Exhibiting open source numerical software packages

ODE solvers in Python

Eduard Campillo-Funollet

UK-APASI, 16/03/2021

Overview

Example model

Euler's method

Runge-Kutta methods

Implicit methods

Object oriented solver

Example model

Pendulum

Model:

$$\begin{cases} \theta'' = -k\sin(\theta), \\ \theta(0) = \theta_0, \quad \theta'(0) = \theta_0'. \end{cases}$$

Energy:

$$E(\theta) = \frac{1}{2}\theta'^2 + k(1 - \cos(\theta)).$$

First order system:

$$\begin{cases} y_{[0]}' = y_{[1]}, \\ y_{[1]}' = -k\sin(y_{[0]}), \\ y_{[0]}(0) = \theta_0, \quad y_{[1]}(0) = \theta_0'. \end{cases}$$

```
def f(t,y,k=1.):
  return np.array([y[1],
    -k*np.sin(y[0])])

def Df(t,y,k=1.):
  return np.array([ [ 0,1 ],
[-np.cos(y[0]), 0 ] ])

def E(y,k=1.):
  return 0.5*y[:,1]**2
    + k*(1 - np.cos(y[:,0]))
```

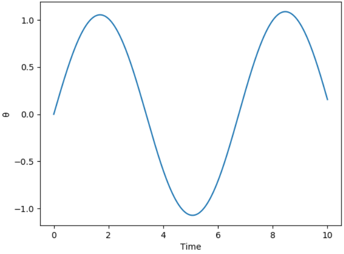
Euler's method

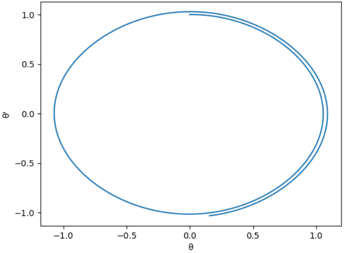
Euler's method

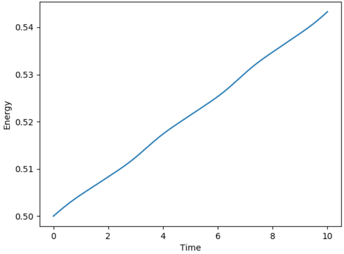
$$y^{(n+1)} = y^{(n)} + hf(t, y^{(n)})$$

```
T = np.linspace(t0,t1,nsteps)
h = T[1]-T[0]
Y = [ y0 ]
for t in T[1:]:
  Y.append(Y[-1]+h*f(t,Y[-1],k))
```

Euler's method - results







Runge-Kutta methods

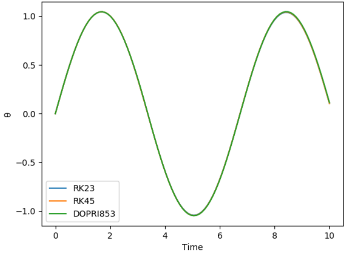
Runge-Kutta

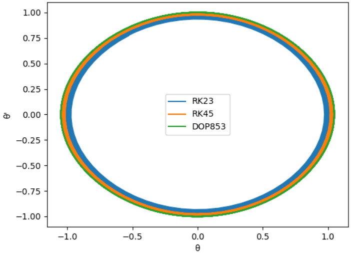
$$y^{(n+1)} = y^{(n)} + h \sum_{i=1}^{s} b_i f\left(t^{(n)} + c_i h, y^{(n)} + h\left(\sum_{j=1}^{s-1} a_{ij} k_j\right)\right)$$

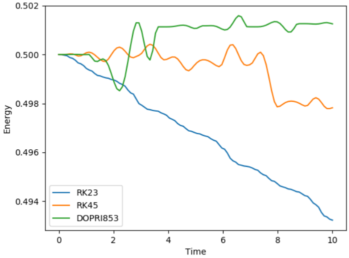
- RK45
- RK23
- DOP853

```
solve_ivp(f, t_span, y0, method, t_eval)
solve_ivp(f, [0,10], [0,1], "RK45", np.linspace(0,10,100))
```

Explicit methods - results







Implicit methods

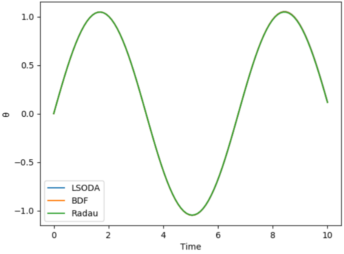
Adams-Moulton/Implicit methods

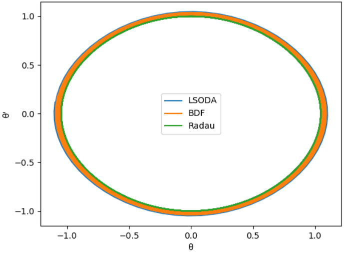
$$y^{(n+1)} = y^{(n)} + h * f(t^{(n+1)}, y^{(n+1)})$$

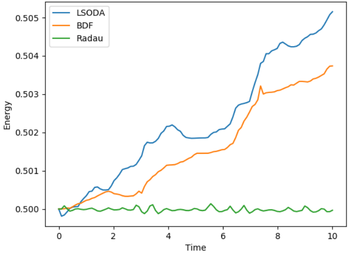
- LSODA
- BDF
- Radau

```
solve_ivp(f, t_span, y0, method, t_eval,jac)
solve_ivp(f, [0,10], [0,1], "LSODA", np.linspace(0,10,100),
```

Explicit methods - results







Performance

Method	Time
Euler	8.79 ms
RK23	4.04 ms
RK45	1.45 ms
DOP853	2.04 ms
LSODA	4 ms
BDF	11.7 ms
Radau	5.29 ms

Object oriented solver

scipy.integrate.ode

Same methods, more control!

```
s = ode(f)
s.set_f_params(k)
s.set_initial_value(x0,0)
s.set_integrator("lsoda")
X = [x0]
T = [0.]
while s.t < tmax:
  X.append(s.integrate(s.t+dt))
  T.append(s.t)
```

Documentation

- https://docs.scipy.org/doc/scipy/reference/index.html
- https://numpy.org/doc/stable/reference/index.html