07a_ode

October 16, 2019

1 ODE

We will solve the following linear Cauchy model

$$y'(t) = \lambda y(t) \tag{1}$$
$$y(0) = 1 \tag{2}$$

whose exact solution is

$$y(t) = e^{\lambda t}$$

[1]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import *
import scipy.linalg
import numpy.linalg

l = -5.
t0 = 0.
tf = 10.
y0 = 1.

s = linspace(t0,tf,5000)

exact = lambda x: exp(l*x)
```

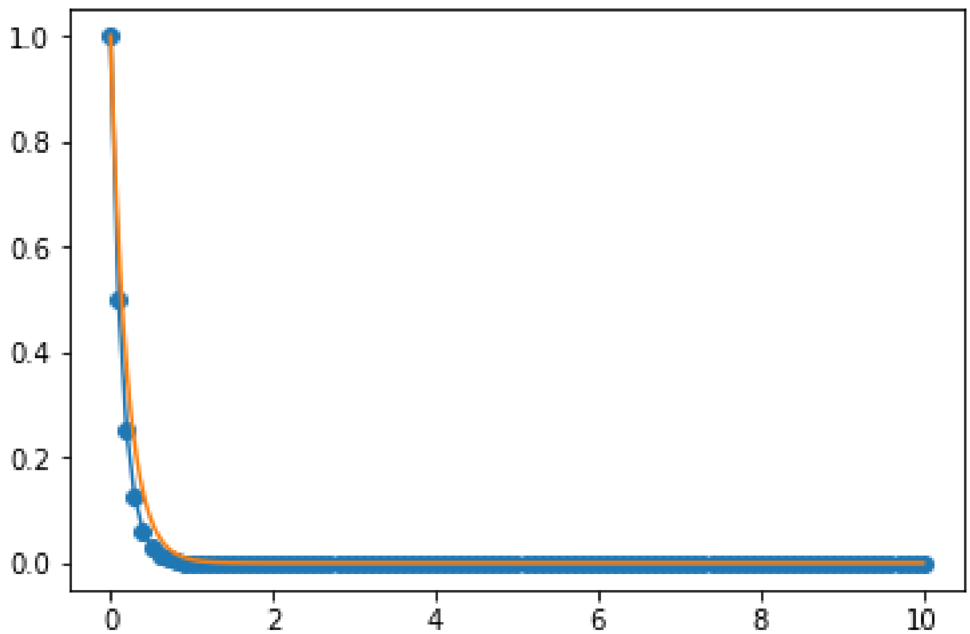
1.0.1 Forward Euler

$$\frac{y_n - y_{n-1}}{h} = f(y_{n-1}, t_{n-1})$$

[8]:
```
def fe(l,y0,t0,tf,h):
    timesteps = arange(t0,tf+1e-10, h)
    sol = zeros_like(timesteps)
    sol[0] = y0
```

```
    for i in range(1,len(sol)):
        sol[i] = sol[i-1]*(1+l*h)

    return sol, timesteps

y, t = fe(l,y0,t0,tf,0.1)

_ = plot(t,y, 'o-')
_ = plot(s,exact(s))

error = numpy.linalg.norm(exact(t) - y, 2)
print(error)
```

```
0.211605395525
```



1.0.2 Backward Euler

$$\frac{y_n - y_{n-1}}{h} = f(y_n, t_n)$$

[9]:
```
def be(l,y0,t0,tf,h):
    pass # TODO
```

1.0.3 θ-method

$$\frac{y_n - y_{n-1}}{h} = \theta\, f(y_n, t_n) + (1-\theta)\, f(y_{n-1}, t_{n-1})$$

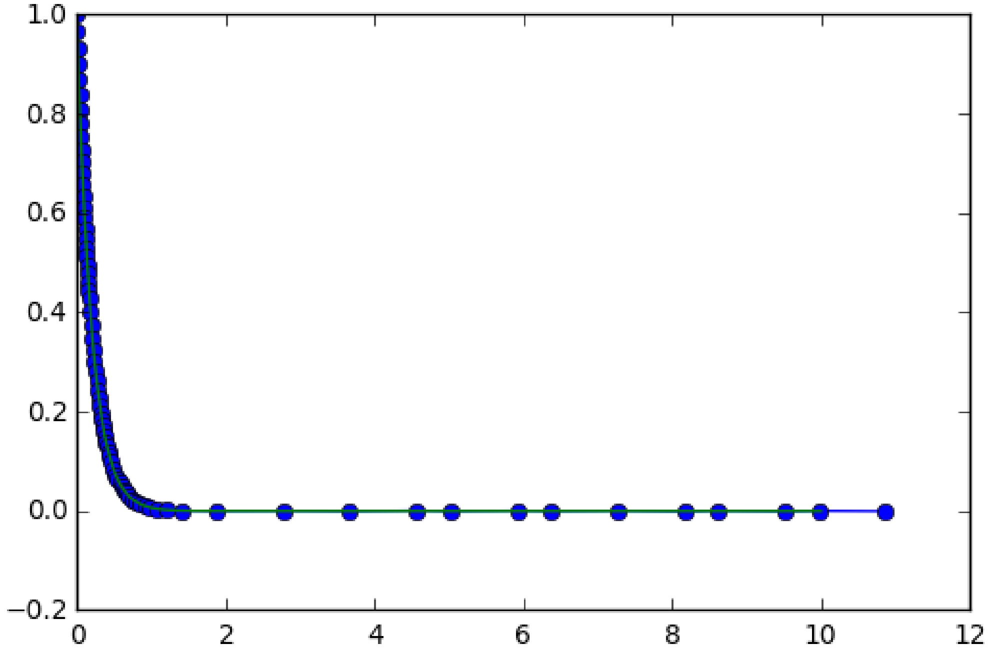
[10]:
```
def tm(theta,l,y0,t0,tf,h):
    pass # TODO
```

1.0.4 Simple adaptive time stepper

For each time step: - Compute solution with CN - Compute solution with BE - Check the difference - If the difference satisfy a given tolerance: - keep the solution of higher order - double the step size - go to the next step - Else: - half the step size and repeat the time step

[33]:
```
def adaptive(l,y0,t0,tf,h0, hmax=0.9,tol=1e-3):
    sol = []
    sol.append(y0)
    t = []
    t.append(t0)
    h = h0
    while t[-1] < tf:
        #print 'current t =', t[-1], '            h=', h
        current_sol = sol[-1]
        current_t = t[-1]
        sol_cn, _ = tm(0.5,l,current_sol,current_t, current_t + h, h)
        sol_be, _ = tm(1.,l,current_sol,current_t, current_t + h, h)

        if (abs(sol_cn[-1] - sol_be[-1]) < tol): #accept
            sol.append(sol_cn[-1])
            t.append(current_t+h)
            h *= 2.
            if h > hmax:
                h=hmax
        else:
            h /= 2.

    return sol, t

y,t = adaptive(l,y0,t0,tf,0.9)
_ = plot(t,y, 'o-')
_ = plot(s,exact(array(s)))

error = numpy.linalg.norm(exact(array(t)) - y, infty)
print error, len(y)
```

```
0.000817298421905 74
```


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
2
4
6
8
10
12

```
[ ]:
```

```
[ ]:
```