  
`amici::SimulationParameters::x0` (*C++ member*), 406  
`amici::SimulationState` (*C++ struct*), 272  
`amici::SimulationState::dx` (*C++ member*), 272  
`amici::SimulationState::state` (*C++ member*), 272  
`amici::SimulationState::sx` (*C++ member*), 272  
`amici::SimulationState::t` (*C++ member*), 272  
`amici::SimulationState::x` (*C++ member*), 272  
`amici::sinteg` (*C++ function*), 499  
`amici::slice` (*C++ function*), 500  
`amici::Solver` (*C++ class*), 407  
`amici::Solver::~~Solver` (*C++ function*), 407  
`amici::Solver::adjInit` (*C++ function*), 427  
`amici::Solver::allocateSolver` (*C++ function*), 424  
`amici::Solver::allocateSolverB` (*C++ function*), 427  
`amici::Solver::applyQuadTolerances` (*C++ function*), 430  
`amici::Solver::applyQuadTolerancesASA` (*C++ function*), 430  
`amici::Solver::applySensitivityTolerances` (*C++ function*), 431  
`amici::Solver::applyTolerances` (*C++ function*), 430  
`amici::Solver::applyTolerancesASA` (*C++ function*), 430  
`amici::Solver::applyTolerancesFSA` (*C++ function*), 430  
`amici::Solver::binit` (*C++ function*), 422  
`amici::Solver::computeDependentInitialStates` (*C++ function*), 433  
`amici::Solver::computeDependentInitialStatesForPresimulation` (*C++ function*), 433  
`amici::Solver::calcICB` (*C++ function*), 409  
`amici::Solver::computeDependentInitialStatesForSimulation` (*C++ function*), 431  
`amici::Solver::computeDependentInitialStatesForSimulationPresimulation` (*C++ function*), 431

---

<code>amici::Solver::computingASA (C++ function),</code>	<code>amici::Solver::getLinearMultistepMethod</code>
<code>419</code>	<code>(C++ function), 414</code>
<code>amici::Solver::computingFSA (C++ function),</code>	<code>amici::Solver::getLinearSolver (C++ func-</code>
<code>419</code>	<code>tion), 416</code>
<code>amici::Solver::diag (C++ function), 430</code>	<code>amici::Solver::getMaxSteps (C++ function),</code>
<code>amici::Solver::diagB (C++ function), 430</code>	<code>414</code>
<code>amici::Solver::dky_ (C++ member), 432</code>	<code>amici::Solver::getMaxStepsBackwardProblem</code>
<code>amici::Solver::dx_ (C++ member), 432</code>	<code>(C++ function), 414</code>
<code>amici::Solver::dxB_ (C++ member), 432</code>	<code>amici::Solver::getModel (C++ function), 429</code>
<code>amici::Solver::force_reinit_postprocess_</code>	<code>amici::Solver::getNewtonDampingFactorLowerBound</code>
<code>Bm (C++ member), 433</code>	<code>(C++ function), 411</code>
<code>amici::Solver::force_reinit_postprocess_</code>	<code>amici::Solver::getNewtonDampingFactorMode</code>
<code>Bm (C++ member), 432</code>	<code>(C++ function), 411</code>
<code>amici::Solver::getAbsoluteTolerance</code>	<code>amici::Solver::getNewtonMaxLinearSteps</code>
<code>(C++ function), 412</code>	<code>(C++ function), 410</code>
<code>amici::Solver::getAbsoluteToleranceB</code>	<code>amici::Solver::getNewtonMaxSteps (C++</code>
<code>(C++ function), 412</code>	<code>function), 410</code>
<code>amici::Solver::getAbsoluteToleranceFSA</code>	<code>amici::Solver::getNonlinearSolverIteration</code>
<code>(C++ function), 412</code>	<code>(C++ function), 415</code>
<code>amici::Solver::getAbsoluteToleranceQuadrature</code>	<code>amici::Solver::getNumErrTestFails (C++</code>
<code>(C++ function), 413</code>	<code>function), 420, 428</code>
<code>amici::Solver::getAbsoluteToleranceSteadyState</code>	<code>amici::Solver::getNumErrTestFailsB (C++</code>
<code>(C++ function), 413</code>	<code>function), 420</code>
<code>amici::Solver::getAbsoluteToleranceSteadyStateSolve</code>	<code>amici::Solver::getNumNonlinSolvConvFails</code>
<code>(C++ function), 414</code>	<code>(C++ function), 420, 428</code>
<code>amici::Solver::getAdjBmem (C++ function),</code>	<code>amici::Solver::getNumNonlinSolvConvFailsB</code>
<code>430</code>	<code>(C++ function), 420</code>
<code>amici::Solver::getAdjInitDone (C++ func-</code>	<code>amici::Solver::getNumRhsEvals (C++ func-</code>
<code>tion), 429</code>	<code>tion), 420, 427</code>
<code>amici::Solver::getAdjointDerivativeState</code>	<code>amici::Solver::getNumRhsEvalsB (C++ func-</code>
<code>(C++ function), 417</code>	<code>tion), 420</code>
<code>amici::Solver::getAdjointQuadrature</code>	<code>amici::Solver::getNumSteps (C++ function),</code>
<code>(C++ function), 417</code>	<code>420, 427</code>
<code>amici::Solver::getAdjointState (C++ func-</code>	<code>amici::Solver::getNumStepsB (C++ function),</code>
<code>tion), 417</code>	<code>420</code>
<code>amici::Solver::getB (C++ function), 421</code>	<code>amici::Solver::getPreequilibration (C++</code>
<code>amici::Solver::getCpuTime (C++ function),</code>	<code>function), 410</code>
<code>419</code>	<code>amici::Solver::getQuad (C++ function), 422</code>
<code>amici::Solver::getCpuTimeB (C++ function),</code>	<code>amici::Solver::getQuadB (C++ function), 422</code>
<code>419</code>	<code>amici::Solver::getQuadDky (C++ function),</code>
<code>amici::Solver::getDerivativeState (C++</code>	<code>426</code>
<code>function), 417</code>	<code>amici::Solver::getQuadDkyB (C++ function),</code>
<code>amici::Solver::getDky (C++ function), 426</code>	<code>426</code>
<code>amici::Solver::getDkyB (C++ function), 426</code>	<code>amici::Solver::getQuadInitDone (C++ func-</code>
<code>amici::Solver::getInitDone (C++ function),</code>	<code>tion), 430</code>
<code>429</code>	<code>amici::Solver::getQuadInitDoneB (C++</code>
<code>amici::Solver::getInitDoneB (C++ function),</code>	<code>function), 429</code>
<code>429</code>	<code>amici::Solver::getQuadrature (C++ func-</code>
<code>amici::Solver::getInternalSensitivityMethod</code>	<code>tion), 418</code>
<code>(C++ function), 416</code>	<code>amici::Solver::getRelativeTolerance</code>
<code>amici::Solver::getInterpolationType</code>	<code>(C++ function), 411</code>
<code>(C++ function), 415</code>	<code>amici::Solver::getRelativeToleranceB</code>
<code>amici::Solver::getLastOrder (C++ function),</code>	<code>(C++ function), 412</code>
<code>420, 428</code>	<code>amici::Solver::getRelativeToleranceFSA</code>

```

    (C++ function), 412
amici::Solver::getRelativeToleranceQuadratures (C++ function), 413
amici::Solver::getRelativeToleranceSteadyState (C++ function), 413
amici::Solver::getRelativeToleranceSteadyStateSensitivity (C++ function), 414
amici::Solver::getReturnDataReportingMode (C++ function), 416
amici::Solver::getRootInfo (C++ function), 409
amici::Solver::getSens (C++ function), 421
amici::Solver::getSensDky (C++ function), 426
amici::Solver::getSensInitDone (C++ function), 429
amici::Solver::getSensitivityMethod (C++ function), 410
amici::Solver::getSensitivityMethodPreequilibration (C++ function), 410
amici::Solver::getSensitivityOrder (C++ function), 411
amici::Solver::getStabilityLimitFlag (C++ function), 415
amici::Solver::getState (C++ function), 417
amici::Solver::getStateOrdering (C++ function), 415
amici::Solver::getStateSensitivity (C++ function), 417
amici::Solver::gett (C++ function), 419
amici::Solver::init (C++ function), 422
amici::Solver::initializeLinearSolver (C++ function), 428
amici::Solver::initializeLinearSolverB (C++ function), 429
amici::Solver::initializeNonLinearSolver (C++ function), 428
amici::Solver::initializeNonLinearSolverB (C++ function), 429
amici::Solver::initializeNonLinearSolverSensitivity (C++ function), 423
amici::Solver::initSteadystate (C++ function), 422
amici::Solver::interp_type_ (C++ member), 432
amici::Solver::ism_ (C++ member), 431
amici::Solver::iter_ (C++ member), 432
amici::Solver::linear_solver_ (C++ member), 432
amici::Solver::linear_solver_B_ (C++ member), 432
amici::Solver::lmm_ (C++ member), 431
amici::Solver::maxsteps_ (C++ member), 432
amici::Solver::non_linear_solver_ (C++ member), 432
amici::Solver::non_linear_solver_B_ (C++ member), 432
amici::Solver::non_linear_solver_sensitivity_ (C++ member), 432
amici::Solver::nplist (C++ function), 419
amici::Solver::nquad (C++ function), 419
amici::Solver::nx (C++ function), 419
amici::Solver::operator== (C++ function), 433
amici::Solver::qbinit (C++ function), 423
amici::Solver::quadInit (C++ function), 427
amici::Solver::quadReInitB (C++ function), 418
amici::Solver::quadSStolerances (C++ function), 427
amici::Solver::quadSStolerancesB (C++ function), 427
amici::Solver::reInit (C++ function), 418
amici::Solver::reInitB (C++ function), 418
amici::Solver::reInitPostProcessB (C++ function), 421
amici::Solver::reInitPostProcessF (C++ function), 421
amici::Solver::resetDiagnosis (C++ function), 419
amici::Solver::resetMutableMemory (C++ function), 430
amici::Solver::rootInit (C++ function), 423
amici::Solver::run (C++ function), 407
amici::Solver::runB (C++ function), 408
amici::Solver::sdx_ (C++ member), 432
amici::Solver::sensInit1 (C++ function), 422
amici::Solver::sensReInit (C++ function), 418
amici::Solver::sensToggleOff (C++ function), 418
amici::Solver::setAbsoluteTolerance (C++ function), 412
amici::Solver::setAbsoluteToleranceB (C++ function), 413
amici::Solver::setAbsoluteToleranceFSA (C++ function), 412
amici::Solver::setAbsoluteToleranceQuadratures (C++ function), 413
amici::Solver::setAbsoluteToleranceSteadyState (C++ function), 413
amici::Solver::setAbsoluteToleranceSteadyStateSensitivity (C++ function), 414
amici::Solver::setAdjInitDone (C++ function), 431
amici::Solver::setBandJacFn (C++ function), 423
amici::Solver::setBandJacFnB (C++ function), 423

```

tion), 423  
 amici::Solver::setDenseJacFn (C++ function), 423  
 amici::Solver::setDenseJacFnB (C++ function), 423  
 amici::Solver::setErrHandlerFn (C++ function), 424  
 amici::Solver::setId (C++ function), 425  
 amici::Solver::setInitDone (C++ function), 431  
 amici::Solver::setInitDoneB (C++ function), 431  
 amici::Solver::setInternalSensitivityMethod (C++ function), 416  
 amici::Solver::setInterpolationType (C++ function), 415  
 amici::Solver::setJacTimesVecFn (C++ function), 423  
 amici::Solver::setJacTimesVecFnB (C++ function), 423  
 amici::Solver::setLinearMultistepMethod (C++ function), 415  
 amici::Solver::setLinearSolver (C++ function), 416, 428  
 amici::Solver::setLinearSolverB (C++ function), 428  
 amici::Solver::setMaxNumSteps (C++ function), 425  
 amici::Solver::setMaxNumStepsB (C++ function), 425  
 amici::Solver::setMaxSteps (C++ function), 414  
 amici::Solver::setMaxStepsBackwardProblem (C++ function), 414  
 amici::Solver::setNewtonDampingFactorLowerBound (C++ function), 411  
 amici::Solver::setNewtonDampingFactorMode (C++ function), 411  
 amici::Solver::setNewtonMaxLinearSteps (C++ function), 411  
 amici::Solver::setNewtonMaxSteps (C++ function), 410  
 amici::Solver::setNonLinearSolver (C++ function), 428  
 amici::Solver::setNonLinearSolverB (C++ function), 428  
 amici::Solver::setNonlinearSolverIteration (C++ function), 415  
 amici::Solver::setNonLinearSolverSens (C++ function), 429  
 amici::Solver::setPreequilibration (C++ function), 410  
 amici::Solver::setQuadErrCon (C++ function), 424  
 amici::Solver::setQuadErrConB (C++ function), 424  
 amici::Solver::setQuadInitDone (C++ function), 431  
 amici::Solver::setQuadInitDoneB (C++ function), 431  
 amici::Solver::setRelativeTolerance (C++ function), 411  
 amici::Solver::setRelativeToleranceB (C++ function), 412  
 amici::Solver::setRelativeToleranceFSA (C++ function), 412  
 amici::Solver::setRelativeToleranceQuadratures (C++ function), 413  
 amici::Solver::setRelativeToleranceSteadyState (C++ function), 413  
 amici::Solver::setRelativeToleranceSteadyStateSens (C++ function), 414  
 amici::Solver::setReturnDataReportingMode (C++ function), 416  
 amici::Solver::setSensErrCon (C++ function), 424  
 amici::Solver::setSensInitDone (C++ function), 431  
 amici::Solver::setSensInitOff (C++ function), 431  
 amici::Solver::setSensitivityMethod (C++ function), 410  
 amici::Solver::setSensitivityMethodPreequilibration (C++ function), 410  
 amici::Solver::setSensitivityOrder (C++ function), 411  
 amici::Solver::setSensParams (C++ function), 426  
 amici::Solver::setSensSSStolerances (C++ function), 424  
 amici::Solver::setSparseJacFn (C++ function), 423  
 amici::Solver::setSparseJacFn\_ss (C++ function), 424  
 amici::Solver::setSparseJacFnB (C++ function), 423  
 amici::Solver::setSSStolerances (C++ function), 424  
 amici::Solver::setSSStolerancesB (C++ function), 427  
 amici::Solver::setStabilityLimitFlag (C++ function), 415  
 amici::Solver::setStabLimDet (C++ function), 425  
 amici::Solver::setStabLimDetB (C++ function), 425  
 amici::Solver::setStateOrdering (C++ function), 415



`amici::Solver::setStopTime (C++ function), 421`  
`amici::Solver::setSuppressAlg (C++ function), 425`  
`amici::Solver::setup (C++ function), 408`  
`amici::Solver::setupB (C++ function), 408`  
`amici::Solver::setupSteadystate (C++ function), 408`  
`amici::Solver::setUserData (C++ function), 424`  
`amici::Solver::setUserDataB (C++ function), 425`  
`amici::Solver::solve (C++ function), 421`  
`amici::Solver::solveB (C++ function), 409`  
`amici::Solver::solveF (C++ function), 421`  
`amici::Solver::Solver (C++ function), 407`  
`amici::Solver::solver_memory_ (C++ member), 431`  
`amici::Solver::solver_memory_B_ (C++ member), 431`  
`amici::Solver::solver_was_called_B_ (C++ member), 432`  
`amici::Solver::solver_was_called_F_ (C++ member), 432`  
`amici::Solver::step (C++ function), 408`  
`amici::Solver::storeDiagnosis (C++ function), 419`  
`amici::Solver::storeDiagnosisB (C++ function), 419`  
`amici::Solver::switchForwardSensisOff (C++ function), 410`  
`amici::Solver::sx_ (C++ member), 432`  
`amici::Solver::t_ (C++ member), 432`  
`amici::Solver::turnOffRootFinding (C++ function), 409`  
`amici::Solver::updateAndReinitStatesAndSensis (C++ function), 409`  
`amici::Solver::writeSolution (C++ function), 416`  
`amici::Solver::writeSolutionB (C++ function), 416`  
`amici::Solver::x_ (C++ member), 432`  
`amici::Solver::xB_ (C++ member), 432`  
`amici::Solver::xQ_ (C++ member), 432`  
`amici::Solver::xQB_ (C++ member), 432`  
`amici::spline (C++ function), 501`  
`amici::spline_pos (C++ function), 501`  
`amici::SteadyStateContext (C++ enum), 470`  
`amici::SteadyStateContext::newtonSensi (C++ enumerator), 470`  
`amici::SteadyStateContext::sensiStorage (C++ enumerator), 470`  
`amici::SteadyStateContext::solverCreation (C++ enumerator), 470`  
`amici::SteadystateProblem (C++ class), 433`  
`amici::SteadystateProblem::applyNewtonsMethod (C++ function), 436`  
`amici::SteadystateProblem::checkConvergence (C++ function), 436`  
`amici::SteadystateProblem::checkSteadyStateSuccess (C++ function), 439`  
`amici::SteadystateProblem::computeQBfromQ (C++ function), 437`  
`amici::SteadystateProblem::computeSteadyStateQuadrature (C++ function), 434`  
`amici::SteadystateProblem::createSteadystateSimSolver (C++ function), 436`  
`amici::SteadystateProblem::findSteadyState (C++ function), 434`  
`amici::SteadystateProblem::findSteadyStateByNewton (C++ function), 434`  
`amici::SteadystateProblem::findSteadyStateBySimulation (C++ function), 434`  
`amici::SteadystateProblem::getAdjointQuadrature (C++ function), 439`  
`amici::SteadystateProblem::getAdjointUpdates (C++ function), 438`  
`amici::SteadystateProblem::getCPUTime (C++ function), 438`  
`amici::SteadystateProblem::getCPUTimeB (C++ function), 438`  
`amici::SteadystateProblem::getDJydx (C++ function), 438`  
`amici::SteadystateProblem::getEquilibrationQuadrature (C++ function), 437`  
`amici::SteadystateProblem::getFinalSimulationState (C++ function), 437`  
`amici::SteadystateProblem::getNumLinSteps (C++ function), 438`  
`amici::SteadystateProblem::getNumSteps (C++ function), 438`  
`amici::SteadystateProblem::getNumStepsB (C++ function), 438`  
`amici::SteadystateProblem::getQuadratureByLinSolve (C++ function), 435`  
`amici::SteadystateProblem::getQuadratureBySimulation (C++ function), 435`  
`amici::SteadystateProblem::getResidualNorm (C++ function), 438`  
`amici::SteadystateProblem::getSensitivityFlag (C++ function), 435`  
`amici::SteadystateProblem::getState (C++ function), 437`  
`amici::SteadystateProblem::getStateSensitivity (C++ function), 437`  
`amici::SteadystateProblem::getSteadyStateStatus (C++ function), 438`  
`amici::SteadystateProblem::getSteadyStateTime`

(C++ function), 438  
 amici::SteadystateProblem::getWrmsNorm (C++ function), 435  
 amici::SteadystateProblem::handleSteadyStateFailure (C++ function), 435  
 amici::SteadystateProblem::hasQuadratureamici::SUNLinSolKLU::StateOrdering (C++ enum), 441  
 amici::SteadystateProblem::initializeBackwardProblemamici::SUNLinSolKLU::StateOrdering::AMD (C++ enumerator), 441  
 amici::SteadystateProblem::runSteadystateSimulationamici::SUNLinSolKLU::StateOrdering::COLAMD (C++ enumerator), 441  
 amici::SteadystateProblem::SteadystateProblemamici::SUNLinSolKLU::StateOrdering::natural (C++ enumerator), 441  
 amici::SteadystateProblem::storeSimulationStateamici::SUNLinSolKLU::SUNLinSolKLU (C++ function), 441  
 amici::SteadystateProblem::workSteadyStateBackwardProblemamici::SUNLinSolPCG (C++ class), 442  
 amici::SteadystateProblem::workSteadyStateProblemamici::SUNLinSolPCG::getNumIters (C++ function), 443  
 amici::SteadystateProblem::writeErrorStringamici::SUNLinSolPCG::getResid (C++ function), 443  
 amici::SteadyStateSensitivityMode (C++ enum), 470  
 amici::SteadyStateSensitivityMode::newtonOnlyamici::SUNLinSolPCG::getResNorm (C++ function), 443  
 amici::SteadyStateSensitivityMode::simulationFailureamici::SUNLinSolPCG::setATimes (C++ function), 442  
 amici::SteadyStateStatus (C++ enum), 471  
 amici::SteadyStateStatus::failedamici::SUNLinSolPCG::setPreconditioner (C++ function), 443  
 amici::SteadyStateStatus::failed\_convergenceamici::SUNLinSolPCG::setScalingVectors (C++ function), 443  
 amici::SteadyStateStatus::failed\_dampingamici::SUNLinSolPCG::SUNLinSolPCG (C++ function), 442  
 amici::SteadyStateStatus::failed\_factorizationamici::SUNLinSolSPBCGS (C++ class), 444  
 amici::SteadyStateStatus::failed\_too\_long\_simulationamici::SUNLinSolSPBCGS::getNumIters (C++ function), 445  
 amici::SteadyStateStatus::not\_runamici::SUNLinSolSPBCGS::getResid (C++ function), 445  
 amici::SteadyStateStatus::successamici::SUNLinSolSPBCGS::getResNorm (C++ function), 445  
 amici::SUNLinSolBand (C++ class), 439  
 amici::SUNLinSolBand::getMatrix (C++ function), 440  
 amici::SUNLinSolBand::SUNLinSolBand (C++ function), 439  
 amici::SUNLinSolDense (C++ class), 440  
 amici::SUNLinSolDense::getMatrix (C++ function), 440  
 amici::SUNLinSolDense::SUNLinSolDense (C++ function), 440  
 amici::SUNLinSolKLU (C++ class), 441  
 amici::SUNLinSolKLU::getMatrix (C++ function), 441  
 amici::SUNLinSolKLU::reInit (C++ function), 441  
 amici::SUNLinSolKLU::setOrdering (C++ function), 442  
 amici::SUNLinSolKLU::StateOrdering (C++ enum), 441  
 amici::SUNLinSolKLU::StateOrdering::AMD (C++ enumerator), 441  
 amici::SUNLinSolKLU::StateOrdering::COLAMD (C++ enumerator), 441  
 amici::SUNLinSolKLU::StateOrdering::natural (C++ enumerator), 441  
 amici::SUNLinSolKLU::SUNLinSolKLU (C++ function), 441  
 amici::SUNLinSolPCG (C++ class), 442  
 amici::SUNLinSolPCG::getNumIters (C++ function), 443  
 amici::SUNLinSolPCG::getResid (C++ function), 443  
 amici::SUNLinSolPCG::getResNorm (C++ function), 443  
 amici::SUNLinSolPCG::setATimes (C++ function), 442  
 amici::SUNLinSolPCG::setPreconditioner (C++ function), 443  
 amici::SUNLinSolPCG::setScalingVectors (C++ function), 443  
 amici::SUNLinSolPCG::SUNLinSolPCG (C++ function), 442  
 amici::SUNLinSolSPBCGS (C++ class), 444  
 amici::SUNLinSolSPBCGS::getNumIters (C++ function), 445  
 amici::SUNLinSolSPBCGS::getResid (C++ function), 445  
 amici::SUNLinSolSPBCGS::getResNorm (C++ function), 445  
 amici::SUNLinSolSPBCGS::setATimes (C++ function), 444  
 amici::SUNLinSolSPBCGS::setPreconditioner (C++ function), 444  
 amici::SUNLinSolSPBCGS::setScalingVectors (C++ function), 445  
 amici::SUNLinSolSPBCGS::SUNLinSolSPBCGS (C++ function), 444  
 amici::SUNLinSolSPFGMR (C++ class), 445  
 amici::SUNLinSolSPFGMR::getNumIters (C++ function), 446  
 amici::SUNLinSolSPFGMR::getResid (C++ function), 446  
 amici::SUNLinSolSPFGMR::getResNorm (C++ function), 446  
 amici::SUNLinSolSPFGMR::setATimes (C++ function), 446

`amici::SUNLinSolSPFGMR::setPreconditioner` (C++ function), 446

`amici::SUNLinSolSPFGMR::setScalingVectors` (C++ function), 446

`amici::SUNLinSolSPFGMR::SUNLinSolSPFGMR` (C++ function), 446

`amici::SUNLinSolSPGMR` (C++ class), 447

`amici::SUNLinSolSPGMR::getNumIters` (C++ function), 448

`amici::SUNLinSolSPGMR::getResid` (C++ function), 448

`amici::SUNLinSolSPGMR::getResNorm` (C++ function), 448

`amici::SUNLinSolSPGMR::setATimes` (C++ function), 447

`amici::SUNLinSolSPGMR::setPreconditioner` (C++ function), 447

`amici::SUNLinSolSPGMR::setScalingVectors` (C++ function), 448

`amici::SUNLinSolSPGMR::SUNLinSolSPGMR` (C++ function), 447

`amici::SUNLinSolSPTFQMR` (C++ class), 448

`amici::SUNLinSolSPTFQMR::getNumIters` (C++ function), 449

`amici::SUNLinSolSPTFQMR::getResid` (C++ function), 450

`amici::SUNLinSolSPTFQMR::getResNorm` (C++ function), 450

`amici::SUNLinSolSPTFQMR::setATimes` (C++ function), 449

`amici::SUNLinSolSPTFQMR::setPreconditioner` (C++ function), 449

`amici::SUNLinSolSPTFQMR::setScalingVectors` (C++ function), 449

`amici::SUNLinSolSPTFQMR::SUNLinSolSPTFQMR` (C++ function), 449

`amici::SUNLinSolWrapper` (C++ class), 450

`amici::SUNLinSolWrapper::~SUNLinSolWrapper` (C++ function), 451

`amici::SUNLinSolWrapper::get` (C++ function), 451

`amici::SUNLinSolWrapper::getLastFlag` (C++ function), 452

`amici::SUNLinSolWrapper::getMatrix` (C++ function), 452

`amici::SUNLinSolWrapper::getType` (C++ function), 451

`amici::SUNLinSolWrapper::initialize` (C++ function), 452

`amici::SUNLinSolWrapper::operator=` (C++ function), 451

`amici::SUNLinSolWrapper::setup` (C++ function), 451, 452

`amici::SUNLinSolWrapper::Solve` (C++ function), 452

`amici::SUNLinSolWrapper::solver_` (C++ member), 453

`amici::SUNLinSolWrapper::space` (C++ function), 452

`amici::SUNLinSolWrapper::SUNLinSolWrapper` (C++ function), 451

`amici::SUNMatrixWrapper` (C++ class), 453

`amici::SUNMatrixWrapper::~SUNMatrixWrapper` (C++ function), 454

`amici::SUNMatrixWrapper::capacity` (C++ function), 455

`amici::SUNMatrixWrapper::columns` (C++ function), 455

`amici::SUNMatrixWrapper::data` (C++ function), 455

`amici::SUNMatrixWrapper::get` (C++ function), 454

`amici::SUNMatrixWrapper::get_data` (C++ function), 455

`amici::SUNMatrixWrapper::get_indexptr` (C++ function), 456

`amici::SUNMatrixWrapper::get_indexval` (C++ function), 456

`amici::SUNMatrixWrapper::matrix_id` (C++ function), 459

`amici::SUNMatrixWrapper::multiply` (C++ function), 457, 458

`amici::SUNMatrixWrapper::num_indexptrs` (C++ function), 455

`amici::SUNMatrixWrapper::num_nonzeros` (C++ function), 455

`amici::SUNMatrixWrapper::operator=` (C++ function), 454

`amici::SUNMatrixWrapper::realloc` (C++ function), 454

`amici::SUNMatrixWrapper::reallocate` (C++ function), 454

`amici::SUNMatrixWrapper::refresh` (C++ function), 459

`amici::SUNMatrixWrapper::rows` (C++ function), 455

`amici::SUNMatrixWrapper::scale` (C++ function), 457

`amici::SUNMatrixWrapper::scatter` (C++ function), 458

`amici::SUNMatrixWrapper::set_data` (C++ function), 456

`amici::SUNMatrixWrapper::set_indexptr` (C++ function), 457

`amici::SUNMatrixWrapper::set_indexptrs` (C++ function), 457

`amici::SUNMatrixWrapper::set_indexval` (C++ function), 456



amici::SUNMatrixWrapper::set\_indexvals (C++ function), 456  
 amici::SUNMatrixWrapper::sparse\_add (C++ function), 458  
 amici::SUNMatrixWrapper::sparse\_multiply (C++ function), 458  
 amici::SUNMatrixWrapper::sparse\_sum (C++ function), 458  
 amici::SUNMatrixWrapper::sparsetype (C++ function), 457  
 amici::SUNMatrixWrapper::SUNMatrixWrapper (C++ function), 453, 454  
 amici::SUNMatrixWrapper::to\_dense (C++ function), 459  
 amici::SUNMatrixWrapper::to\_diag (C++ function), 459  
 amici::SUNMatrixWrapper::transpose (C++ function), 459  
 amici::SUNMatrixWrapper::zero (C++ function), 459  
 amici::SUNNonLinSolFixedPoint (C++ class), 460  
 amici::SUNNonLinSolFixedPoint::getSysFn (C++ function), 460  
 amici::SUNNonLinSolFixedPoint::SUNNonLinSolFixedPoint (C++ function), 460  
 amici::SUNNonLinSolNewton (C++ class), 461  
 amici::SUNNonLinSolNewton::getSysFn (C++ function), 461  
 amici::SUNNonLinSolNewton::SUNNonLinSolNewton (C++ function), 461  
 amici::SUNNonLinSolWrapper (C++ class), 462  
 amici::SUNNonLinSolWrapper::~~SUNNonLinSolWrapper (C++ function), 462  
 amici::SUNNonLinSolWrapper::get (C++ function), 463  
 amici::SUNNonLinSolWrapper::getCurIter (C++ function), 464  
 amici::SUNNonLinSolWrapper::getNumConvFails (C++ function), 464  
 amici::SUNNonLinSolWrapper::getNumIters (C++ function), 464  
 amici::SUNNonLinSolWrapper::getType (C++ function), 463  
 amici::SUNNonLinSolWrapper::initialize (C++ function), 465  
 amici::SUNNonLinSolWrapper::operator= (C++ function), 462  
 amici::SUNNonLinSolWrapper::setConvTestFn (C++ function), 464  
 amici::SUNNonLinSolWrapper::setLSetupFn (C++ function), 463  
 amici::SUNNonLinSolWrapper::setLSolveFn (C++ function), 464  
 amici::SUNNonLinSolWrapper::setMaxIters (C++ function), 464  
 amici::SUNNonLinSolWrapper::setSysFn (C++ function), 463  
 amici::SUNNonLinSolWrapper::setup (C++ function), 463  
 amici::SUNNonLinSolWrapper::Solve (C++ function), 463  
 amici::SUNNonLinSolWrapper::solver (C++ member), 465  
 amici::SUNNonLinSolWrapper::SUNNonLinSolWrapper (C++ function), 462  
 amici::unscaleParameters (C++ function), 501  
 amici::wrapErrorHandlerFn (C++ function), 501  
 amici::writeMatlabField0 (C++ function), 502  
 amici::writeMatlabField1 (C++ function), 502  
 amici::writeMatlabField2 (C++ function), 503  
 amici::writeMatlabField3 (C++ function), 503  
 amici::writeMatlabField4 (C++ function), 504  
 amici::writeSlice (C++ function), 504, 505  
 AMICI\_H5\_RESTORE\_ERROR\_HANDLER (C macro), 513  
 AMICI\_H5\_SAVE\_ERROR\_HANDLER (C macro), 513  
 amici\_to\_petascale () (in module amici.parameter\_mapping), 251  
 AMICI\_VERSION (C macro), 513  
 amievent (built-in class), 530  
 amifun (built-in class), 531  
 append () (amici.parameter\_mapping.ParameterMapping method), 249  
 apply\_template () (in module amici.ode\_export), 238  
 as\_wrapperfun () (in module amici.gradient\_check), 247  
 assignmentRules2observables () (in module amici.sbml\_import), 194

## B

backtraceString () (in module amici.amici), 187  
 BoolVector (class in amici.amici), 130  
 boost::serialization::archiveVector (C++ function), 506  
 boost::serialization::serialize (C++ function), 506, 507

## C

cast\_to\_sym () (in module amici.ode\_export), 238  
 check\_close () (in module amici.gradient\_check), 247  
 check\_derivatives () (in module amici.gradient\_check), 247  
 check\_event\_support () (amici.sbml\_import.SbmlImporter method), 192

[check\\_finite\\_difference\(\)](#) (in module *amici.gradient\_check*), 248  
[check\\_results\(\)](#) (in module *amici.gradient\_check*), 248  
[check\\_support\(\)](#) (*amici.sbml\_import.SbmlImporter* method), 192  
[clone\(\)](#) (*amici.amici.Model* method), 148  
[clone\(\)](#) (*amici.amici.Solver* method), 177  
[colptrs\(\)](#) (*amici.ode\_export.ODEModel* method), 227  
[compile\\_model\(\)](#) (*amici.ode\_export.ODEExporter* method), 223  
[compiledWithOpenMP\(\)](#) (in module *amici.amici*), 187  
[computingASA\(\)](#) (*amici.amici.Solver* method), 177  
[computingFSA\(\)](#) (*amici.amici.Solver* method), 178  
[conservation\\_law\\_has\\_multispecies\(\)](#) (*amici.ode\_export.ODEModel* method), 227  
[ConservationLaw](#) (class in *amici.ode\_export*), 215  
[Constant](#) (class in *amici.ode\_export*), 216  
[constant\\_species\\_to\\_parameters\(\)](#) (in module *amici.petab\_import*), 198  
[constructEdataFromDataFrame\(\)](#) (in module *amici.pandas*), 243  
[count\(\)](#) (*amici.parameter\_mapping.ParameterMapping* method), 249  
[create\\_dummy\\_sbml\(\)](#) (in module *amici.petab\_import\_pysb*), 206  
[create\\_edata\\_for\\_condition\(\)](#) (in module *amici.petab\_objective*), 208  
[create\\_edatas\(\)](#) (in module *amici.petab\_objective*), 208  
[create\\_parameter\\_df\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* method), 203  
[create\\_parameter\\_mapping\(\)](#) (in module *amici.petab\_objective*), 208  
[create\\_parameter\\_mapping\\_for\\_condition\(\)](#) (in module *amici.petab\_objective*), 209  
[create\\_parameterized\\_edatas\(\)](#) (in module *amici.petab\_objective*), 209  
[csc\\_matrix\(\)](#) (in module *amici.ode\_export*), 238  
[CVODES](#), 41

## D

[DAE](#), 41  
[DoubleVector](#) (class in *amici.amici*), 130

## E

[enum\(\)](#) (in module *amici.amici*), 187  
[eq\(\)](#) (*amici.ode\_export.ODEModel* method), 228  
[Event](#) (class in *amici.ode\_export*), 217  
[ExpData](#) (class in *amici.amici*), 130  
[ExpData\(\)](#) (in module *amici*), 127

[ExpDataPtr](#) (class in *amici.amici*), 141  
[ExpDataPtrVector](#) (class in *amici.amici*), 142  
[Expression](#) (class in *amici.ode\_export*), 218  
[extract\\_monomers\(\)](#) (in module *amici.pysb\_import*), 195

## F

[fill\\_in\\_parameters\(\)](#) (in module *amici.parameter\_mapping*), 251  
[fill\\_in\\_parameters\\_for\\_condition\(\)](#) (in module *amici.parameter\_mapping*), 251  
[fixed parameters](#), 41  
[FixedParameterContext](#) (class in *amici.amici*), 142  
[free\\_symbols\(\)](#) (*amici.ode\_export.ODEModel* method), 228  
[from\\_combine\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* static method), 203  
[from\\_files\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* static method), 203  
[from\\_folder\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* static method), 203  
[from\\_yaml\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* static method), 204

## G

[generate\\_basic\\_variables\(\)](#) (*amici.ode\_export.ODEModel* method), 228  
[generate\\_flux\\_symbol\(\)](#) (in module *amici.ode\_export*), 239  
[generate\\_measurement\\_symbol\(\)](#) (in module *amici.ode\_export*), 239  
[generate\\_model\\_code\(\)](#) (*amici.ode\_export.ODEExporter* method), 223  
[get\\_appearance\\_counts\(\)](#) (*amici.ode\_export.ODEModel* method), 228  
[get\\_conservation\\_laws\(\)](#) (*amici.ode\_export.ODEModel* method), 228  
[get\\_dt\(\)](#) (*amici.ode\_export.State* method), 235  
[get\\_fixed\\_parameters\(\)](#) (in module *amici.petab\_import*), 198  
[get\\_free\\_symbols\(\)](#) (*amici.ode\_export.State* method), 235  
[get\\_function\\_extern\\_declaration\(\)](#) (in module *amici.ode\_export*), 239  
[get\\_id\(\)](#) (*amici.ode\_export.ConservationLaw* method), 215  
[get\\_id\(\)](#) (*amici.ode\_export.Constant* method), 216  
[get\\_id\(\)](#) (*amici.ode\_export.Event* method), 218  
[get\\_id\(\)](#) (*amici.ode\_export.Expression* method), 219

`get_id()` (*amici.ode\_export.LogLikelihood method*), 220  
`get_id()` (*amici.ode\_export.ModelQuantity method*), 221  
`get_id()` (*amici.ode\_export.Observable method*), 232  
`get_id()` (*amici.ode\_export.Parameter method*), 233  
`get_id()` (*amici.ode\_export.SigmaY method*), 234  
`get_id()` (*amici.ode\_export.State method*), 236  
`get_lb()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_logger()` (*in module amici.logging*), 246  
`get_measurement_symbol()` (*amici.ode\_export.Observable method*), 232  
`get_model_override_implementation()` (*in module amici.ode\_export*), 239  
`get_model_parameters()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_name()` (*amici.ode\_export.ConservationLaw method*), 215  
`get_name()` (*amici.ode\_export.Constant method*), 216  
`get_name()` (*amici.ode\_export.Event method*), 218  
`get_name()` (*amici.ode\_export.Expression method*), 219  
`get_name()` (*amici.ode\_export.LogLikelihood method*), 220  
`get_name()` (*amici.ode\_export.ModelQuantity method*), 221  
`get_name()` (*amici.ode\_export.Observable method*), 232  
`get_name()` (*amici.ode\_export.Parameter method*), 233  
`get_name()` (*amici.ode\_export.SigmaY method*), 234  
`get_name()` (*amici.ode\_export.State method*), 236  
`get_noise_distributions()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_observable_ids()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_observables()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_observation_model()` (*in module amici.petab\_import*), 198  
`get_optimization_parameter_scales()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_optimization_parameters()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_optimization_to_simulation_parameter_mapping()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 204  
`get_sigmas()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 205  
`get_simulation_conditions_from_measurement_df()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 205  
`get_species_initial()` (*in module amici.sbml\_import*), 194  
`get_sunindex_extern_declaration()` (*in module amici.ode\_export*), 239  
`get_sunindex_override_implementation()` (*in module amici.ode\_export*), 240  
`get_switch_statement()` (*in module amici.ode\_export*), 240  
`get_ub()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 205  
`get_val()` (*amici.ode\_export.ConservationLaw method*), 215  
`get_val()` (*amici.ode\_export.Constant method*), 216  
`get_val()` (*amici.ode\_export.Event method*), 218  
`get_val()` (*amici.ode\_export.Expression method*), 219  
`get_val()` (*amici.ode\_export.LogLikelihood method*), 220  
`get_val()` (*amici.ode\_export.ModelQuantity method*), 221  
`get_val()` (*amici.ode\_export.Observable method*), 232  
`get_val()` (*amici.ode\_export.Parameter method*), 233  
`get_val()` (*amici.ode\_export.SigmaY method*), 234  
`get_val()` (*amici.ode\_export.State method*), 236  
`get_x_ids()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 205  
`get_x_nominal()` (*amici.petab\_import\_pysb.PysbPetabProblem method*), 205  
`getAbsoluteTolerance()` (*amici.amici.Solver method*), 178  
`getAbsoluteToleranceB()` (*amici.amici.Solver method*), 178  
`getAbsoluteToleranceFSA()` (*amici.amici.Solver method*), 178  
`getAbsoluteToleranceQuadratures()` (*amici.amici.Solver method*), 178  
`getAbsoluteToleranceSteadyState()` (*amici.amici.Solver method*), 178  
`getAbsoluteToleranceSteadyStateSensi()` (*amici.amici.Solver method*), 178  
`getAlwaysCheckFinite()` (*amici.amici.Model method*), 149  
`getAmiciCommit()` (*amici.amici.Model method*), 149  
`getAmiciVersion()` (*amici.amici.Model method*), 149  
`getCpuTime()` (*amici.amici.Solver method*), 178

`getCpuTimeB()` (*amici.amici.Solver method*), 178  
`getDataObservablesAsDataFrame()` (in module *amici.pandas*), 243  
`getEdataFromDataFrame()` (in module *amici.pandas*), 244  
`getExpressionIds()` (*amici.amici.Model method*), 149  
`getExpressionNames()` (*amici.amici.Model method*), 149  
`getFixedParameterById()` (*amici.amici.Model method*), 149  
`getFixedParameterByName()` (*amici.amici.Model method*), 149  
`getFixedParameterIds()` (*amici.amici.Model method*), 149  
`getFixedParameterNames()` (*amici.amici.Model method*), 149  
`getFixedParameters()` (*amici.amici.Model method*), 150  
`getInitialStates()` (*amici.amici.Model method*), 150  
`getInitialStateSensitivities()` (*amici.amici.Model method*), 150  
`getInternalSensitivityMethod()` (*amici.amici.Solver method*), 179  
`getInterpolationType()` (*amici.amici.Solver method*), 179  
`getLastOrder()` (*amici.amici.Solver method*), 179  
`getLinearMultistepMethod()` (*amici.amici.Solver method*), 179  
`getLinearSolver()` (*amici.amici.Solver method*), 179  
`getMaxSteps()` (*amici.amici.Solver method*), 179  
`getMaxStepsBackwardProblem()` (*amici.amici.Solver method*), 179  
`getName()` (*amici.amici.Model method*), 150  
`getNewtonDampingFactorLowerBound()` (*amici.amici.Solver method*), 179  
`getNewtonDampingFactorMode()` (*amici.amici.Solver method*), 179  
`getNewtonMaxLinearSteps()` (*amici.amici.Solver method*), 179  
`getNewtonMaxSteps()` (*amici.amici.Solver method*), 180  
`getNonlinearSolverIteration()` (*amici.amici.Solver method*), 180  
`getNumErrTestFails()` (*amici.amici.Solver method*), 180  
`getNumErrTestFailsB()` (*amici.amici.Solver method*), 180  
`getNumNonlinSolvConvFails()` (*amici.amici.Solver method*), 180  
`getNumNonlinSolvConvFailsB()` (*amici.amici.Solver method*), 180  
`getNumRhsEvals()` (*amici.amici.Solver method*), 180  
`getNumRhsEvalsB()` (*amici.amici.Solver method*), 180  
`getNumSteps()` (*amici.amici.Solver method*), 180  
`getNumStepsB()` (*amici.amici.Solver method*), 180  
`getObservableIds()` (*amici.amici.Model method*), 150  
`getObservableNames()` (*amici.amici.Model method*), 150  
`getObservedData()` (*amici.amici.ExpData method*), 136  
`getObservedDataPtr()` (*amici.amici.ExpData method*), 136  
`getObservedDataStdDev()` (*amici.amici.ExpData method*), 136  
`getObservedDataStdDevPtr()` (*amici.amici.ExpData method*), 136  
`getObservedEvents()` (*amici.amici.ExpData method*), 136  
`getObservedEventsPtr()` (*amici.amici.ExpData method*), 136  
`getObservedEventsStdDev()` (*amici.amici.ExpData method*), 136  
`getObservedEventsStdDevPtr()` (*amici.amici.ExpData method*), 137  
`getParameterById()` (*amici.amici.Model method*), 150  
`getParameterByName()` (*amici.amici.Model method*), 150  
`getParameterIds()` (*amici.amici.Model method*), 150  
`getParameterList()` (*amici.amici.Model method*), 150  
`getParameterNames()` (*amici.amici.Model method*), 151  
`getParameters()` (*amici.amici.Model method*), 151  
`getParameterScale()` (*amici.amici.Model method*), 151  
`getPreequilibration()` (*amici.amici.Solver method*), 181  
`getReinitializationStateIdxs()` (*amici.amici.Model method*), 151  
`getReinitializeFixedParameterInitialStates()` (*amici.amici.Model method*), 151  
`getRelativeTolerance()` (*amici.amici.Solver method*), 181  
`getRelativeToleranceB()` (*amici.amici.Solver method*), 181  
`getRelativeToleranceFSA()` (*amici.amici.Solver method*), 181  
`getRelativeToleranceQuadratures()` (*amici.amici.Solver method*), 181  
`getRelativeToleranceSteadyState()` (*am-*

*ici.amici.Solver method*), 181  
 getRelativeToleranceSteadyStateSensi() (*amici.amici.Solver method*), 181  
 getResidualsAsDataFrame() (*in module amici.pandas*), 244  
 getReturnDataReportingMode() (*amici.amici.Solver method*), 181  
 getScaledParameter() (*in module amici.amici*), 187  
 getSensitivityMethod() (*amici.amici.Solver method*), 181  
 getSensitivityMethodPreequilibration() (*amici.amici.Solver method*), 181  
 getSensitivityOrder() (*amici.amici.Solver method*), 182  
 getSimulationObservablesAsDataFrame() (*in module amici.pandas*), 244  
 getSimulationStatesAsDataFrame() (*in module amici.pandas*), 245  
 getSolver() (*amici.amici.Model method*), 151  
 getStabilityLimitFlag() (*amici.amici.Solver method*), 182  
 getStateIds() (*amici.amici.Model method*), 151  
 getStateIsNonNegative() (*amici.amici.Model method*), 151  
 getStateNames() (*amici.amici.Model method*), 151  
 getStateOrdering() (*amici.amici.Solver method*), 182  
 getSteadyStateSensitivityMode() (*amici.amici.Model method*), 152  
 gett() (*amici.amici.Solver method*), 182  
 getTimepoint() (*amici.amici.ExpData method*), 137  
 getTimepoint() (*amici.amici.Model method*), 152  
 getTimepoints() (*amici.amici.ExpData method*), 137  
 getTimepoints() (*amici.amici.Model method*), 152  
 getUnscaledParameter() (*in module amici.amici*), 188  
 getUnscaledParameters() (*amici.amici.Model method*), 152  
 grouper() (*in module amici.sbml\_import*), 194  
 gsl::make\_span(C++ function), 507, 508

## H

has\_fixed\_parameter\_ic() (*in module amici.pysb\_import*), 195  
 hasCustomInitialStates() (*amici.amici.Model method*), 152  
 hasCustomInitialStateSensitivities() (*amici.amici.Model method*), 152  
 hasExpressionIds() (*amici.amici.Model method*), 152  
 hasExpressionNames() (*amici.amici.Model method*), 152

hasFixedParameterIds() (*amici.amici.Model method*), 152  
 hasFixedParameterNames() (*amici.amici.Model method*), 153  
 hasObservableIds() (*amici.amici.Model method*), 153  
 hasObservableNames() (*amici.amici.Model method*), 153  
 hasParameterIds() (*amici.amici.Model method*), 153  
 hasParameterNames() (*amici.amici.Model method*), 153  
 hasQuadraticLLH() (*amici.amici.Model method*), 153  
 hasStateIds() (*amici.amici.Model method*), 153  
 hasStateNames() (*amici.amici.Model method*), 153

## I

IDAS, 41  
 import\_from\_sbml\_importer() (*amici.ode\_export.ODEModel method*), 228  
 import\_model() (*in module amici.petab\_import*), 198  
 import\_model\_module() (*in module amici*), 127  
 import\_model\_pysb() (*in module amici.petab\_import\_pysb*), 206  
 import\_model\_sbml() (*in module amici.petab\_import*), 199  
 import\_petab\_problem() (*in module amici.petab\_import*), 200  
 index() (*amici.parameter\_mapping.ParameterMapping method*), 249  
 InternalSensitivityMethod (class in *amici.amici*), 142  
 InterpolationType (class in *amici.amici*), 142  
 IntVector (class in *amici.amici*), 142  
 is\_assignment\_rule\_target() (*amici.sbml\_import.SbmlImporter method*), 192  
 is\_rate\_rule\_target() (*amici.sbml\_import.SbmlImporter method*), 192  
 is\_valid\_identifier() (*in module amici.ode\_export*), 240  
 isFixedParameterStateReinitializationAllowed() (*amici.amici.Model method*), 153  
 isSetObservedData() (*amici.amici.ExpData method*), 137  
 isSetObservedDataStdDev() (*amici.amici.ExpData method*), 137  
 isSetObservedEvents() (*amici.amici.ExpData method*), 137  
 isSetObservedEventsStdDev() (*amici.amici.ExpData method*), 137



## K

`k()` (*amici.amici.Model method*), 154

## L

`LinearMultistepMethod` (*class in amici.amici*), 143

`LinearSolver` (*class in amici.amici*), 143

`log_execution_time()` (*in module amici.logging*), 246

`LogLikelihood` (*class in amici.ode\_export*), 219

## M

`M_1_PI` (*C macro*), 514

`M_2_PI` (*C macro*), 514

`M_2_SQRTPI` (*C macro*), 514

`M_E` (*C macro*), 514

`M_LN10` (*C macro*), 514

`M_LN2` (*C macro*), 515

`M_LOG10E` (*C macro*), 515

`M_LOG2E` (*C macro*), 515

`M_PI` (*C macro*), 515

`M_PI_2` (*C macro*), 516

`M_PI_4` (*C macro*), 516

`M_SQRT1_2` (*C macro*), 516

`M_SQRT2` (*C macro*), 516

`main()` (*in module amici.petab\_import*), 200

`Model` (*class in amici.amici*), 143

`ModelDimensions` (*class in amici.amici*), 159

`ModelPtr` (*class in amici.amici*), 162

`ModelQuantity` (*class in amici.ode\_export*), 221

module

`amici`, 125

`amici.amici`, 128

`amici.gradient_check`, 246

`amici.import_utils`, 211

`amici.logging`, 245

`amici.ode_export`, 214

`amici.pandas`, 243

`amici.parameter_mapping`, 249

`amici.petab_import`, 197

`amici.petab_import_pysb`, 201

`amici.petab_objective`, 207

`amici.plotting`, 242

`amici.pysb_import`, 195

`amici.sbml_import`, 189

## N

`name()` (*amici.ode\_export.ODEModel method*), 228

`ncl()` (*amici.amici.Model method*), 154

`NewtonDampingFactorMode` (*class in amici.amici*), 162

`nk()` (*amici.amici.Model method*), 154

`nmaxevent()` (*amici.amici.ExpData method*), 138

`nMaxEvent()` (*amici.amici.Model method*), 154

`noise_distribution_to_cost_function()`  
(*in module amici.import\_utils*), 211

`NonlinearSolverIteration` (*class in amici.amici*), 163

`np()` (*amici.amici.Model method*), 154

`nplist()` (*amici.amici.Model method*), 154

`nplist()` (*amici.amici.Solver method*), 182

`nquad()` (*amici.amici.Solver method*), 182

`nt()` (*amici.amici.ExpData method*), 138

`nt()` (*amici.amici.Model method*), 154

`num_cons_law()` (*amici.ode\_export.ODEModel method*), 228

`num_const()` (*amici.ode\_export.ODEModel method*), 228

`num_events()` (*amici.ode\_export.ODEModel method*), 229

`num_expr()` (*amici.ode\_export.ODEModel method*), 229

`num_obs()` (*amici.ode\_export.ODEModel method*), 229

`num_par()` (*amici.ode\_export.ODEModel method*), 229

`num_state_reinits()` (*amici.ode\_export.ODEModel method*), 229

`num_states_rdata()` (*amici.ode\_export.ODEModel method*), 229

`num_states_solver()` (*amici.ode\_export.ODEModel method*), 229

`nx()` (*amici.amici.Solver method*), 182

`nx_reinit()` (*amici.amici.Model method*), 154

`nytrue()` (*amici.amici.ExpData method*), 138

`nztrue()` (*amici.amici.ExpData method*), 138

## O

`Observable` (*class in amici.ode\_export*), 231

`ODE`, 41

`ode_model_from_pysb_importer()` (*in module amici.pysb\_import*), 195

`ODEExporter` (*class in amici.ode\_export*), 222

`ODEModel` (*class in amici.ode\_export*), 224

## P

`Parameter` (*class in amici.ode\_export*), 232

`ParameterMapping` (*class in amici.parameter\_mapping*), 249

`ParameterMappingForCondition` (*class in amici.parameter\_mapping*), 250

`ParameterScaling` (*class in amici.amici*), 163

`parameterScalingFromIntVector()` (*in module amici.amici*), 188

`ParameterScalingVector` (*class in amici.amici*), 163

[parse\\_cli\\_args\(\)](#) (in module *amici.petab\_import*), 200  
[parse\\_events\(\)](#) (*amici.ode\_export.ODEModel* method), 229  
[PETab](#), 41  
[petab\\_noise\\_distributions\\_to\\_amici\(\)](#) (in module *amici.petab\_import*), 200  
[petab\\_scale\\_to\\_amici\\_scale\(\)](#) (in module *amici.petab\_import*), 200  
[petab\\_to\\_amici\\_scale\(\)](#) (in module *amici.parameter\_mapping*), 252  
[plist\(\)](#) (*amici.amici.Model* method), 154  
[plotObservableTrajectories\(\)](#) (in module *amici.plotting*), 242  
[plotStateTrajectories\(\)](#) (in module *amici.plotting*), 242  
[preequilibration](#), 41  
[presimulation](#), 41  
[process\\_conservation\\_laws\(\)](#) (*amici.sbml\_import.SbmlImporter* method), 192  
[PySB](#), 41  
[pysb2amici\(\)](#) (in module *amici.pysb\_import*), 196  
[pysb\\_model\\_from\\_path\(\)](#) (in module *amici.pysb\_import*), 197  
[PysbPetabProblem](#) (class in *amici.petab\_import\_pysb*), 201

## R

[RDataReporting](#) (class in *amici.amici*), 163  
[rdatas\\_to\\_measurement\\_df\(\)](#) (in module *amici.petab\_objective*), 209  
[rdatas\\_to\\_simulation\\_df\(\)](#) (in module *amici.petab\_objective*), 210  
[readSolverSettingsFromHDF5\(\)](#) (in module *amici*), 127  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.ExpData* method), 138  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.SimulationParameters* method), 174  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.ExpData* method), 138  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.SimulationParameters* method), 174  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.ExpData* method), 138  
[reinitializeAllFixedParameterDependentInitialStates\(\)](#) (*amici.amici.SimulationParameters* method), 174  
[remove\\_typedefs\(\)](#) (in module *amici.ode\_export*), 240  
[replace\\_logx\(\)](#) (in module *amici.sbml\_import*), 194

[requireSensitivitiesForAllParameters\(\)](#) (*amici.amici.Model* method), 155  
[ReturnData](#) (class in *amici.amici*), 164  
[ReturnDataPtr](#) (class in *amici.amici*), 169  
[rowvals\(\)](#) (*amici.ode\_export.ODEModel* method), 229  
[runAmiciSimulation\(\)](#) (in module *amici*), 127  
[runAmiciSimulation\(\)](#) (in module *amici.amici*), 188  
[runAmiciSimulations\(\)](#) (in module *amici*), 127  
[runAmiciSimulations\(\)](#) (in module *amici.amici*), 188

## S

[safe\\_substitute\(\)](#) (*amici.ode\_export.TemplateAmici* method), 237  
[sample\\_parameter\\_startpoints\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem* method), 205  
[SBML](#), 41  
[sbml2amici\(\)](#) (*amici.sbml\_import.SbmlImporter* method), 192  
[SbmlImporter](#) (class in *amici.sbml\_import*), 190  
[scale\\_parameter\(\)](#) (in module *amici.parameter\_mapping*), 252  
[scale\\_parameters\\_dict\(\)](#) (in module *amici.parameter\_mapping*), 252  
[scaleParameters\(\)](#) (in module *amici.amici*), 189  
[SecondOrderMode](#) (class in *amici.amici*), 170  
[SensitivityMethod](#) (class in *amici.amici*), 171  
[SensitivityOrder](#) (class in *amici.amici*), 171  
[set\\_conservation\\_law\(\)](#) (*amici.ode\_export.State* method), 236  
[set\\_dt\(\)](#) (*amici.ode\_export.State* method), 236  
[set\\_log\\_level\(\)](#) (in module *amici.logging*), 246  
[set\\_name\(\)](#) (*amici.ode\_export.ODEExporter* method), 224  
[set\\_paths\(\)](#) (*amici.ode\_export.ODEExporter* method), 224  
[set\\_val\(\)](#) (*amici.ode\_export.ConservationLaw* method), 215  
[set\\_val\(\)](#) (*amici.ode\_export.Constant* method), 217  
[set\\_val\(\)](#) (*amici.ode\_export.Event* method), 218  
[set\\_val\(\)](#) (*amici.ode\_export.Expression* method), 219  
[set\\_val\(\)](#) (*amici.ode\_export.LogLikelihood* method), 220  
[set\\_val\(\)](#) (*amici.ode\_export.ModelQuantity* method), 221  
[set\\_val\(\)](#) (*amici.ode\_export.Observable* method), 232  
[set\\_val\(\)](#) (*amici.ode\_export.Parameter* method), 233  
[set\\_val\(\)](#) (*amici.ode\_export.SigmaY* method), 234  
[set\\_val\(\)](#) (*amici.ode\_export.State* method), 236

`setAbsoluteTolerance()` (*amici.amici.Solver method*), 182  
`setAbsoluteToleranceB()` (*amici.amici.Solver method*), 182  
`setAbsoluteToleranceFSA()` (*amici.amici.Solver method*), 183  
`setAbsoluteToleranceQuadratures()` (*amici.amici.Solver method*), 183  
`setAbsoluteToleranceSteadyState()` (*amici.amici.Solver method*), 183  
`setAbsoluteToleranceSteadyStateSensi()` (*amici.amici.Solver method*), 183  
`setAllStatesNonNegative()` (*amici.amici.Model method*), 155  
`setAlwaysCheckFinite()` (*amici.amici.Model method*), 155  
`setFixedParameterById()` (*amici.amici.Model method*), 155  
`setFixedParameterByName()` (*amici.amici.Model method*), 155  
`setFixedParameters()` (*amici.amici.Model method*), 155  
`setFixedParametersByIdRegex()` (*amici.amici.Model method*), 155  
`setFixedParametersByNameRegex()` (*amici.amici.Model method*), 155  
`setInitialStates()` (*amici.amici.Model method*), 156  
`setInitialStateSensitivities()` (*amici.amici.Model method*), 156  
`setInternalSensitivityMethod()` (*amici.amici.Solver method*), 183  
`setInterpolationType()` (*amici.amici.Solver method*), 183  
`setLinearMultistepMethod()` (*amici.amici.Solver method*), 183  
`setLinearSolver()` (*amici.amici.Solver method*), 183  
`setMaxSteps()` (*amici.amici.Solver method*), 183  
`setMaxStepsBackwardProblem()` (*amici.amici.Solver method*), 183  
`setNewtonDampingFactorLowerBound()` (*amici.amici.Solver method*), 184  
`setNewtonDampingFactorMode()` (*amici.amici.Solver method*), 184  
`setNewtonMaxLinearSteps()` (*amici.amici.Solver method*), 184  
`setNewtonMaxSteps()` (*amici.amici.Solver method*), 184  
`setNMaxEvent()` (*amici.amici.Model method*), 156  
`setNonlinearSolverIteration()` (*amici.amici.Solver method*), 184  
`setObservedData()` (*amici.amici.ExpData method*), 139  
`setObservedDataStdDev()` (*amici.amici.ExpData method*), 139  
`setObservedEvents()` (*amici.amici.ExpData method*), 140  
`setObservedEventsStdDev()` (*amici.amici.ExpData method*), 140  
`setParameterById()` (*amici.amici.Model method*), 156  
`setParameterByName()` (*amici.amici.Model method*), 156  
`setParameterList()` (*amici.amici.Model method*), 157  
`setParameters()` (*amici.amici.Model method*), 158  
`setParametersByIdRegex()` (*amici.amici.Model method*), 158  
`setParametersByNameRegex()` (*amici.amici.Model method*), 158  
`setParameterScale()` (*amici.amici.Model method*), 157  
`setPreequilibration()` (*amici.amici.Solver method*), 184  
`setReinitializationStateIdxs()` (*amici.amici.Model method*), 158  
`setReinitializeFixedParameterInitialStates()` (*amici.amici.Model method*), 158  
`setRelativeTolerance()` (*amici.amici.Solver method*), 184  
`setRelativeToleranceB()` (*amici.amici.Solver method*), 184  
`setRelativeToleranceFSA()` (*amici.amici.Solver method*), 185  
`setRelativeToleranceQuadratures()` (*amici.amici.Solver method*), 185  
`setRelativeToleranceSteadyState()` (*amici.amici.Solver method*), 185  
`setRelativeToleranceSteadyStateSensi()` (*amici.amici.Solver method*), 185  
`setReturnDataReportingMode()` (*amici.amici.Solver method*), 185  
`setSensitivityMethod()` (*amici.amici.Solver method*), 185  
`setSensitivityMethodPreequilibration()` (*amici.amici.Solver method*), 185  
`setSensitivityOrder()` (*amici.amici.Solver method*), 185  
`setStabilityLimitFlag()` (*amici.amici.Solver method*), 185  
`setStateIsNonNegative()` (*amici.amici.Model method*), 158  
`setStateOrdering()` (*amici.amici.Solver method*), 185  
`setSteadyStateSensitivityMode()` (*amici.amici.Model method*), 158  
`setT0()` (*amici.amici.Model method*), 158



[setTimepoints\(\)](#) (*amici.amici.ExpData method*), [141](#)  
[setTimepoints\(\)](#) (*amici.amici.Model method*), [159](#)  
[setUnscaledInitialStateSensitivities\(\)](#) (*amici.amici.Model method*), [159](#)  
[show\\_model\\_info\(\)](#) (in module *amici.petab\_import*), [200](#)  
[SigmaY](#) (*class in amici.ode\_export*), [233](#)  
[simulate\\_petab\(\)](#) (in module *amici.petab\_objective*), [210](#)  
[SimulationParameters](#) (*class in amici.amici*), [171](#)  
[smart\\_is\\_zero\\_matrix\(\)](#) (in module *amici.ode\_export*), [240](#)  
[smart\\_jacobian\(\)](#) (in module *amici.ode\_export*), [240](#)  
[smart\\_multiply\(\)](#) (in module *amici.ode\_export*), [241](#)  
[smart\\_subs\(\)](#) (in module *amici.import\_utils*), [213](#)  
[smart\\_subs\\_dict\(\)](#) (in module *amici.import\_utils*), [213](#)  
[Solver](#) (*class in amici.amici*), [175](#)  
[SolverPtr](#) (*class in amici.amici*), [186](#)  
[sparseeq\(\)](#) (*amici.ode\_export.ODEModel method*), [229](#)  
[sparsesym\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[species\\_to\\_parameters\(\)](#) (in module *amici.petab\_import*), [201](#)  
[State](#) (*class in amici.ode\_export*), [235](#)  
[state\\_has\\_conservation\\_law\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[state\\_has\\_fixed\\_parameter\\_initial\\_condition\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[state\\_is\\_constant\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[SteadyStateSensitivityMode](#) (*class in amici.amici*), [186](#)  
[SteadyStateStatus](#) (*class in amici.amici*), [186](#)  
[SteadyStateStatusVector](#) (*class in amici.amici*), [187](#)  
[StringDoubleMap](#) (*class in amici.amici*), [187](#)  
[StringVector](#) (*class in amici.amici*), [187](#)  
[strip\\_pysb\(\)](#) (in module *amici.ode\_export*), [241](#)  
[subset\\_dict\(\)](#) (in module *amici.petab\_objective*), [211](#)  
[substitute\(\)](#) (*amici.ode\_export.TemplateAmici method*), [237](#)  
[SUNDIALS](#), [41](#)  
[SWIG](#), [41](#)  
[switchForwardSensisOff\(\)](#) (*amici.amici.Solver method*), [186](#)  
[sym\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[sym\\_names\(\)](#) (*amici.ode\_export.ODEModel method*), [230](#)  
[sym\\_or\\_eq\(\)](#) (*amici.ode\_export.ODEModel method*), [231](#)  
[symbol\\_with\\_assumptions\(\)](#) (in module *amici.ode\_export*), [241](#)

## T

[t0\(\)](#) (*amici.amici.Model method*), [159](#)  
[TemplateAmici](#) (*class in amici.ode\_export*), [236](#)  
[to\\_files\(\)](#) (*amici.petab\_import\_pysb.PysbPetabProblem method*), [205](#)  
[toposort\\_symbols\(\)](#) (in module *amici.import\_utils*), [213](#)

## U

[unscale\\_parameter\(\)](#) (in module *amici.parameter\_mapping*), [252](#)  
[unscale\\_parameters\\_dict\(\)](#) (in module *amici.parameter\_mapping*), [252](#)  
[unscaleParameters\(\)](#) (in module *amici.amici*), [189](#)

## V

[val\(\)](#) (*amici.ode\_export.ODEModel method*), [231](#)  
[var\\_in\\_function\\_signature\(\)](#) (in module *amici.ode\_export*), [241](#)

## W

[writeSolverSettingsToHDF5\(\)](#) (in module *amici*), [128](#)