numerical solution of ODEs

Ben Bolker

17:23 19 March 2017

Differential equations

- ubiquitous in applied math, engineering, epidemiology
- modeling continuous-state, continuous-time, non-spatial, deterministic systems
- simple univariate ODEs, e.g. $dN/dt = rN$ or $dN/dt = rN(1 - N/K)$
- *systems* of ODEs, e.g. **SIR model**
- solving analytically is deep & complicated, gives general answers
- solving numerically is usually easy (but we only get answers for a particular set of parameters)
- to solve an ODE numerically we need
  - *gradients* (dx/dt), specified as a function of current state and time (for *non-autonomous* systems)
  - *initial conditions*
  - time step, length of time to integrate for

Euler's method

- simplest possible approach
- $N(t + \delta t) = N(t) + \delta t \cdot \frac{dN}{dt}$
- let's solve the logistic equation

```
import numpy as np
import matplotlib.pyplot as plt
## parameters
r=1
K=4
## time info
dt=0.1
t_tot=15
t_vec = np.arange(0,t_tot,step=dt)
x = np.zeros(len(t_vec))
x[0]=0.1  ## initial conditions
for i in range(1,len(t_vec)):
   g = r*x[i-1]*(1-x[i-1]/K)
   x[i] = x[i-1]+g*dt
fig, ax = plt.subplots()
ax.plot(t_vec,x)
fig.savefig("pix/logist1.png")
```

Now let's rewrite it a little more generally:

```
def logist_grad(x,t,params):
    r,K = params  ## unpack parameters
    return(r*x*(1-x/K))
```

Now let's rewrite it a general Euler-stepping function:

```
def euler_solve(t_vec,x0,gradfun,params):
   """solve differential equation by Euler's method"""
```

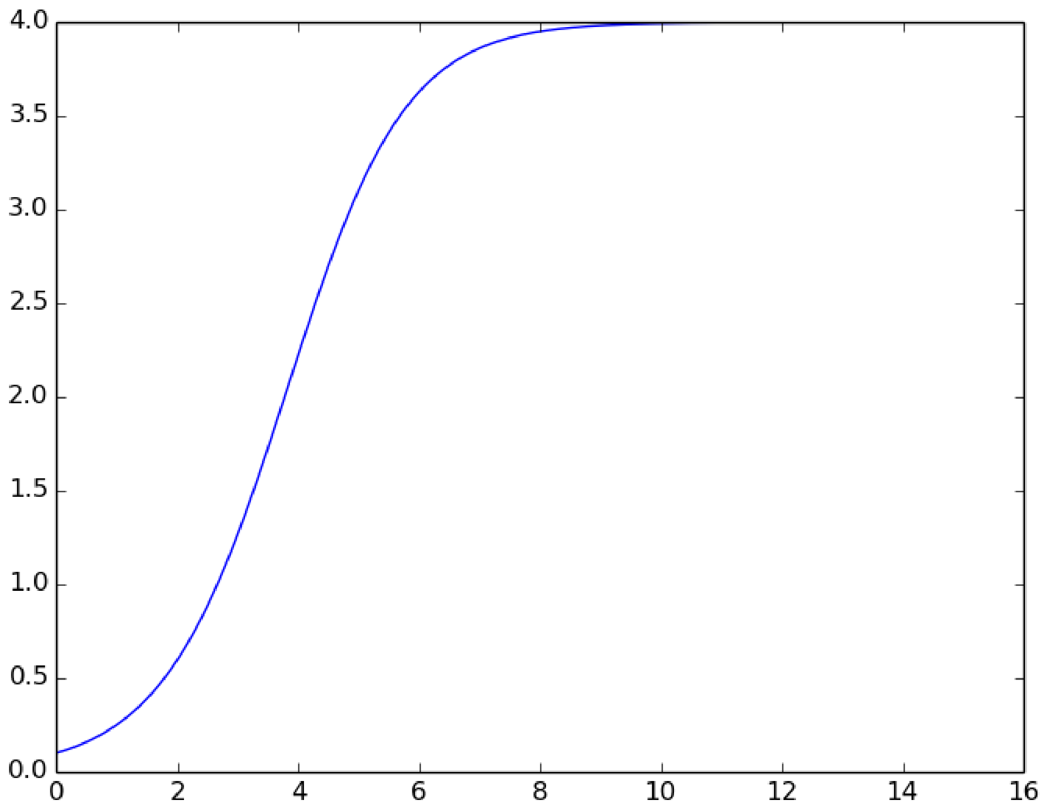

Figure 1:

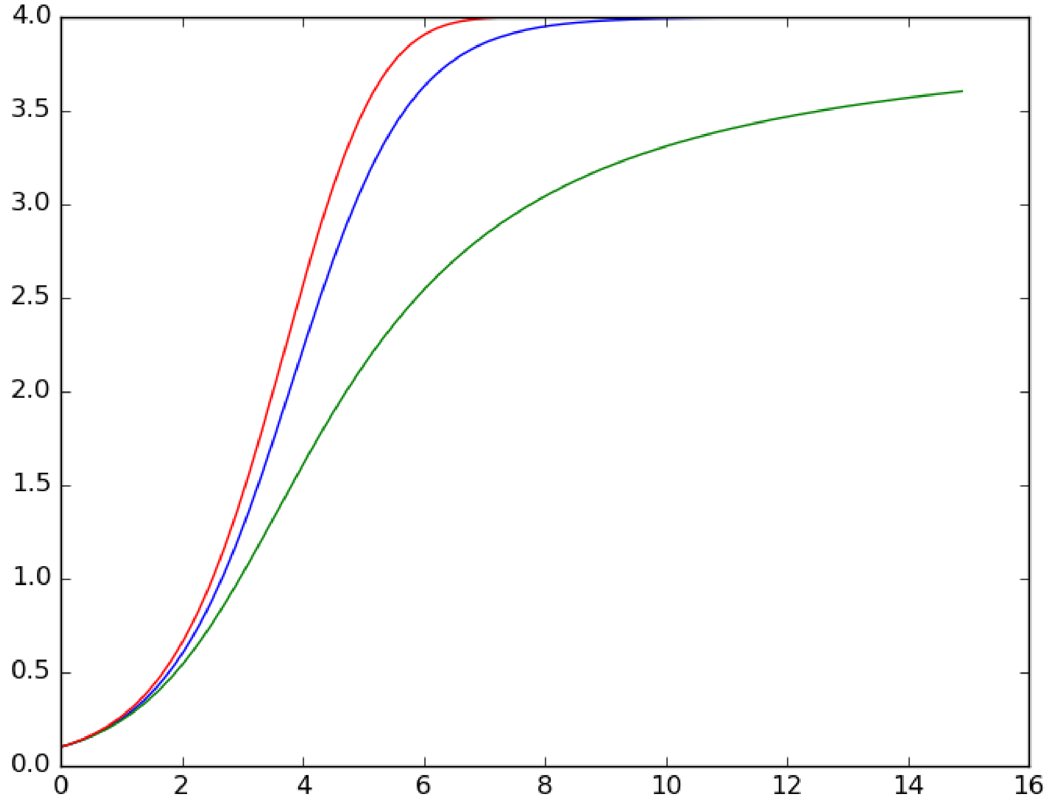

Figure 2:

Now we can easily change the model, e.g. try running the *theta-logistic* model:

```
def theta_logist_grad(x,t,params):
   r,K,theta = params
   return(r*x*(1-x/K)**theta)
```

What about *systems* of equations?

$$
\begin{aligned}
\frac{dS}{dt} &= -\beta S I \\
\frac{dI}{dt} &= \quad \beta S I - \gamma I \\
\frac{dR}{dt} &= \qquad\qquad + \gamma I
\end{aligned}
$$

Modifications needed to solve a system of equations:

```
def euler_msolve(t_vec,x0,gradfun,params):
    """solve differential equation by Euler's method"""
   x = np.zeros((len(t_vec),len(x0)))
```

Define a gradient function for the SIR model:

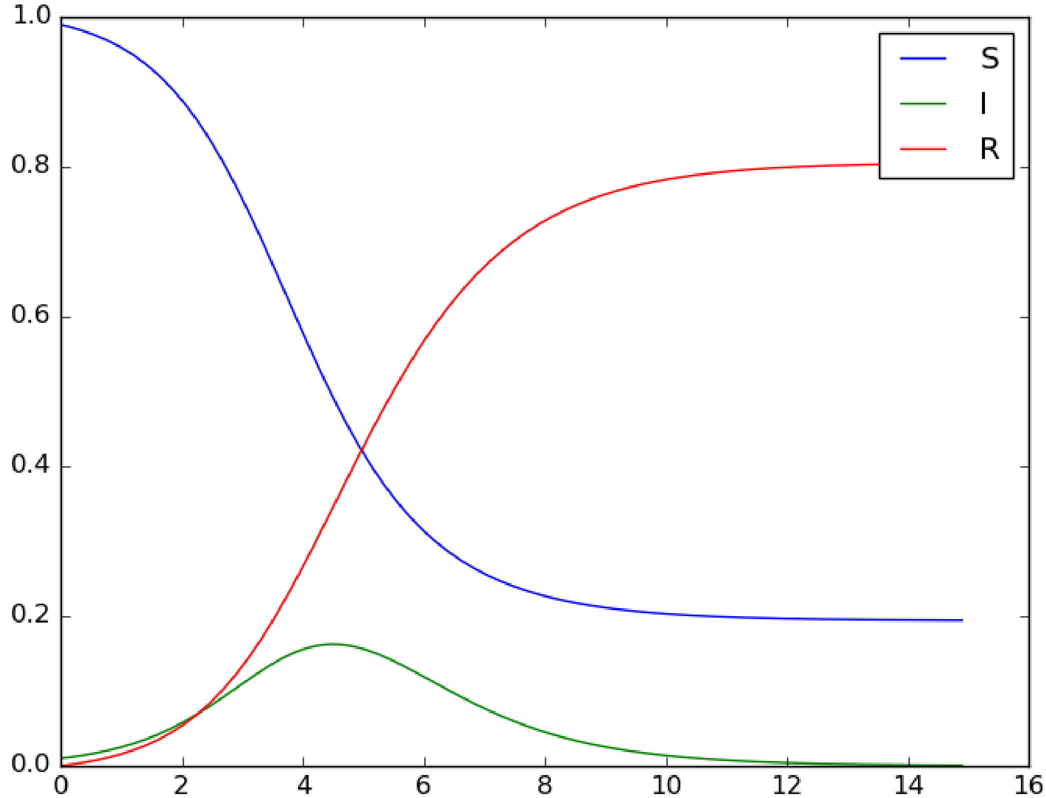

Figure 3:

```
def SIR_grad(x,t,params):
   beta,gamma = params
   S,I,R = x
```

Solve it:

```
SIR_sol1 = euler_msolve(t_vec,x0=(0.99,0.01,0),
```

Plot it:

```
fig, ax = plt.subplots()
ax.plot(t_vec,SIR_sol1)
ax.legend(("S","I","R"))
fig.savefig("pix/SIR1.png")
```

It's always better not to re-invent the wheel!

```
import scipy.integrate
SIR_sol2 = scipy.integrate.odeint(SIR_grad,
                       y0=(0.99,0.01,0),
                       t=t_vec,
                       args=((2,1),))
```

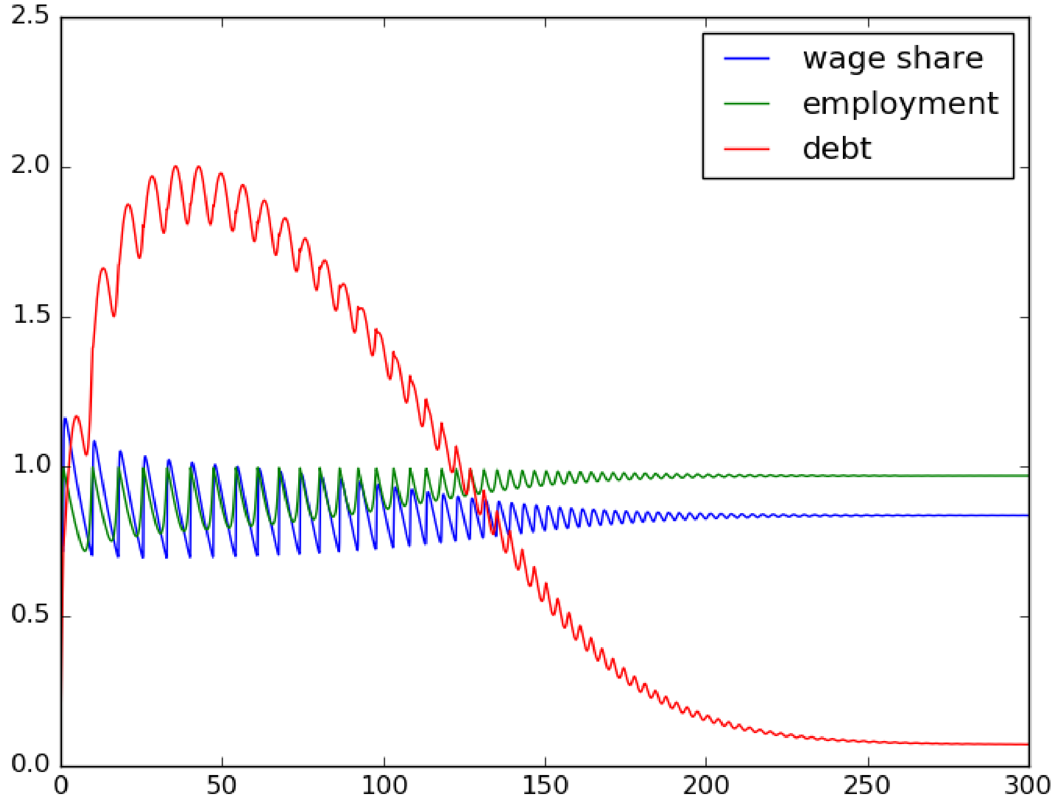

Figure 4:

Results of a more complicated ODE from Grasselli and Costa Lima, “An analysis of the Keen model for credit expansion, asset price bubbles and financial fragility” (Math Finan Econ (2012) 6:191–210 DOI:10.1007/s11579-012-0071-8; code