Solving ODEs in Python

Joe Pitt-Francis
Computer Science

(From MATLAB slides by James Osborne)

Aim and contents

- Aim: Learn techniques for the solution of systems of *Ordinary Differential Equations*
- Contents:
  - Analytical methods for simple ODEs
  - Reducing the order of ODEs
  - Numerical methods for first order ODEs
    - Half-day exercise
  - Python for solving initial value problems
  - Python for solving boundary value problems

First order ODEs?

- ODE - Ordinary Differential Equation,
  - With respect to one variable, t or x etc.
- Order of ODE - order of the highest derivative
- First order ODE:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y), \; y(0) = a.$$

- Simple problems – solve analytically
  - Separable solutions, integrating factors
- Highly non-linear problems or unknown integral, then solve numerically
  - Forward Euler method, Runge-Kutta method...
  - In-built `scipy` (or other) solvers

Analytical methods

Analytical techniques

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{f(x)}{g(y)},$$ Separable solutions

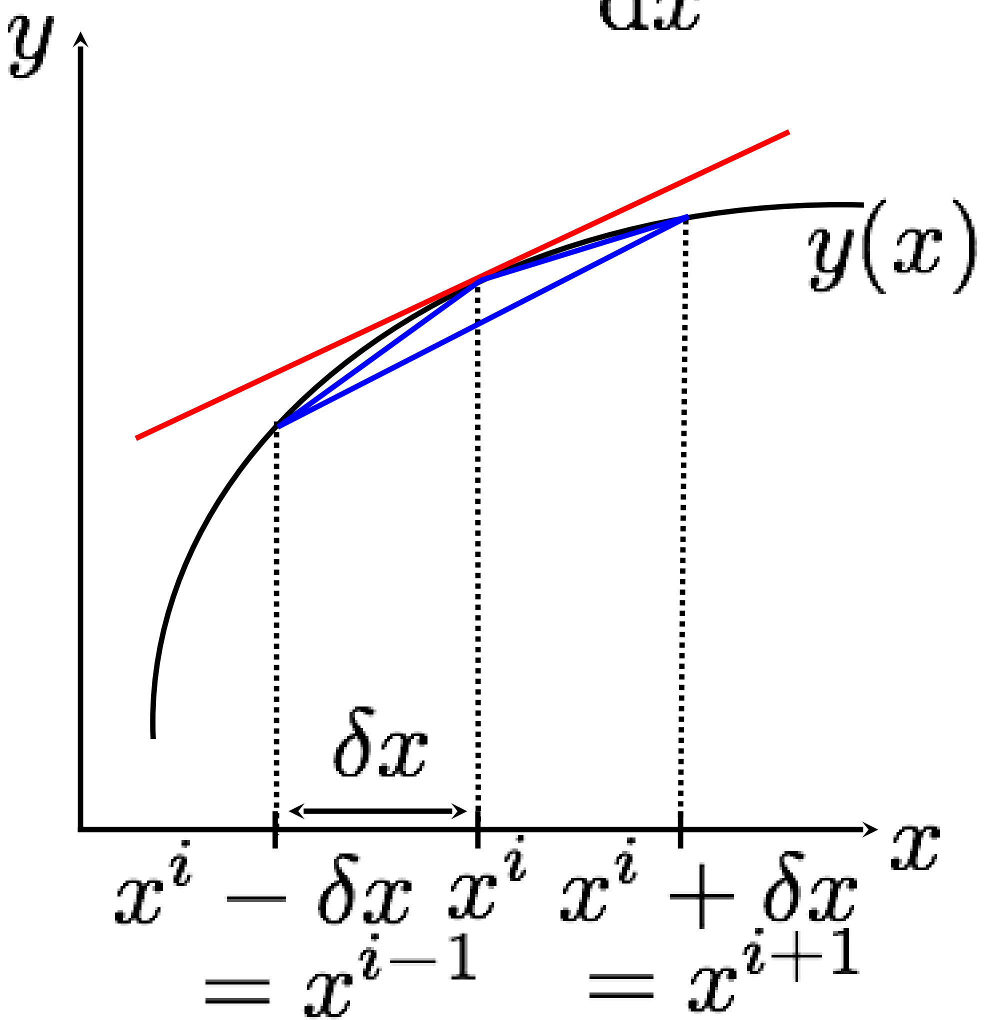
$$\frac{\mathrm{d}y}{\mathrm{d}x} + f(x)y = g(x),$$ Integrating factor

$$a\frac{\mathrm{d}^2y}{\mathrm{d}x^2} + b\frac{\mathrm{d}y}{\mathrm{d}x} + cy = 0,$$ Auxiliary equation

Numerical approaches

Numerical differentiation

Aim to calculate $\frac{\mathrm{d}y}{\mathrm{d}x}$ numerically.



Backward difference

$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{y(x^i) - y(x^{i-1})}{\delta x},$$

Forward difference

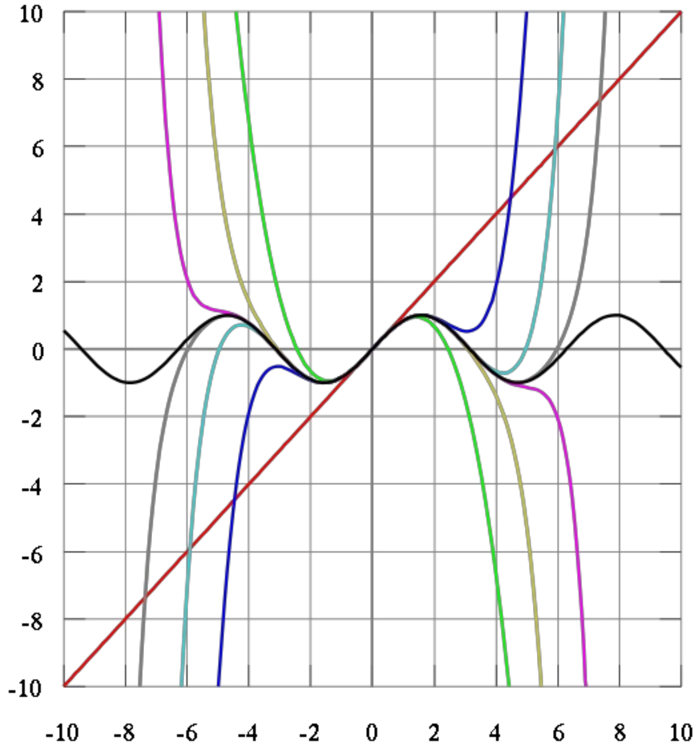
$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{y(x^{i+1}) - y(x^i)}{\delta x},$$

Central difference

$$\frac{\mathrm{d}y}{\mathrm{d}x} \approx \frac{y(x^{i+1}) - y(x^{i-1})}{2\delta x}.$$

Taylor expansion

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2 f''(a)}{2!} + \frac{(x-a)^3 f'''(a)}{3!} + \cdots$$



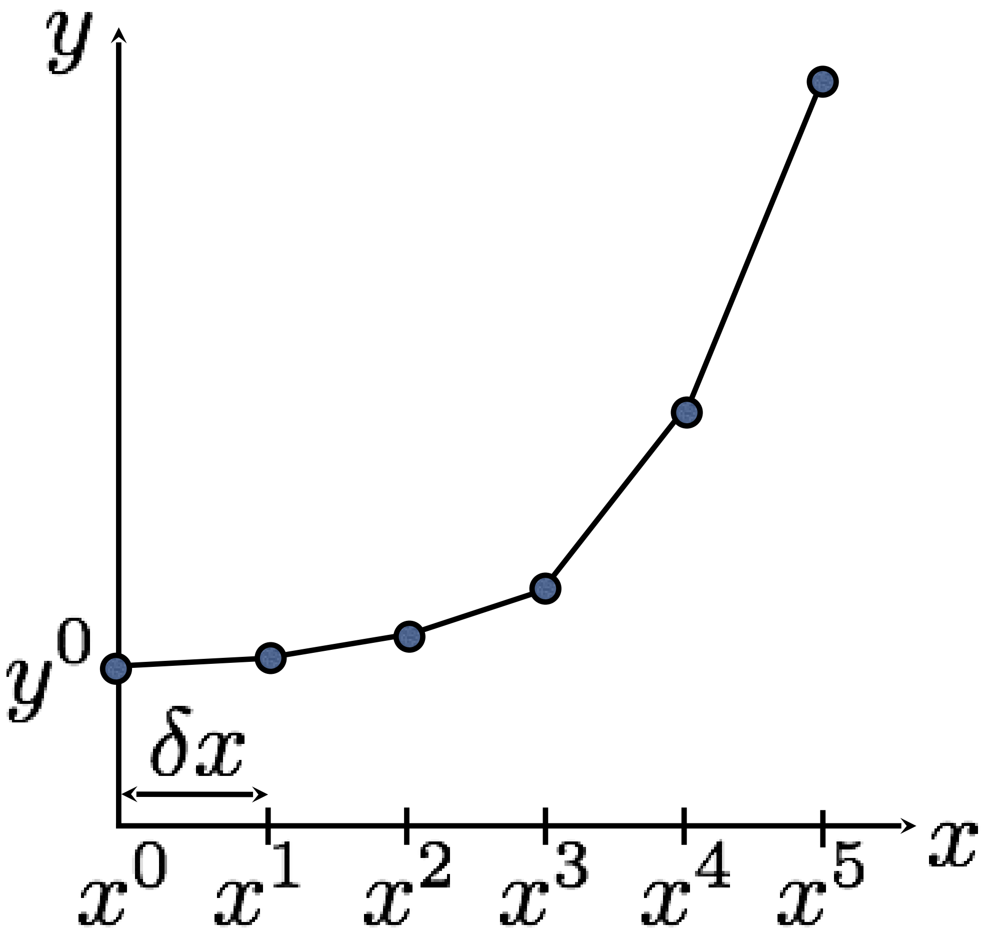
$\sin(x)$

From Wikipedia

Use this to prove the finite difference formulas

Euler method

Want to solve $\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y),$ such that $y(0) = a.$



$$x^0 = 0,$$
$$x^i = x^{i-1} + \delta x$$
$$y^0 = a,$$
$$y^i \approx y(x^i),$$

$$\frac{y^{i+1} - y^i}{\delta x} = f(x, y).$$

Forward vs Backward Euler

$$\frac{y^{i+1} - y^i}{\delta x} = f(x^i, y^i),$$

Forward Euler method
"Explicit"

$$y^{i+1} = y^i + \delta x f(x^i, y^i),$$

$$\frac{y^{i+1} - y^i}{\delta x} = f(x^{i+1}, y^{i+1}),$$

Backward Euler method
"Implicit"

$$y^{i+1} - \delta x f(x^{i+1}, y^{i+1}) = y^i,$$

Forward - conditionally stable
Backward - unconditionally stable

Euler method for systems of ODEs

Can extend this to systems of ODEs

$$\left.\begin{aligned} \frac{\mathrm{d}y_1}{\mathrm{d}x} &= f_1(x, y_1, y_2, y_3), \\ \frac{\mathrm{d}y_2}{\mathrm{d}x} &= f_2(x, y_1, y_2, y_3), \\ \frac{\mathrm{d}y_3}{\mathrm{d}x} &= f_3(x, y_1, y_2, y_3), \end{aligned}\right\} \quad \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x} = \mathbf{f}(x, \mathbf{y}),$$

$$\mathbf{y}^i = (y_1^i, y_2^i, y_3^i), \qquad \frac{\mathrm{d}\mathbf{y}}{\mathrm{d}x} \approx \frac{\mathbf{y}^{i+1} - \mathbf{y}^i}{\delta x}.$$

Higher order ODEs

$$a(x)\frac{\mathrm{d}^2y}{\mathrm{d}x^2} + b(x)\frac{\mathrm{d}y}{\mathrm{d}x} + c(x)y = f(x),$$

Reduce to a system of first order ODEs

$$\left.\begin{aligned} \frac{\mathrm{d}y}{\mathrm{d}x} &= z, \\ \frac{\mathrm{d}z}{\mathrm{d}x} &= \frac{f(x) - b(x)z - c(x)y}{a(x)}, \end{aligned}\right\}$$

System of first order ODEs

Improving on Euler

- Use error analysis to improve placement of nodes (adaptivity)
- Higher order methods:
  - Runge-Kutta;
  - Dormand-Prince;
  - Adams-Bashforth;
  - Adams-Moulton
- See Suli and Mayers “An Introduction to Numerical Analysis” for more details

Pause for “Plan”

- Write your own solver
  - **Morning exercise**
- Use Python to solve systems of ODEs
  - Afternoon...

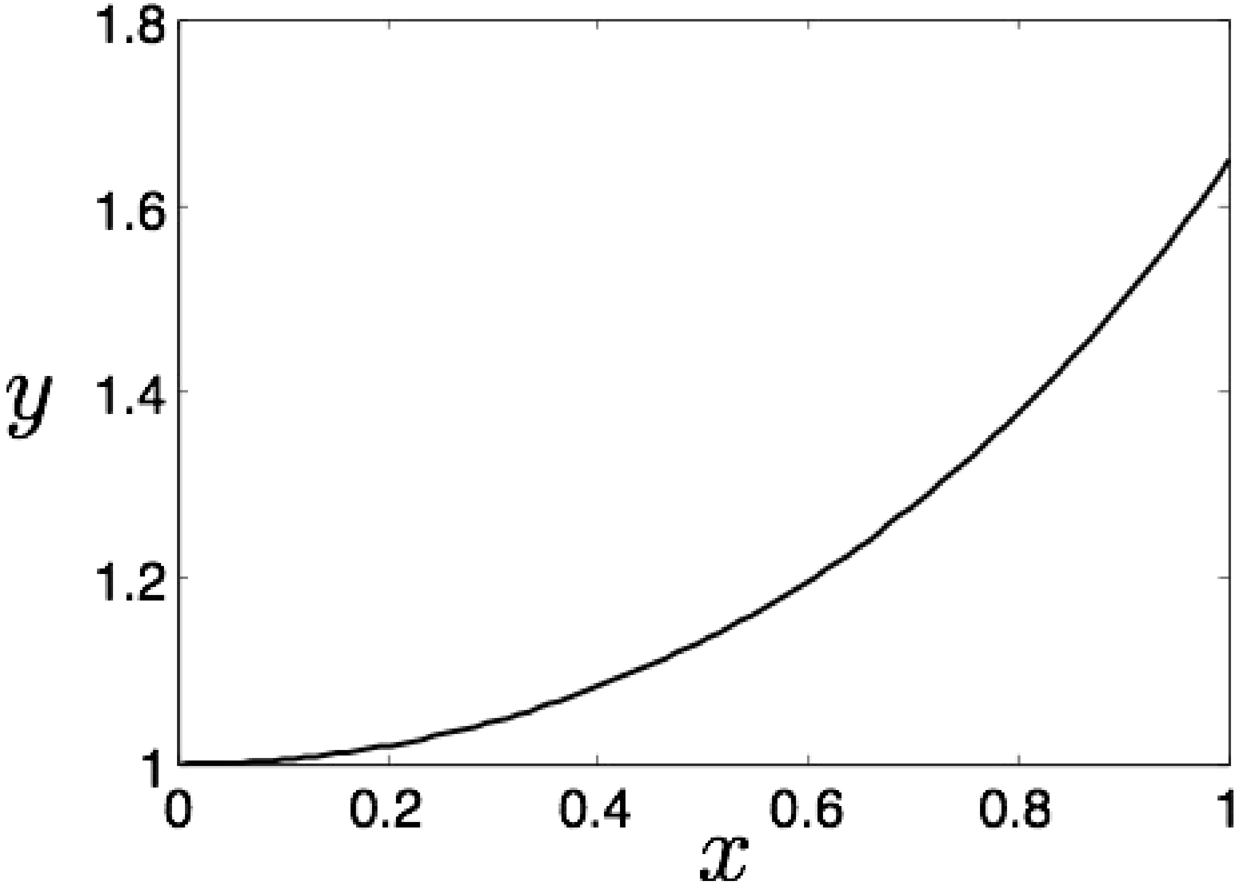
Using Python

Python and ODEs: IVPs

- Initial value problems
  - Using `odeint` or `ode`
  - `odeint` is easy to set up
  - `ode` is more configurable
  - Based on Runge-Kutta schemes
- Two examples:
  - Single ODE; and
  - Coupled system of ODEs

Example problem

$$\frac{\mathrm{d}y}{\mathrm{d}x} = xy, \;\; y(0) = 1, \longrightarrow y(x) = e^{\frac{x^2}{2}}.$$

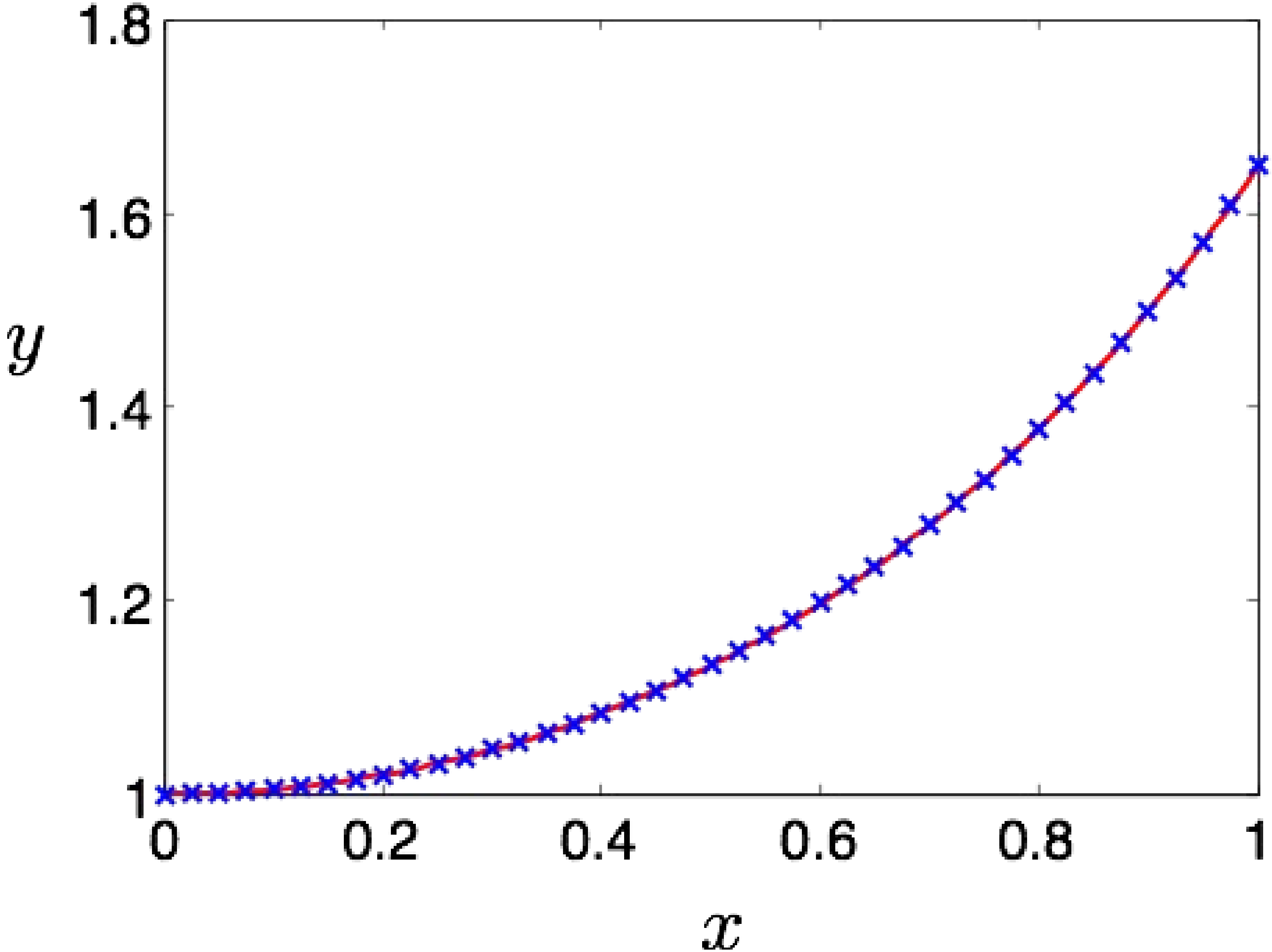


odeint versus ode

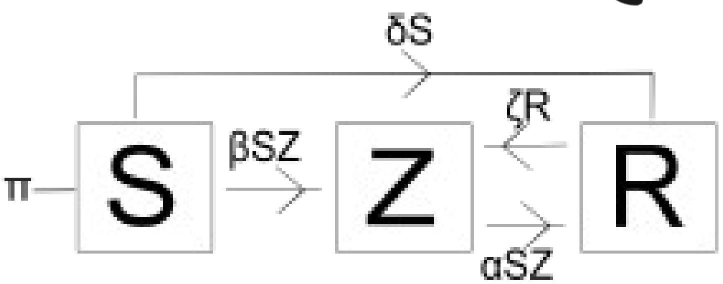
```python
# Function to solve dydx=x*y
def dydx1(y, x):
  # dydx=x*y
  return x*y

y0 = 1
xs = np.linspace(0, 1, 100)
ys = odeint(dydx1, y0, xs)

plt.plot(xs, ys)
plt.xlabel('x');
plt.ylabel('y')
plt.show()
```

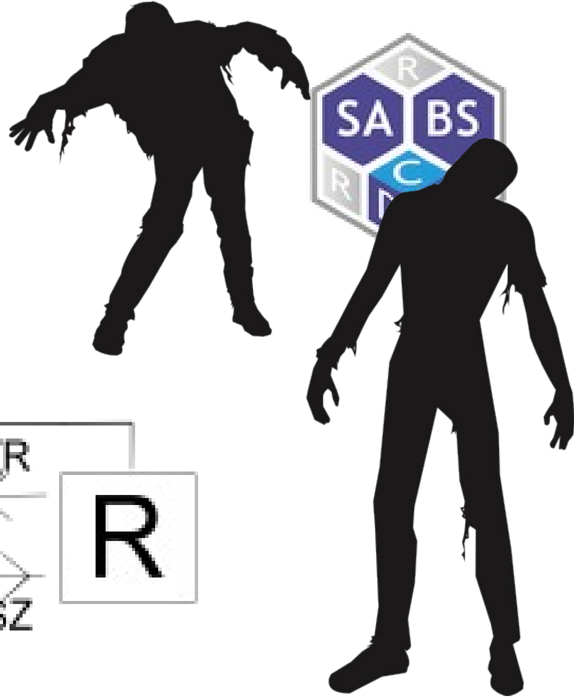
```python
# Function to solve dydx=x*y
def dydx2(x, y):
  # dydx=x*y
  return x*y

y0 = 1; x = 0
solver = ode(dydx2)
solver.set_initial_value(y0, x)
xs = [x];  ys = [y0]
while x<1:
    x += 0.01
    y=solver.integrate(x)
    ys.append(y[0])
    xs.append(x)

plt.plot(xs, ys)
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

Results



Zombie model





- "When zombies Attack!" (Munz et al. 2009) presents model of zombie invasion
- System of 3 ODEs:
  - S - Susceptibles
  - Z - Zombies
  - R - Removed

$$\frac{\mathrm{d}S}{\mathrm{d}t} = \Pi - \beta S Z - \delta S,$$

$$\frac{\mathrm{d}Z}{\mathrm{d}t} = \beta S Z + \zeta R - \alpha S Z,$$

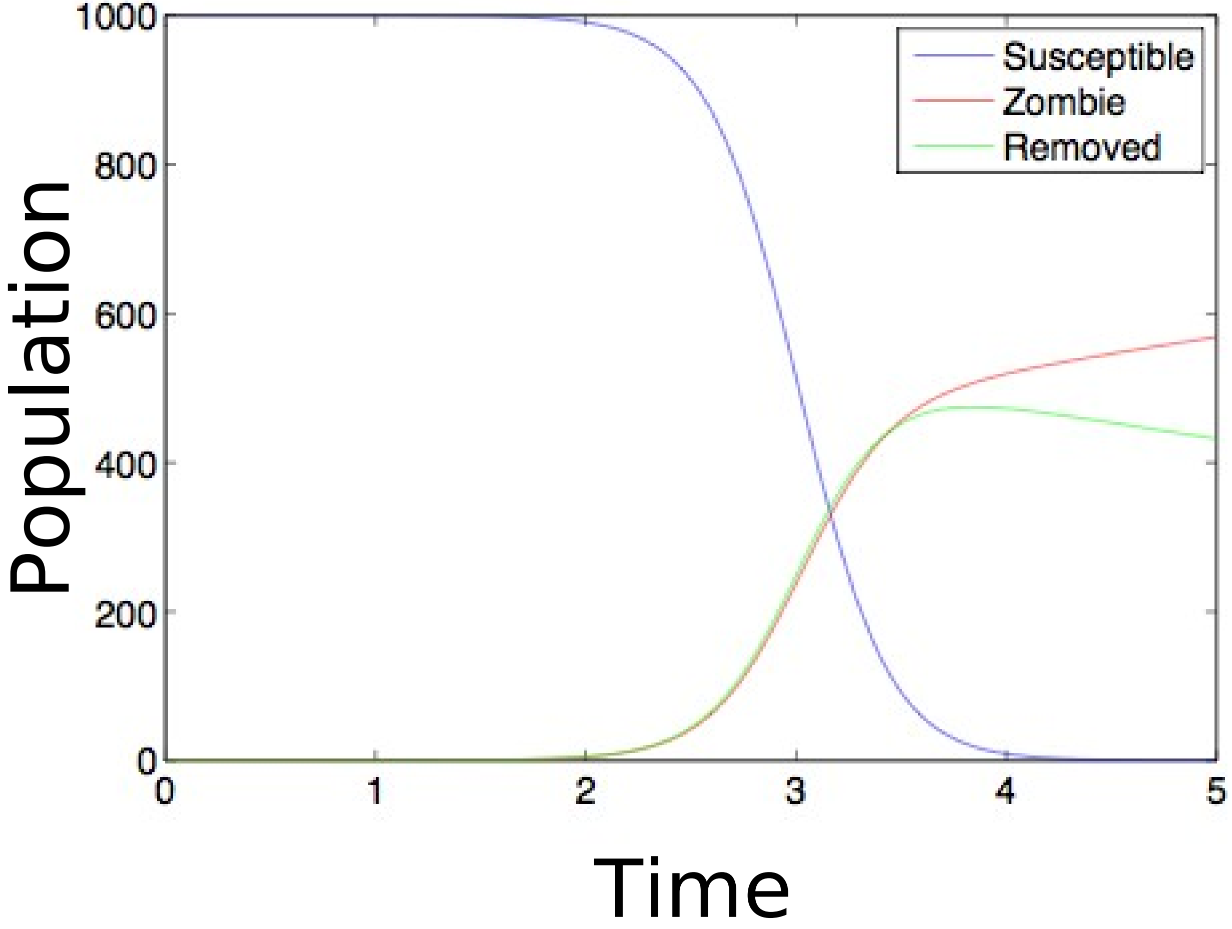
$$\frac{\mathrm{d}R}{\mathrm{d}t} = \delta S + \alpha S Z - \zeta R.$$

Zombie code

```python
# Function to solve the SZR Zombie ODE system.
alpha = 0.005; zeta = 0.1; pi = 0
beta = 0.0095; delta = 0.0001

def SZR(Y,t):
    S = Y[0]; Z = Y[1]; R = Y[2]
    #Susceptible / Zombie / Removed
    dSdt = pi - beta*S*Z - delta*S
    dZdt = beta*S*Z + zeta*R - alpha*S*Z
    dRdt = delta*S + alpha*S*Z - zeta*R
    return [dSdt, dZdt, dRdt]

S0=1000; Z0=0; R0=0
EndTime = 5
t = np.linspace(0, EndTime, 100)
Y0 = [S0, Z0, R0]
Y = odeint(SZR, Y0, t)
plt.plot(t,Y[:,0], 'b', label='Susceptible')
plt.plot(t,Y[:,1], 'r', label='Zombie')
plt.plot(t,Y[:,2], 'g', label='Removed')
plt.xlabel('Time');plt.ylabel('Population')
plt.legend();plt.show()
```

Zombie results

$$S_0 = 1000,$$
$$Z_0 = 0,$$
$$R_0 = 0.$$



More complicated models in exercises

Python and ODEs: BVPs

- Boundary value problems
  - The shooting method
- Simple example:
  - $$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} + y = 0, \text{ with } y'(0) = 1, y(\pi) = 0.$$

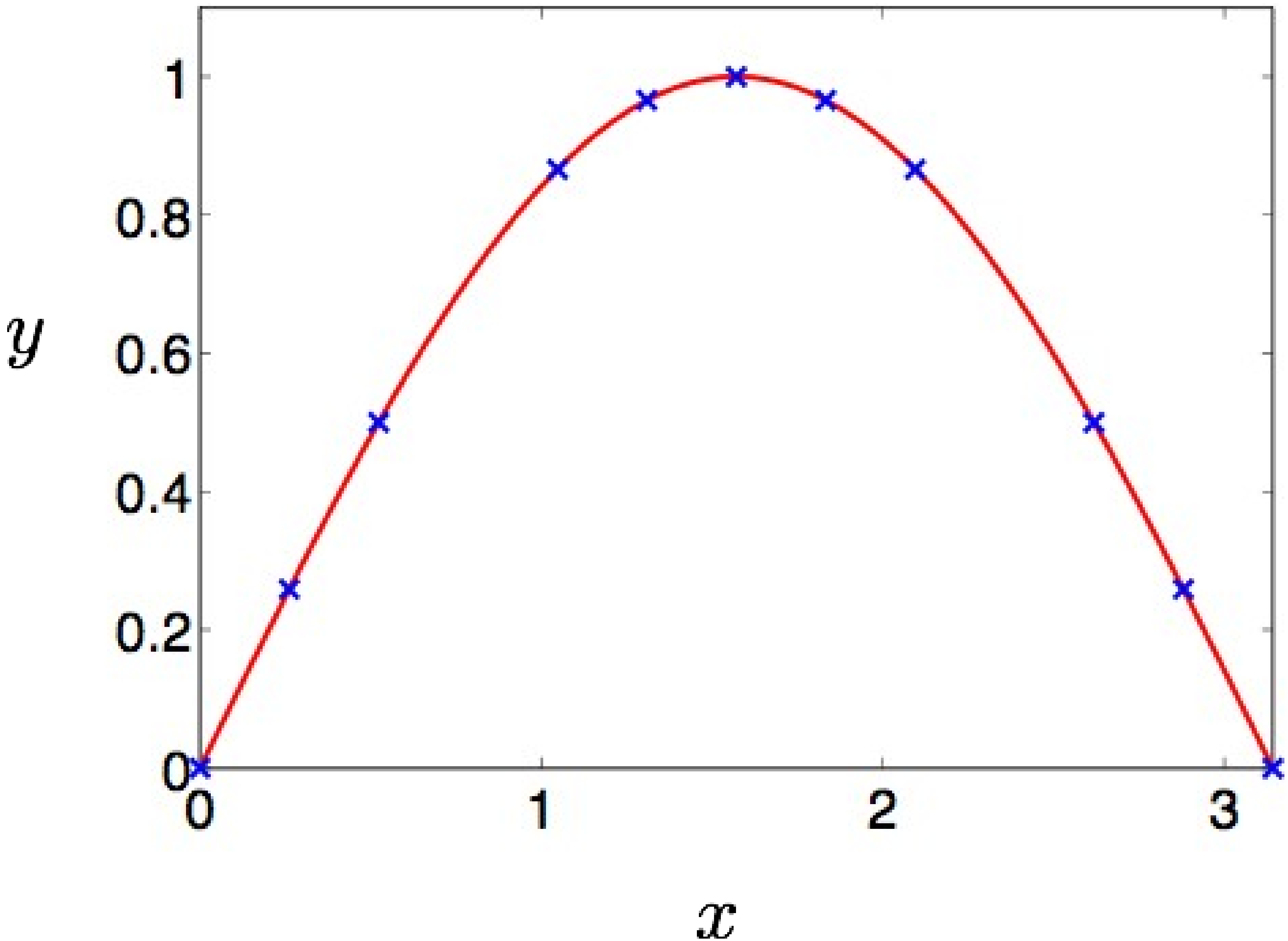
$$\left.\begin{aligned} \frac{\mathrm{d}y}{\mathrm{d}x} &= z, \\ \frac{\mathrm{d}z}{\mathrm{d}x} &= -y, \end{aligned}\right\} \text{First order system}$$

Exact solution $y = sin(x)$.

Simple BVP code

```
# Solving y'' + y =0
# y[0] is y, y[1] is y'
# dy[0]/dx = y[1] and dy[1]/dx = -y[0]
def dydx(x, y):
     return np.vstack((y[1], -y[0]))

def bcs(yat0, yatpi):
    # Neumann/Dirichlet y'(x=0)= 1, y(x=pi) = 0
    return (yat0[1]-1, yatpi[0])

x = np.linspace(0, math.pi, 10)
init_y = np.ones((2, x.size))
sol = solve_bvp(dydx, bcs, x, init_y)
plt.plot(sol.x, sol.y[0], 'b+')
xs = np.linspace(0, math.pi, 100)
plt.plot(xs, np.sin(xs), 'r')
plt.xlabel('x'); plt.ylabel('y')
plt.show()
```

Results



Overview

- Techniques for solution of simple ODEs
- Reduction of higher order systems to first order
- Numerical differentiation
- Use Python to solve systems of ODEs with
  - initial conditions, and
  - boundary conditions

Separable solutions

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{f(x)}{g(y)}, \qquad \int g(y)\mathrm{d}y = \int f(x)\mathrm{d}x + C,$$

Example

$$\frac{\mathrm{d}y}{\mathrm{d}x} = xy, \;\; y(0) = 1,$$

$$y(x) = e^{\frac{x^2}{2}}$$

Integrating factors

$$\frac{\mathrm{d}y}{\mathrm{d}x} + f(x)y = g(x), \qquad \phi(x) = e^{\{\int f(x)\mathrm{d}x\}},$$

Example

$$\frac{\mathrm{d}y}{\mathrm{d}x} + \frac{y}{x} = 1, \; x > 0, \; y(1) = 0,$$

$$\phi(x) = x, \quad y(x) = \frac{x}{2} - \frac{1}{2x}.$$

Second order ODE with linear coefficients

$$a\frac{\mathrm{d}^2y}{\mathrm{d}x^2} + b\frac{\mathrm{d}y}{\mathrm{d}x} + cy = 0,$$ try $$y(x) = Ae^{kx}.$$

$$ak^2 + bk + c = 0,$$ Auxiliary equation

A,B from ICS/BCS

real
$k = a, b$

$$y = Ae^{ax} + Be^{bx},$$

imaginary
$k = \pm\beta i$

$$y = A\sin(\beta x) + B\cos(\beta x),$$

complex
$k = \alpha \pm \beta i$

$$y = e^{\alpha x}[A\sin(\beta x) + B\cos(\beta x)].$$