Odes Documentation

Release 2.2.2

B. Malengier

Mar 15, 2017

Contents

The ODES scikit provides access to Ordinary Differential Equation (ODE) solvers and Differential Algebraic Equation (DAE) solvers.

A convenience function *odeint* is available for fast and fire and forget integration. Object oriented class solvers *ode* and *dae* are available for fine control. Finally, the low levels solvers are also directly exposed for specialised needs.

Contents:

CHAPTER 1

odeint Convenience function

scikits.odes.odeint.**odeint**(*rhsfun*, *tout*, *y0*, *method='bdf'*, ***options*)

Integrate a system of ordinary differential equations. *odeint* is a wrapper around the ode class, as a confenience function to quickly integrate a system of ode. Solves the initial value problem for stiff or non-stiff systems of first order ode's:

rhs = dy/dt = fun(t, y)

where y can be a vector, then rhsfun must be a function computing rhs with signature:

rhsfun(t, y, rhs)

storing the computated dy/dt in the rhs array passed to the function

Parameters

- **rhsfun** (*callable(t, y, out)*) – Computes the derivative at dy/dt in terms of t and y, and stores in out
- **y0** (*array*) – Initial condition on y (can be a vector).
- **t** (*array*) – A sequence of time points for which to solve for y. The initial value point should be the first element of this sequence.
- **method** (*string, solution method to use.*) – Available are all the ode class solvers as well as some convenience shorthands:

Method	Meaning
bdf	This uses the 'cvode' solver in default from, which is a variable step, variable coefficient Backward Differentiation Formula solver, good for stiff ODE. Newton iterations are used to solve the nonlinear system.
admo	This uses the 'cvode' solver with option lmm_type='ADAMS', which is a variable step Adams-Moulton method (linear multistep method), good for non-stiff ODE. Functional iterations are used to solve the nonlinear system.
rk5	This uses the 'dopri5' solver, which is a variable step Runge-Kutta method of order (4)5 (use for non-stiff ODE)
rk8	This uses the 'dop853' solver, which is a variable step Runge-Kutta method of order 8(5,3)

For educational purposes, you can also access following methods:

Method	Meaning
beuler	This is the Implicit Euler (backward Euler) method (order 1), which is obtained via the 'bdf' method, setting the order option to 1, setting large tolerances, and fixing the stepsize. Use option 'step' to change stepsize, default: step=0.05. Use option 'rtol' and 'atol' to use more strict tolerances Note: this is not completely the backward Euler method, as the cvode solver has added control options!
trapz	This is the Trapezoidal Rule method (order 2), which is obtained via the 'admo' method, setting option order to 2, setting large tolerances and fixing the stepsize. Use option 'step' to change stepsize, default: step=0.05. Use option 'rtol' and 'atol' to use more strict tolerances Note: The cvode solver might change the order to 1 internally in which case this becomes beuler method. Set atol, rtol options as strict as possible.

You can also access the solvers of ode via their names:

Method	Meaning
cvode	This uses the 'cvode' solver
dopri5	This uses the 'dopri5' solver
dop853	This uses the 'dop853' solver

- **options** (*extra solver options, optional*) – Every solver has it's own extra options, see the ode class and the details of the solvers available there to know the options possible per solver

Returns

solution – A single named tuple is returned containing the result of the integration.

Field	Meaning
flag	An integer flag
values	Named tuple with fields t and y
errors	Named tuple with fields t and y
roots	Named tuple with fields t and y
tstop	Named tuple with fields t and y
message	String with message in case of an error

Return type named tuple

See also:

ode() a more object-oriented integrator in scikits.odes

dae() a solver for differential-algebraic equations in scikits.odes

quad() for finding the area under a curve, part of scipy.

Examples

The second order differential equation for the angle *theta* of a pendulum acted on by gravity with friction can be written:

$$\theta''(t) + b\theta'(t) + c\sin(\theta(t)) = 0$$

where b and c are positive constants, and a prime (') denotes a derivative. To solve this equation with *odeint*, we must first convert it to a system of first order equations. By defining the angular velocity `omega(t) =`

`theta'(t)`, we obtain the system:

$$\theta'(t) = \omega(t)\omega'(t) = -b\omega(t) - c\sin(\theta(t))$$

We assume the constants are $b = 0.25$ and $c = 5.0$:

```
>>> b = 0.25
>>> c = 5.0
```

Let *y* be the vector [*theta*, *omega*]. We implement this system in python as:

```
>>> def pend(t, y, out):
 ...     theta, omega = y
 ...     out[:] = [omega, -b*omega - c*np.sin(theta)]
 ...
```

In case you want b and c easily changable, make pend a class method, and consider attributes b and c accessible via *self.b* and *self.c*. For initial conditions, we assume the pendulum is nearly vertical with *theta(0)* = *pi* - 0.1, and it initially at rest, so *omega(0)* = 0. Then the vector of initial conditions is

```
>>> y0 = [np.pi - 0.1, 0.0]
```

We generate a solution 101 evenly spaced samples in the interval 0 <= *t* <= 10. So our array of times is

```
>>> t = np.linspace(0, 10, 101)
```

Call *odeint* to generate the solution.

```
>>> from scikits.odes.odeint import odeint
>>> sol = odeint(pend, t, y0)
```

The solution is a named tuple *sol*. sol.values.y is an array with shape (101, 2). The first column is *theta(t)*, and the second is *omega(t)*. The following code plots both components.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(sol.values.t, sol.values.y[:, 0], 'b', label='theta(t)')
>>> plt.plot(sol.values.t, sol.values.y[:, 1], 'g', label='omega(t)')
>>> plt.legend(loc='best')
>>> plt.xlabel('t')
>>> plt.grid()
>>> plt.show()
```

CHAPTER 2

scikits.odes.ode Class

class scikits.odes.ode.**ode**(*integrator_name*, *eqsrhs*, ***options*)

A generic interface class to differential equation solvers.

See also:

scikits.odes.odeint.odeint an ODE integrator with a simpler interface

scipy.integrate Methods in scipy for ODE integration

Examples

ODE arise in many applications of dynamical systems, as well as in discritisations of PDE (eg moving mesh combined with method of lines). As an easy example, consider the simple oscillator,

```
>>> from __future__ import print_function
>>> from numpy import cos, sin, sqrt
>>> k = 4.0
>>> m = 1.0
>>> initx = [1, 0.1]
>>> def rhseqn(t, x, xdot):
        # we create rhs equations for the problem
        xdot[0] = x[1]
        xdot[1] = - k/m * x[0]
```

```
>>> from scikits.odes import ode
>>> solver = ode('cvode', rhseqn, old_api=False)
>>> result = solver.solve([0., 1., 2.], initx)
>>> print('   t        Solution          Exact')
>>> print('-------------------------------------')
>>> for t, u in zip(result.values.t, result.values.y):
        print('%4.2f %15.6g %15.6g' % (t, u[0], initx[0]*cos(sqrt(k/
↪m)*t)+initx[1]*sin(sqrt(k/m)*t)/sqrt(k/m)))
```

More examples in the Examples directory and IPython worksheets.

Available integrators:

cvode

dopri5

dop853

__init__(*integrator_name*, *eqsrhs*, ***options*)
Initialize the ODE Solver and it's options.

$$\frac{dy(t)}{dt} = f(t, y(t)), \quad y(t_0) = y_0$$

$$y(t_0)[i] = y_0[i], i = 0, ..., \mathrm{len}(y_0) - 1$$

f(t,y) is the right hand side function and returns a vector of size $\mathrm{len}(y_0)$.

Parameters

- **integrator_name** (*name of the integrator solver to use.*) – Currently you can choose *cvode*, *dopri5* and *dop853*.
- **eqsrhs** (*right-hand-side function*) – Right-hand-side of a first order ode. Generally, you can assume the following signature to work:

  eqsrhs(x, y, return_rhs)

  with

  x: independent variable, eg the time, float

  y: array of n unknowns in x

  return_rhs : array that must be updated with the value of the right-hand-side, so f(t,y). The dimension is equal to dim(y)

  return value: An integer, 0 for success, 1 for failure. It is not guaranteed that a solver takes this status into account

  Some solvers will allow userdata to be passed to eqsrhs, or optional formats that are more performant.
- **options** (*additional options of the solver*) – See set_options method of the *integrator_name* you selected for details. Set option *old_api=False* to use the new API. In the future, this will become the default!

__weakref__
list of weak references to the object (if defined)

get_info()
Return additional information about the state of the integrator.

Returns

- *A dictionary filled with internal data as exposed by the integrator.*
- See the *get_info* method of your chosen integrator for details.

init_step(*t0*, *y0*)
Initializes the solver and allocates memory. It is not needed to call this method if solve is used to compute the solution. In the case step is used, init_step must be called first.

Parameters

- **t0** (*initial time*) –
- **y0** (*initial condition for y (can be list or numpy array)*) –

Returns

- *if old_api* – flag - boolean status of the computation (successful or error occured)

  t_out - inititial time

- *if old_api False* –

  A named tuple, with fields: flag = An integer flag (StatusEnum)

  values = Named tuple with entries t and y

  errors = Named tuple with entries t and y

  roots = Named tuple with entries t and y

  tstop = Named tuple with entries t and y

  message= String with message in case of an error

set_options (***options*)
Set specific options for the solver. See the solver documentation for details.

Calling set_options a second time, is only possible for options that can change during runtime.

set_tstop (*tstop*)
Add a stop time to the integrator past which he is not allowed to integrate.

Parameters **tstop** (*float time*) – Time point in the future where the integration must stop. You can indicate like this that integration past this point is not allowed, in order to avoid undefined behavior. You can achieve the same result with a call to *set_options(tstop=tstop)*

solve (*tspan*, *y0*)
Runs the solver.

Parameters

- **tspan** (*a list/array of times at which the computed value will be returned. Must contain the start time.*) –
- **y0** (*list/numpy array of initial values*) –

Returns

- *if old_api* – flag - indicating return status of the solver

  t - numpy array of times at which the computations were successful

  y - numpy array of values corresponding to times t (values of y[i, :] ~ t[i])

  t_err - float or None - if recoverable error occured (for example reached maximum number of allowed iterations), this is the time at which it happened

  y_err - numpy array of values corresponding to time t_err

- *if old_api False* –

  A named tuple, with fields: flag = An integer flag (StatusEnum)

  values = Named tuple with entries t and y

  errors = Named tuple with entries t and y

  roots = Named tuple with entries t and y

tstop = Named tuple with entries t and y

message= String with message in case of an error

step(*t*, *y_retn=None*)

Method for calling successive next step of the ODE solver to allow more precise control over the solver. The 'init_step' method has to be called before the 'step' method.

Parameters **t** (*A step is done towards time t, and output at t returned.*) – This time can be higher or lower than the previous time. If option 'one_step_compute'==True, and the solver supports it, only one internal solver step is done in the direction of t starting at the current step.

If old_api=True, the old behavior is used:

if t>0.0 then integration is performed until this time and results at this time are returned in y_retn

if t<0.0 only one internal step is perfomed towards time abs(t) and results after this one time step are returned

Returns

- *if old_api* – flag - status of the computation (successful or error occured)

  t_out - time, where the solver stopped (when no error occured, t_out == t)

- *if old_api False* –

  A named tuple, with fields: flag = An integer flag (StatusEnum)

  values = Named tuple with entries t and y

  errors = Named tuple with entries t and y

  roots = Named tuple with entries t and y

  tstop = Named tuple with entries t and y

  message= String with message in case of an error

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

G

I

O

S