# Odes Documentation

*Release 2.7.0*

**B. Malengier**

**Jul 12, 2023**

# Contents


**1 Installation** **3**
- 1.1 Requirements before install . . . 3
- 1.2 Installation . . . 4
- 1.3 Installation of ODES from git checkout . . . 4
- 1.4 Troubleshooting . . . 5
- 1.5 Using Nix . . . 6

**2 Structure of `odes` and User's Guide** **7**
- 2.1 Simple Function Interface (`odeint`) . . . 7
- 2.2 Object Oriented Interface (`ode` and `dae`) . . . 8
- 2.3 Lower-level interfaces . . . 10

**3 Choosing a Solver** **11**
- 3.1 Performance of the Solvers . . . 11

**4 Reporting Bugs, Contributing and Releasing** **13**
- 4.1 Reporting Bugs . . . 13
- 4.2 Getting the code . . . 13
- 4.3 Running the Tests . . . 13
- 4.4 Adding Examples . . . 14
- 4.5 Building the Docs . . . 14
- 4.6 Creating a New Release . . . 14

**5 Citing ODES** **17**

**6 Indices and tables** **19**

The ODES scikit provides access to Ordinary Differential Equation (ODE) solvers and Differential Algebraic Equation (DAE) solvers not included in scipy. A convenience function `scikits.odes.odeint.odeint()` is available for fast and fire and forget integration. Object oriented class solvers `scikits.odes.ode.ode` and `scikits.odes.dae.dae` are available for fine control. Finally, the low levels solvers are also directly exposed for specialised needs.

Detailed API documentation can be found here

Contents:

CHAPTER 1

# Installation

## 1.1 Requirements before install

If you use nix (see below), all dependencies will be installed for you.

If you do not wish to use nix then you will need to do the following.

Before building `odes`, you need to have installed:

- numpy (automatically dealt with if using pip >=10)
- Python header files (`python-dev`/`python3-dev` on Debian/Ubuntu-based distributions, `python-devel` on Fedora)
- C compiler
- Fortran compiler (e.g. gfortran)
- Sundials 5.1.0

In addition, if building from a git checkout, you'll also need Cython.

If using precision type double, it is required that Sundials is built with the BLAS/LAPACK interface enabled, so check the Fortran Settings section.

A typical install if sundials download package is extracted into directory sundials-5.1.0, with LAPACK included is on a *nix system:

```
mkdir build-sundials-5.1.0
cd build-sundials-5.1.0/
cmake -DLAPACK_ENABLE=ON -DSUNDIALS_INDEX_SIZE=64 -DCMAKE_INSTALL_PREFIX=<install_
↪path> ../sundials-5.1.0/
make install
```

where you replace `<install_path>` with the install path you want, default normally is `/usr/local/`.

For extended precision, set `-DLAPACK_ENABLE=OFF` and add `-DSUNDIALS_PRECISION=extended`.

**Warning:** Make sure you use the Fortran compiler as used for your BLAS/LAPACK install!

**Tip:** We recommend using OpenBLAS, which provides a optimised BLAS implementation which widely distributed, and which doesn't need to be recompiled for different CPUs.

## 1.2 Installation

To install `odes`, use:

```
pip install scikits.odes
```

which will download the latest version from PyPI. This will handle the installation of the additional runtime dependencies of `odes`. You should then run the tests to make sure everything is set up correctly.

If you have installed SUNDIALS in a non-standard path (e.g. `/usr/` or `/usr/local/`), you can set `$SUNDIALS_INST` in your environment to the installation prefix of SUNDIALS (i.e. value of `<install_path>` mentioned above).

### 1.2.1 Testing your version of `odes`

To test the version in python, use in the python shell:

```
>>> import pkg_resources
>>> pkg_resources.get_distribution("scikits.odes").version
```

### 1.2.2 Running the Tests

You need nose to run the tests. To install nose, run:

```
pip install nose
```

To run the tests, in the python shell:

```
>>> import scikits.odes as od; od.test()
```

Note that the sundials library must be in your `LD_LIBRARY_PATH`. So, make sure the directory `$SUNDIALS_INST/lib` is included. You can do this for example as follows (assuming sundials was installed in `/usr/local`:

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

## 1.3 Installation of ODES from git checkout

You can copy the git repository locally in directory odes with:

```
git clone git://github.com/bmcage/odes.git odes
```

Inside the `odes` directory, run:

```
pip install .
```

which will install the checked out version of `odes`. The same environment variables mentioned above can be used to control installation options.

Alternatively, you can inside the `odes` directory the following command to reach the same result (as root, use `sudo` on linux distribution that support it):

```
DISTUTILS_DEBUG=1  python3 setup.py install
```

**Note:** If you try to run the tests whilst in the `odes` directory, Python will pick up the source directory, and not the built version. Move to a different directory when running the tests.

## 1.4 Troubleshooting

### 1.4.1 LAPACK Not Found

Most issues with using `odes` are due to incorrectly setting the LAPACK libraries, resulting in error, typically:

```
AttributeError: module 'scikits.odes.sundials.cvode' has no attribute 'CVODE'
```

or:

```
undefined reference to dcopy_
```

This is an indication `odes` does not link correctly to the LAPACK directories. You can solve this as follows: When installing sundials, look at output of cmake. If it has:

```
-- A library with BLAS API not found. Please specify library location.
-- LAPACK requires BLAS
-- A library with LAPACK API not found. Please specify library location.
```

then `odes` will not work. First make sure you install sundials with BLAS and LAPACK found. On Debian/Ubuntu one needs `sudo apt-get install libopenblas-dev liblapack-dev` Once installed correctly, the sundials cmake output should be:

```
-- A library with BLAS API found.
-- Looking for Fortran cheev
-- Looking for Fortran cheev - found
-- A library with LAPACK API found.
-- Looking for LAPACK libraries... OK
-- Checking if Lapack works... OK
```

You can check the CMakeCache.txt file to see which libraries are found. It should have output similar to:

```
//Blas and Lapack libraries
LAPACK_LIBRARIES:STRING=/usr/lib/liblapack.so;/usr/lib/libf77blas.so;/usr/lib/
↪libatlas.so
//Path to a library.
LAPACK_lapack_LIBRARY:FILEPATH=/usr/lib/liblapack.so
```

With above output, you can set the LAPACK directories and libs correctly. To force `odes` to find these directories you can set them by force by editing the file `scikits/odes/sundials/setup.py`, and passing the directories and libs as used by sundials:

```
INCL_DIRS_LAPACK = ['/usr/include', '/usr/include/atlas']
LIB_DIRS_LAPACK  = ['/usr/lib']
LIBS_LAPACK      = ['lapack', 'f77blas', 'atlas']
```

Note that on your install, these directories and libs might be different than the example above! With these variables set, installation of `odes` should be successful.

### 1.4.2 Linking Errors

Verify you link to the correct sundials version. Easiest to ensure you only have one `libsundials_xxx` installed. If several are installed, pass the correct one via the `$SUNDIALS_INST` environment variable.

## 1.5 Using Nix

By using the Nix package manager, you can install scikits-odes in one line. Of course you need to install nix first:

```
curl https://nixos.org/nix/install | sh
```

And now you can start a python shell with scikits-odes (and numpy) ready for use:

```
nix-shell \
-p python37Packages.scikits-odes \
-p python37Packages.numpy \
--run "python3"
```

You can verify that lapack is available (although the nix install will have run many tests to check this already), try the following python snippet in the interpreter:

```
import numpy as np
from scikits.odes.odeint import odeint

tout = np.linspace(0, 1)
initial_values = np.array([1])

def right_hand_side(t, y, ydot):
  ydot[0] = y[0]

output = odeint(right_hand_side, tout, initial_values,linsolver='lapackdense')
print(output.values.y)
```

You'll probably want to write a `shell.nix` or similar for your project but you should refer to the nix documentation for this.

CHAPTER 2

# Structure of odes and User's Guide

There are a number of different ways of using odes to solve a system of ODEs/DAEs:

- scikits.odes.ode.ode and scikits.odes.dae.dae classes, which provides an object oriented interface and significant amount of control of the solver.
- scikits.odes.odeint.odeint(), a single function alternative to the object oriented interface.
- Accessing the lower-level solver-specific wrappers, such as the modules in scikits.odes.sundials.

In general, a user supplies a function with the signature:

```
right_hand_side(t: float, y: Array[float], ydot: Array[float]) -> int
```

for the ODE solvers, and:

```
right_hand_side(t: float, y: Array[float], ydot: Array[float], residue: Array[float])␣
↪-> int
```

for the DAE solvers, as well as positions to integrate between and initial values.

## 2.1 Simple Function Interface (odeint)

The simplest user program using the odeint interface, assuming you have implemented the ODE right_hand_side mentioned above, is:

```
import numpy as np
from scikits.odes.odeint import odeint

tout = np.linspace(0, 1)
initial_values = np.array([0])

def right_hand_side(t, y, ydot):
    """
```

(continues on next page)

(continued from previous page)

```
    User's right hand side function
    """
    pass

output = odeint(right_hand_side, tout, initial_values)
print(output.values.y)
```

By default, CVODE's BDF method is used, however a different method can be specified via the `method` keyword. Methods specific to `odeint`, which use the recommended setting for the individual solvers, are:

**bdf** CVODE's BDF method (default)

**admo** CVODE's Adams-Moulton method

**rk5** dopri5 Runge-Kutta method of order (4)5

**rk8** dop853 Runge-Kutta method of order 8(5,3)

**beuler** Implicit/Backward Euler method (for educational purposes only!)

**trapz** Trapezoidal Rule method (for educational purposes only!)

A specific solver (e.g. CVODE) can also be passed in via `method`, in the same way specified by the Object Oriented Interface. Solver specific options can be passed in via additional keyword arguments.

## 2.2 Object Oriented Interface (`ode` and `dae`)

The object oriented interfaces for `ode` and `dae` are almost identical, with solver customisations via either keyword arguments or via a `set_options` method, repeated usage of the same solver via the `solve` method, and individual stepping via the `step` method.

**Note:** `odes` 2.2.2 and later have a new output format, which provides access to more solver information. In a future release, the default will be the new output format. To use the new output format, pass as a keyword argument `old_api=False`.

### 2.2.1 `ode` Object Oriented Interface

The simplest user program using the `ode` interface, assuming you have implemented the ODE `right_hand_side` mentioned above, is:

```
import numpy as np
from scikits.odes.ode import ode

SOLVER = 'cvode'
tout = np.linspace(0, 1)
initial_values = np.array([0])
extra_options = {'old_api': False}

def right_hand_side(t, y, ydot):
    """
    User's right hand side function
    """
    pass
```

(continues on next page)

(continued from previous page)

```
ode_solver = ode(SOLVER, right_hand_side, **extra_options)
output = ode_solver.solve(tout, initial_values)
print(output.values.y)
```

Extra options are solver specific, but there is usually support for passing in user data (passed as additional arguments to the provided `right_hand_side`), and for setting the tolerance of the solver. See *Choosing a Solver* for more information about individual solvers.

#### Examples

There are a number of `ode` examples showing different features, including solver specific features. Here are some of them:

- https://github.com/bmcage/odes/blob/master/ipython_examples/Simple%20Oscillator.ipynb

### 2.2.2 `dae` Object Oriented Interface

The simplest user program using the `dae` interface, assuming you have implemented the DAE `right_hand_side` mentioned above, is:

```
import numpy as np
from scikits.odes.dae import dae

SOLVER = 'ida'
tout = np.linspace(0, 1)
y_initial = np.array([0])
ydot_initial = np.array([0])
extra_options = {'old_api': False}

def right_hand_side(t, y, ydot, residue):
    """
    User's right hand side function
    """
    pass

dae_solver = dae(SOLVER, right_hand_side, **extra_options)
output = dae_solver.solve(tout, y_initial, ydot_initial)
print(output.values.y)
```

Extra options are solver specific, but there is usually support for passing in user data (passed as additional arguments to the provided `right_hand_side`), and for setting the tolerance of the solver. See *Choosing a Solver* for more information about individual solvers.

#### Examples

There are a number of `dae` examples showing different features, including solver specific features. Here are some of them:

- https://github.com/bmcage/odes/blob/master/ipython_examples/Double%20Pendulum%20as%20DAE%20with%20roots.ipynb
- https://github.com/bmcage/odes/blob/master/ipython_examples/Planar%20Pendulum%20as%20DAE.ipynb

## 2.3 Lower-level interfaces

Using the lower-level interfaces is solver-specific, see the API docs for more information and *Choosing a Solver* for comparisons between solvers.

CHAPTER 3

# Choosing a Solver

`odes` interfaces with a number of different solvers:

**CVODE** ODE solver with BDF linear multistep method for stiff problems and Adams-Moulton linear multistep method for nonstiff problems. Supports modern features such as: root (event) finding, error control, and (Krylov-)preconditioning. See `scikits.odes.sundials.cvode` for more details and solver specific arguments. Part of SUNDIALS, it is a replacement for the earlier `vode`/`dvode`.

**IDA** DAE solver with BDF linear multistep method for stiff problems and Adams-Moulton linear multistep method for nonstiff problems. Supports modern features such as: root (event) finding, error control, and (Krylov-)preconditioning. See `scikits.odes.sundials.ida` for more details and solver specific arguments. Part of SUNDIALS.

**dopri5** Part of `scipy.integrate`, explicit Runge-Kutta method of order (4)5 with stepsize control.

**dop853** Part of `scipy.integrate`, explicit Runge-Kutta method of order 8(5,3) with stepsize control.

`odes` also includes for comparison reasons the historical solvers:

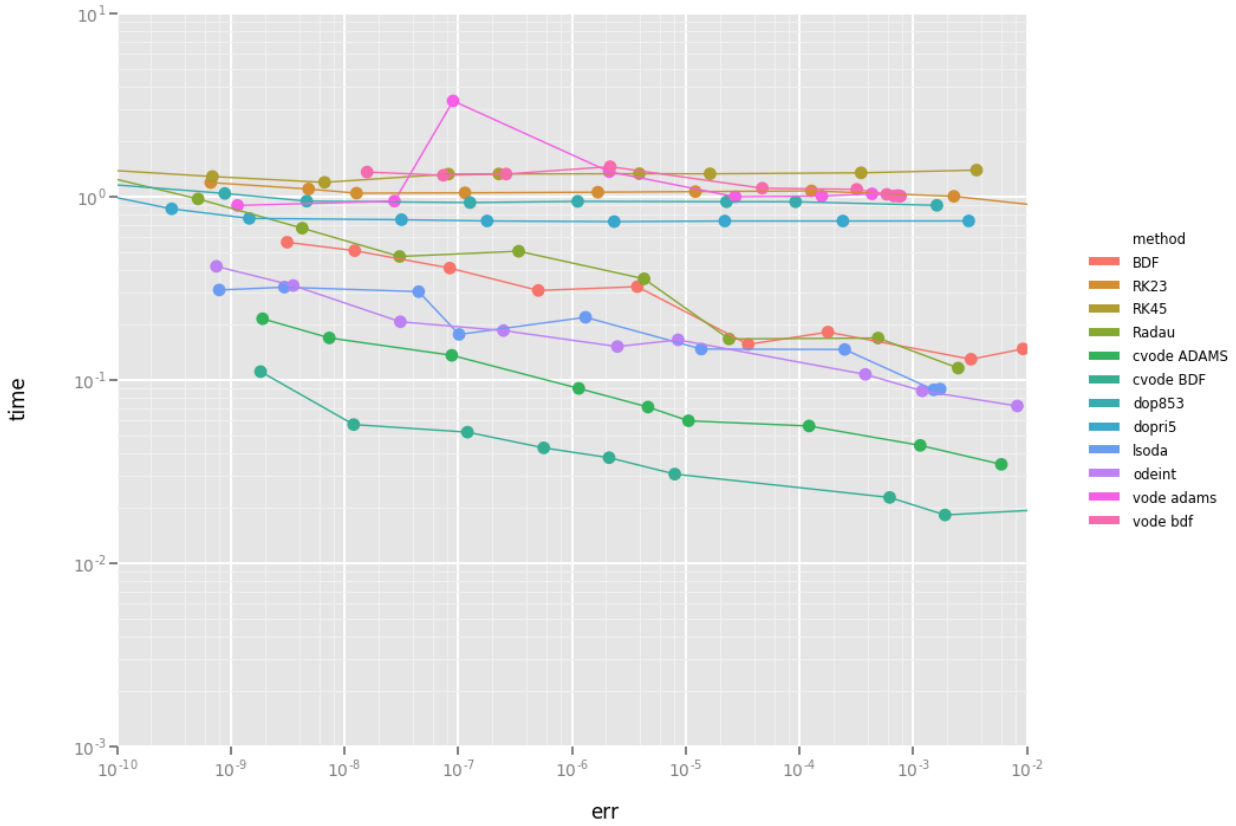
**lsodi** Part of odepack, IDA should be used instead of this. See `scikits.odes.lsodiint` for more details.

**ddaspk** Part of daspk, IDA should be used instead of this. See `scikits.odes.ddaspkint` for more details.

Support for other SUNDIALS solvers (e.g. ARKODE) is currently not implemented, nor is support for non-serial methods (e.g. MPI, OpenMP). Contributions adding support new SUNDIALS solvers or features is welcome.

## 3.1 Performance of the Solvers

A comparison of different methods is given in following image. In this BDF, RK23, RK45 and Radau are python implementations; cvode is the CVODE interface included in `odes`; lsoda, odeint and vode are the scipy integrators (2016), dopri5 and dop853 are the Runge-Kutta methods in scipy. For this problem, cvode performs fastest at a preset tolerance.




You can generate above graph via the Performance notebook.

CHAPTER 4

# Reporting Bugs, Contributing and Releasing

We welcome contributions, whether as bug reports, improvements to the code, or more examples.

Please note that all contributions are subject to our code of conduct.

## 4.1 Reporting Bugs

`odes` bug tracker is on GitHub.

When reporting bugs, please include the versions of Python, `odes` and SUNDIALS, as well as which OS this appears on.

## 4.2 Getting the code

The primary repository is at https://github.com/bmcage/odes, and it is the repository that pull requests should be made against.

Work should be done in a private branch based on master, with pull requests made against master.

## 4.3 Running the Tests

`odes` uses tox to manage testing across different versions.

To install tox, use:

```
pip install tox
```

and to run the tests, inside the top level of the repository, run:

```
tox
```

## 4.4 Adding Examples

Examples should be added in the `examples` folder.

### 4.4.1 Adding ipython/jupyter notebook examples

Please submit extra jupyter notebook examples of usage of `odes`. Example notebooks should go in `ipython_examples`, and add a short description to `ipython_examples/README.md`.

## 4.5 Building the Docs

The documentation for `odes` is split into two parts, the main docs (of which this is a part), and the API docs. Both the main docs and API docs use sphinx to build the docs, and running `make html` inside either of the associated directories will cause sphinx to create a html version of the docs.

The main docs are located in the `docs` directory, and the requirements for building it are in `docs/requirements.txt`.

The API docs are located in the `apidocs` directory, and the requirements for building it are in `apidocs/requirements.txt`.

## 4.6 Creating a New Release

There are five steps to creating a new `odes` release:

1. Make a non-development version.
2. Create a new release on GitHub.
3. Publish the new release on Zenodo.
4. Upload the new release to PyPI.
5. Bump the version to the next development version.

The main docs should automatically build on readthedocs, and the API docs should be built by doctr. You should check that the docs have updated once you have make the release. If docs are not updated automatically, login to readthedocs go to scikits, builds, and build latest and master manually.

### 4.6.1 Making a non-development version

To make a non-development version, inside `common.py` change `DEV=True` to `DEV=False`, and if needed, modify `MAJOR`, `MINOR` and `MICRO` to set the new release version. Then commit only these changes and push them to the main repository (bmcage/odes).

### 4.6.2 Creating a new release on GitHub

On GitHub, draft a new release by clicking the appropriate button. Use the version number from the non-development commit as the title, and hit release. This will upload the release for a DOI to Zenodo as draft.

### 4.6.3 Publishing the new release on Zenodo

Go to uploads in Zenodo, edit the uploaded new release, adding addition information as needed such as ORCID, save and hit the publish button. This will generate a DOI.

### 4.6.4 Uploading the new release to PyPI

Make sure the current checkout is the non-development commit. To make sure no additional changes are included in the release, run:

```
git stash save --no-keep-index --all
```

This saves the current working directory, then cleans it. The changes can be retrieved by running `git stash pop` (but you should not do this until the end).

In the cleaned repository, run:

```
python setup.py sdist --formats=gztar
```

which creates a `dist` directory containing a `tar.gz` file, the sdist for the release. To upload the sdist to PyPI, run:

```
python3 -m pip install --user --upgrade twine
python3 -m twine upload --repository-url https://upload.pypi.org/legacy/ dist/*
```

See https://packaging.python.org/tutorials/distributing-packages/#uploading-your-project-to-pypi for more information about uploading to PyPI.

### 4.6.5 Bumping the version to the next development version

Modify `MAJOR`, `MINOR` and `MICRO` in `common.py` to a later version (increasing `MICRO` by 1 is sufficient). Also in `common.py`, change back to `DEV=True`. Finally, copy the DOI badge of of the latest release from Zenodo to the `README.md`, and commit only these two files. You can now run `git stash pop` to retrieve what you were working on.

### 4.6.6 Build documentation on ReadTheDocs

Login on readthedocs, as release manager you should have access to project `scikits-odes`. Update the build documentation: latest, new release, master. On new bugfix point release, deactivate the previous point release docs.

# CHAPTER 5

# Citing ODES

If you use ODES as part of your research, can you please cite the ODES JOSS paper. A bibtex entry for the paper is below:

```
@article{ODES,
  doi = {10.21105/joss.00165},
  url = {https://doi.org/10.21105/joss.00165},
  year  = {2018},
  month = {feb},
  publisher = {The Open Journal},
  volume = {3},
  number = {22},
  pages = {165},
  author = {Benny Malengier and Pavol Ki{\v{s}}on and James Tocknell and Claas Abert␣
↪and Florian Bruckner and Marc-Antonio Bisotti},
  title = {{ODES}: a high level interface to {ODE} and {DAE} solvers},
  journal = {The Journal of Open Source Software}
}
```

Individual releases have DOIs on Zenodo, which you can cite in addition to the JOSS paper. Additionally, if you use one of the SUNDIALS solvers, we strongly encourage you to cite the SUNDIALS papers. Finally, ODES is built upon the hard work of the numpy, scipy and Cython developers, we strongly encourage you to cite them also.

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search