Odes Documentation

Release 2.3.2.dev0

B. Malengier

Jan 09, 2018

Contents

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

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 2.7.0

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

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-2.7.0 is on a *nix system:

```
mkdir build-sundials-2.7.0
cd build-sundials-2.7.0/
cmake -DLAPACK_ENABLE=ON -DCMAKE_INSTALL_PREFIX=<install_path> ../sundials-2.7.0/
make install
```

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 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()
```

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.

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.

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):
    """
    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

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:

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:

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:

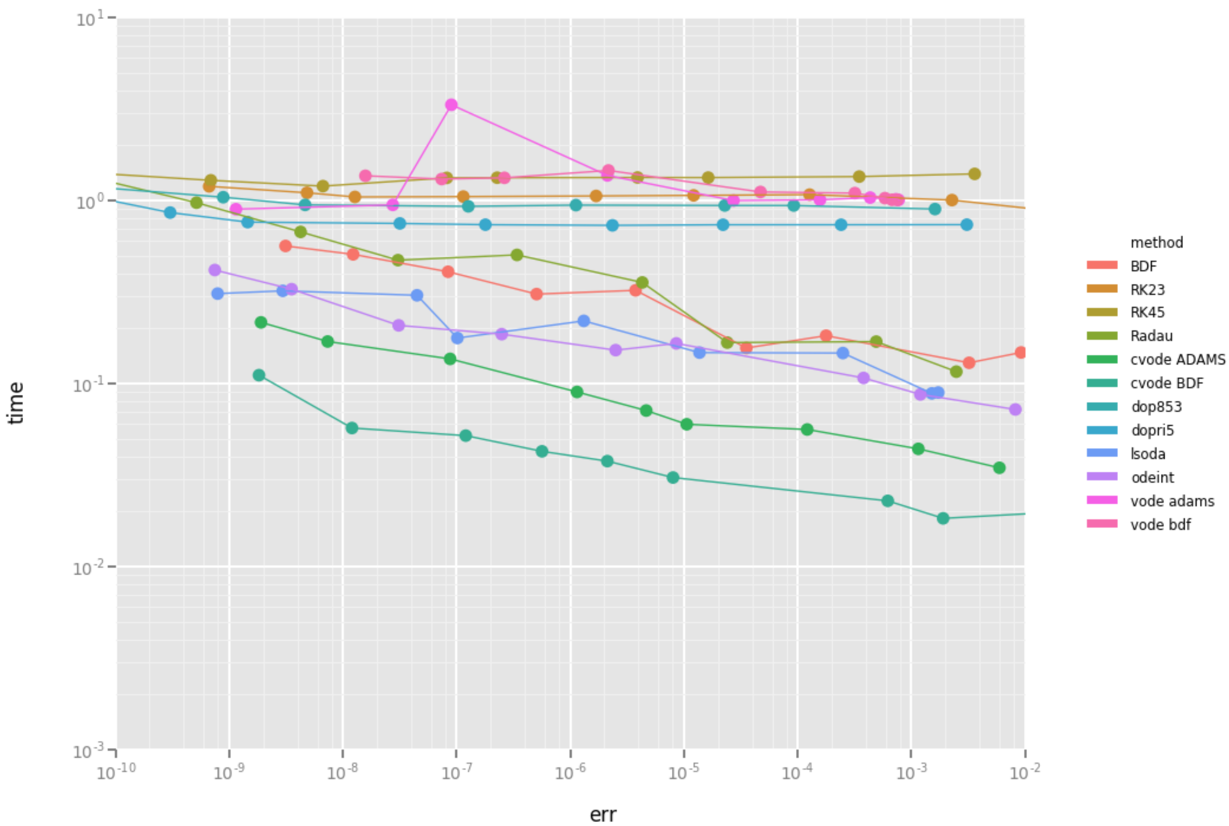
lsodi Part of odepack, IDA should be used instead of this. See *scikits.odes.lsodiint* for more details.

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 Creating a New Release

1. Set in `common.py` version string and `DEV=False`, commit this.
2. On GitHub, draft a new release by clicking the appropriate button. Give correct version number, and hit release. This will upload the release for a DOI to Zenodo as draft.
3. Go to uploads in Zenodo, edit the uploaded new release, save and hit the publish button. This will generate a DOI.
4. Update to PyPI: `python setup.py sdist --formats=gztar register upload`
5. Update version string to a higher number in `common.py`, and `DEV=True`, next copy the DOI badge of Zenodo in the `README.md`, commit these two files.

For the documentation, you need following packages:

```
sudo apt-get install python-sphinx python-numpydoc python-mock
```

After local install, create the new documentation via

1. Go to the sphinx directory: `cd sphinxdoc`
2. Create the documentation: `make html`
3. Upload the new html doc.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search