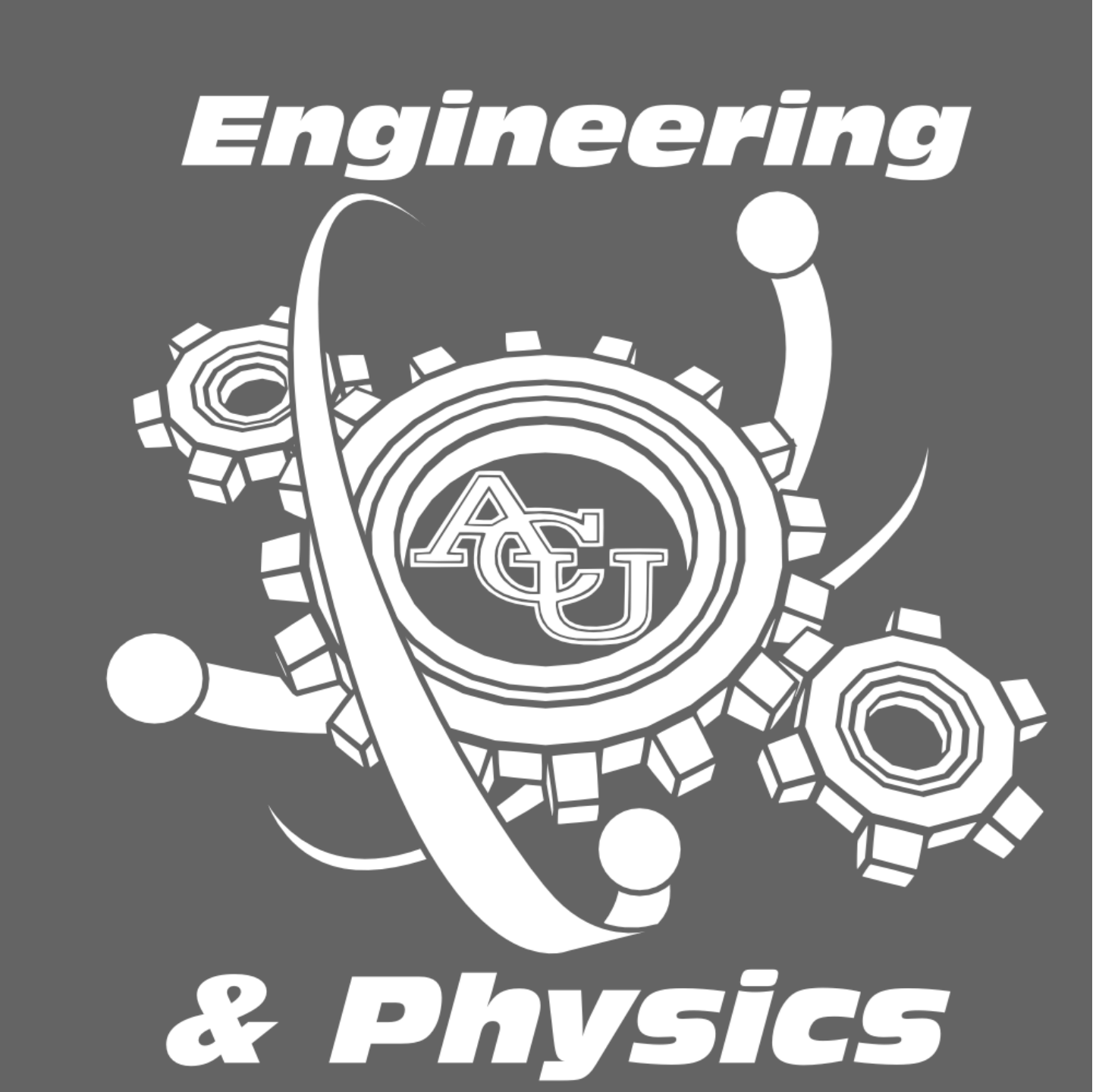


PHYS 351
ODE IVP

Dr. Daugherity

Abilene Christian University

Problem Statement

Find an unknown function $y(t)$ given:

- information about its derivatives
- an initial value $y(t_0) = y_0$

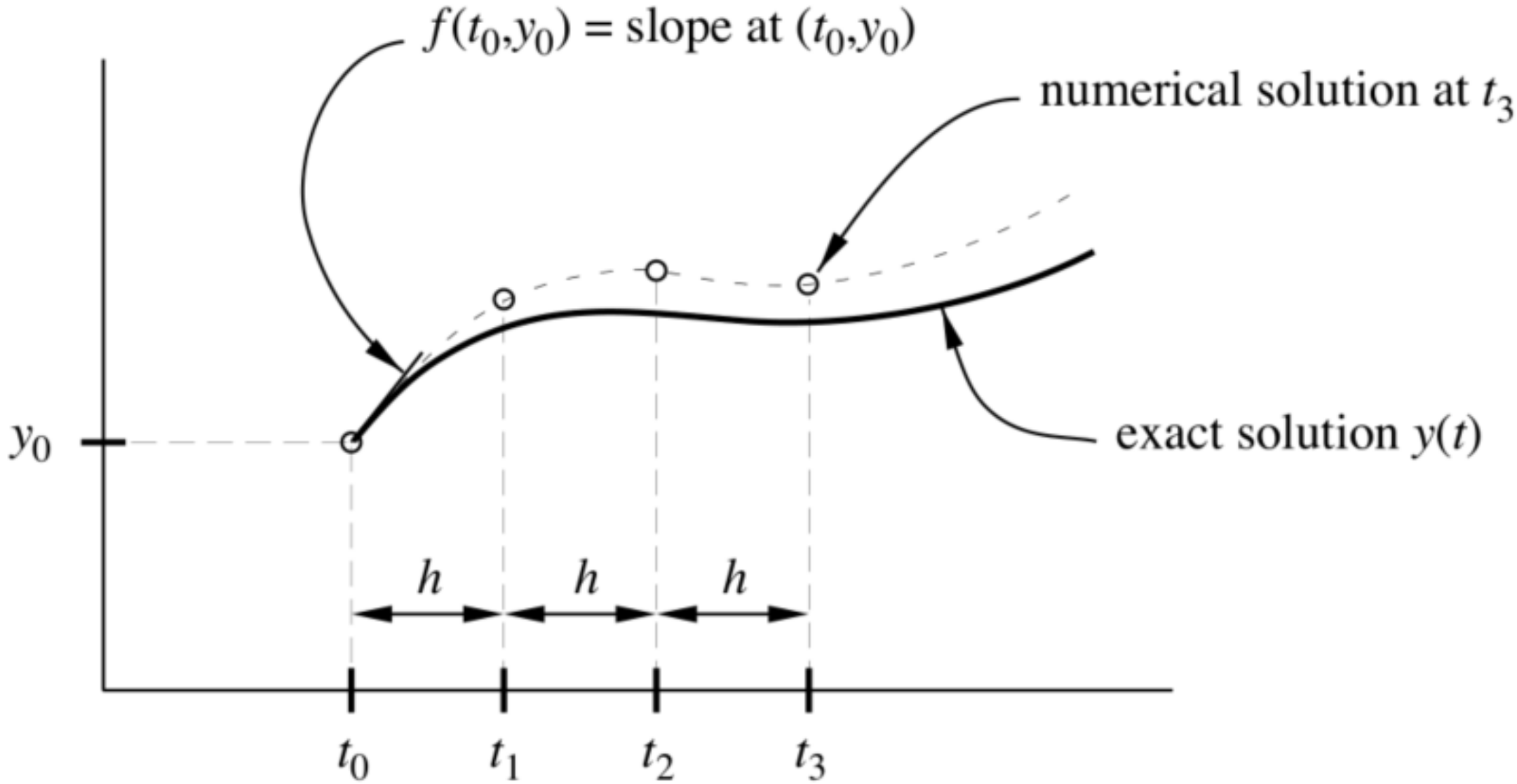
INITIAL VALUE PROBLEM!

All constraints occur at the same time

We also want to do this for **coupled systems**!

Only first derivatives

FIRST-ORDER PROBLEMS


$f(t_0,y_0)$ = slope at (t_0,y_0)
numerical solution at t_3
exact solution $y(t)$
y_0
h
h
h
t_0
t_1
t_2
t_3

Euler

$$y(t_0 + h) \approx y(t_0) + h\, y'(t_0, y_0)$$

Leonhard Euler
1707 - 1783

While each step has error h^2 the number of steps goes as $1/h$, therefore the total error is $O(h)$

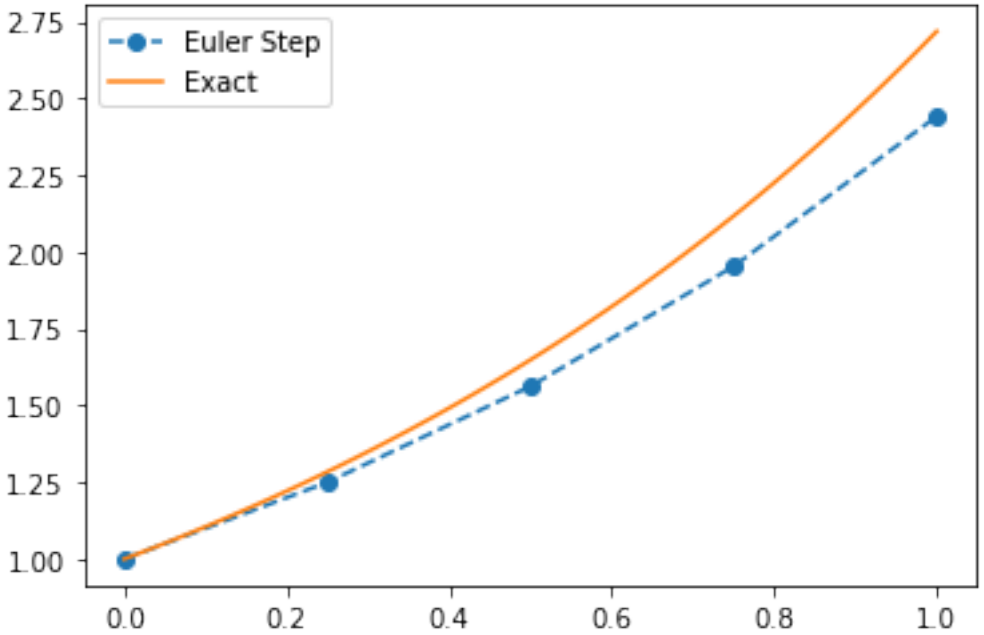
Euler

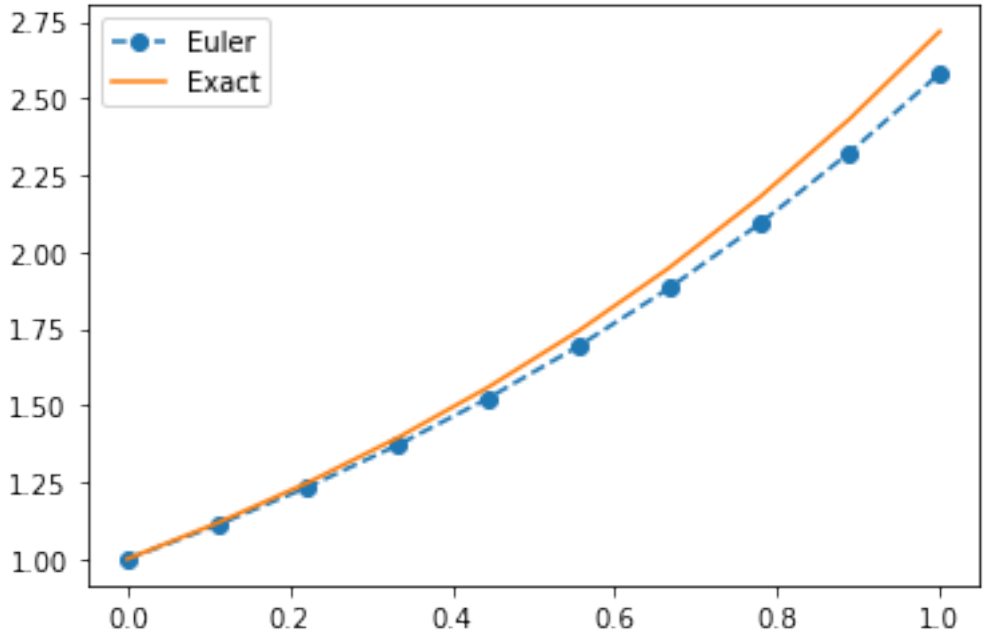
- Make a function `yprime(t,y)` that returns y'
- Try coding one step of Euler
- Try many steps…

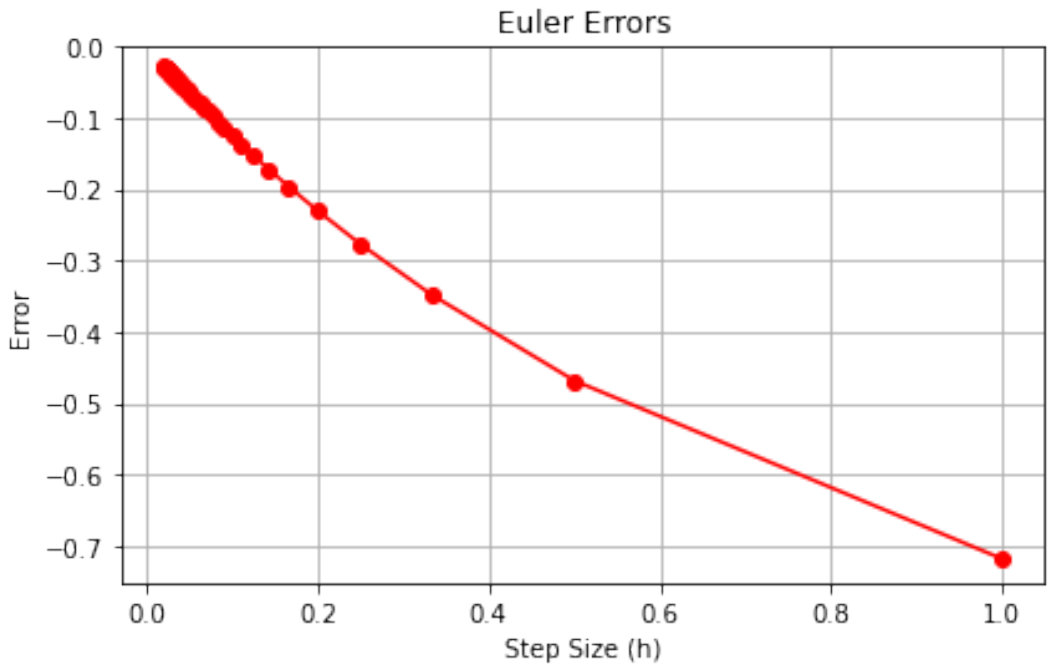
Leonhard Euler
1707 - 1783

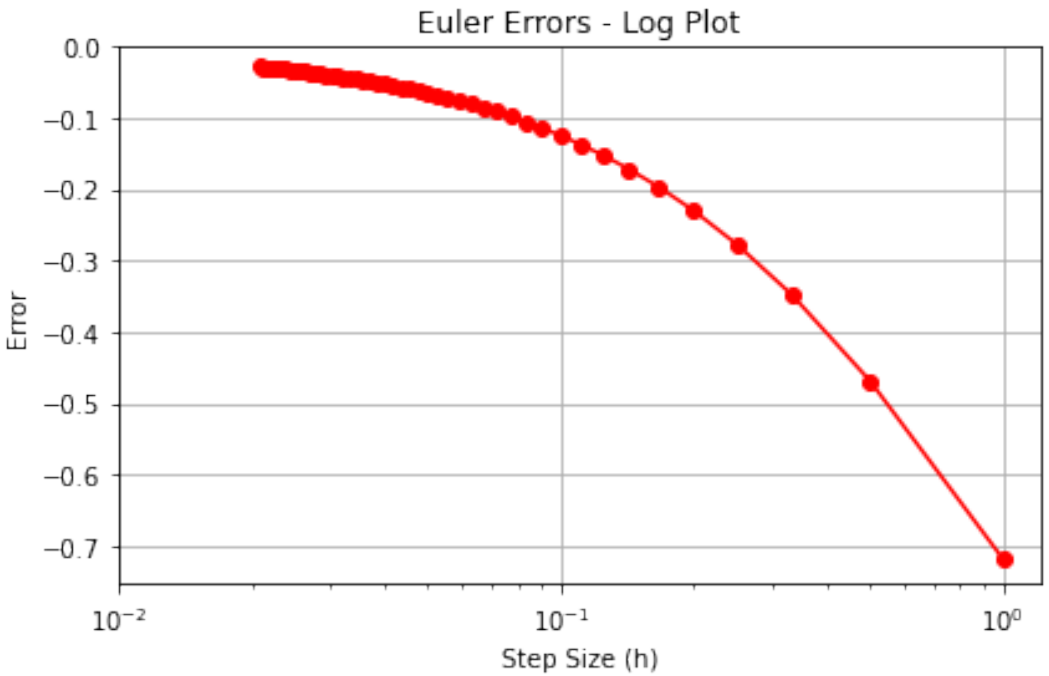
- Test on an exponential $y'(t) = ky$, exact solution $y(t) = y_0 e^{kt}$

```
def f(t,y):
  return y
```









http://mathforcollege.com/nm/mws/gen/08ode/mws_gen_ode_txt_euler.pdf

Example 3

A ball at 1200K is allowed to cool down in air at an ambient temperature of 300K. Assuming heat is lost only due to radiation, the differential equation for the temperature of the ball is given by

$$\frac{d\theta}{dt} = -2.2067\times10^{-12}\left(\theta^4 - 81\times10^8\right),\ \theta(0) = 1200\text{K}$$

where θ is in K and t in seconds. Find the temperature at $t = 480$ seconds using Euler's method. Assume a step size of $h = 240$ seconds.

The exact solution of the ordinary differential equation is given by the solution of a non-linear equation as

$$0.92593 \ln \frac{\theta - 300}{\theta + 300} - 1.8519 \tan^{-1}\left(0.333 \times 10^{-2} \theta\right) = -0.22067 \times 10^{-3} t - 2.9282 \qquad (4)$$

The solution to this nonlinear equation is

$$\theta = 647.57\,\text{K}$$

The values of the calculated temperature at $t = 480\,\text{s}$ as a function of step size are plotted in Figure 5.

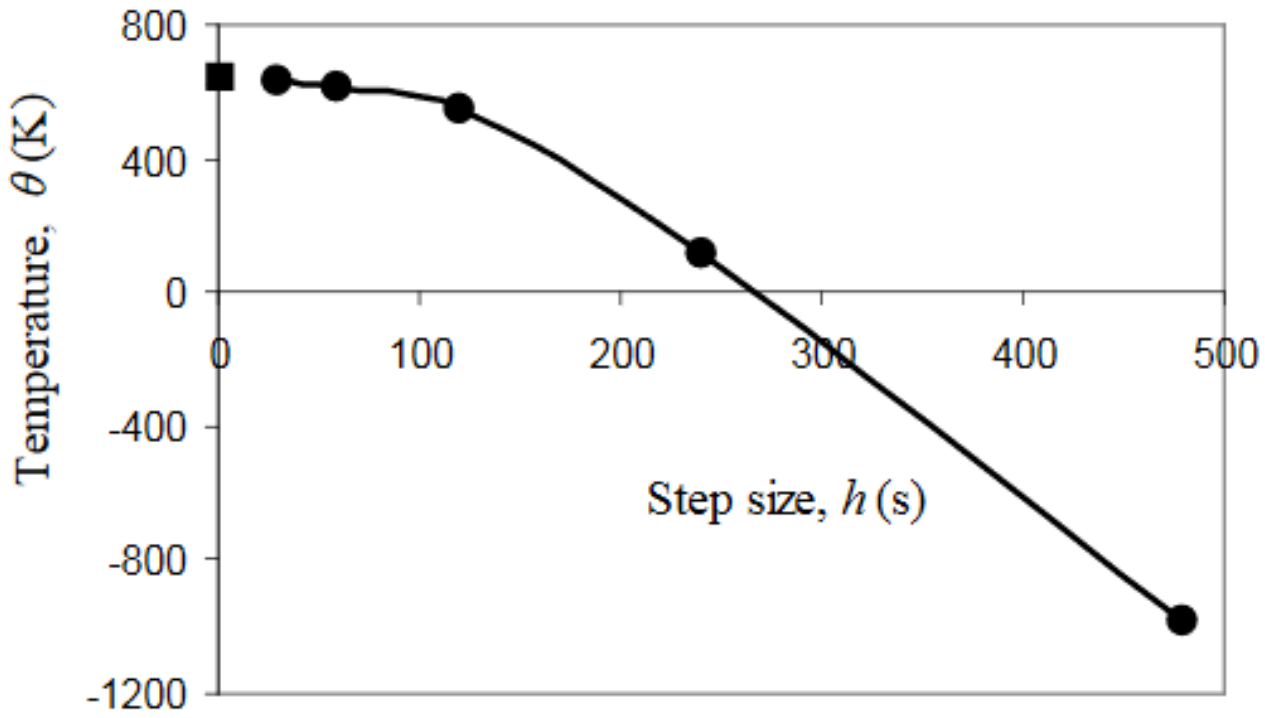
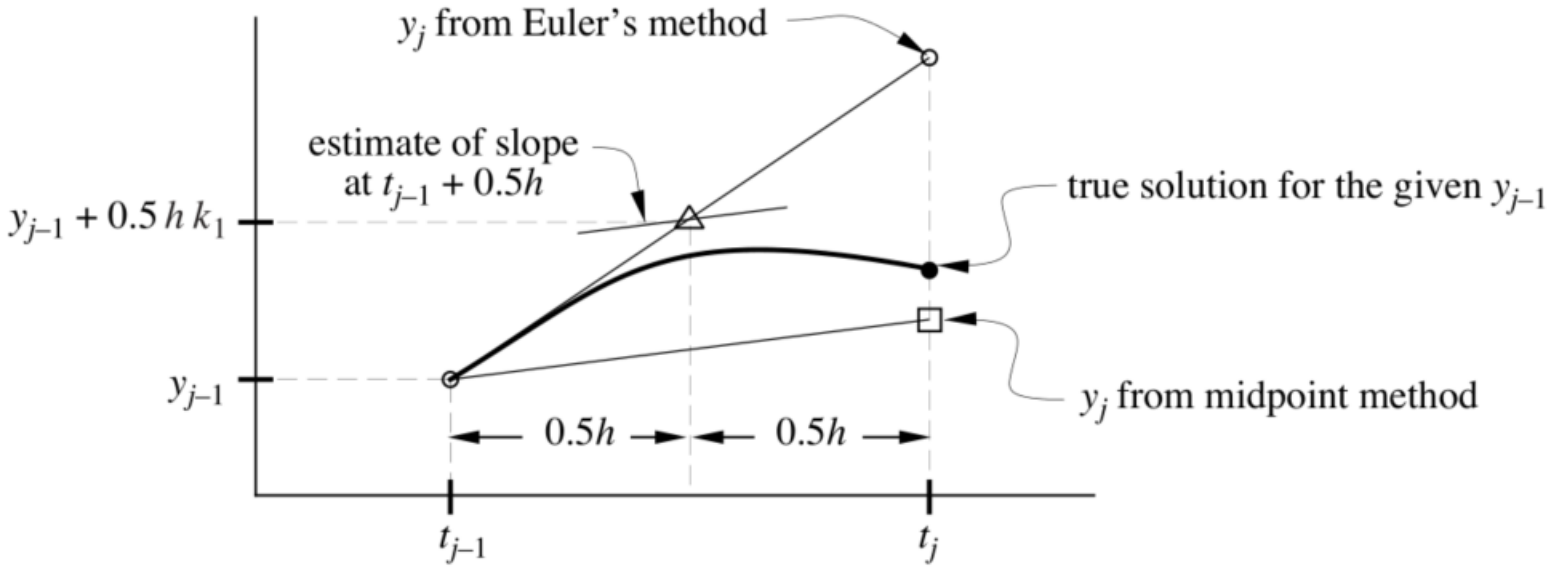
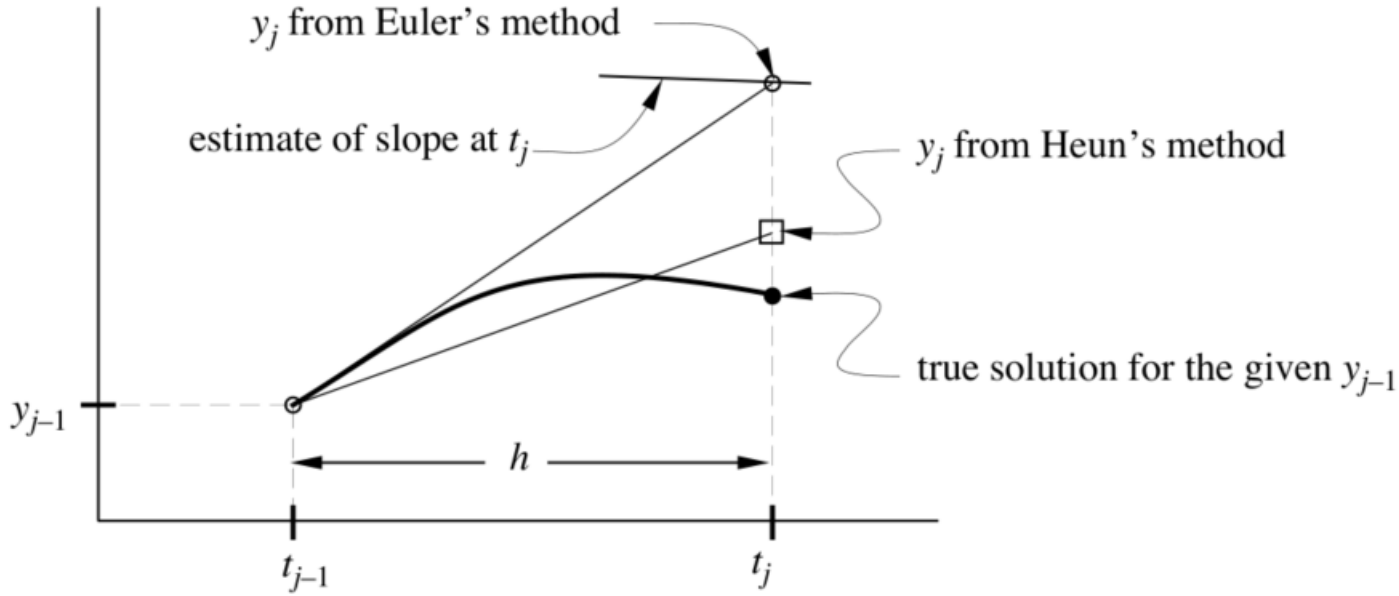


Figure 5 Effect of step size in Euler's method.

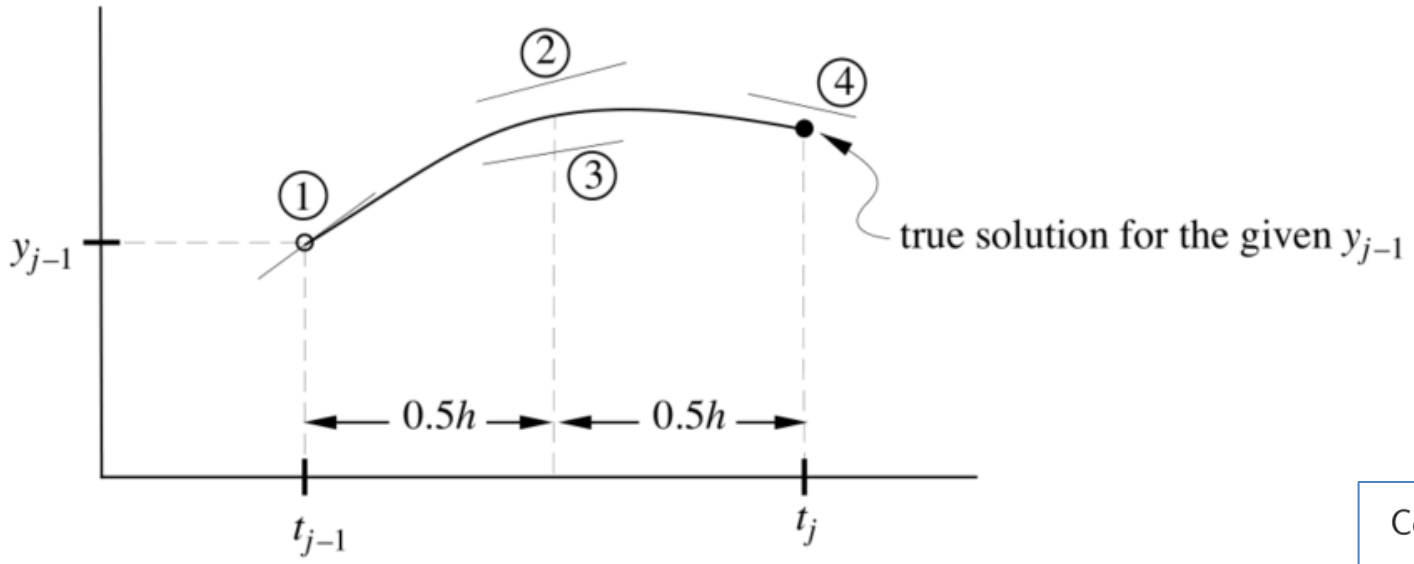
Higher Accuracy

2-POINT METHODS AND MORE


y_j from Euler's method
estimate of slope
at $t_{j-1} + 0.5h$
$y_{j-1} + 0.5\,h\,k_1$
y_{j-1}
true solution for the given y_{j-1}
y_j from midpoint method
$0.5h$
$0.5h$
t_{j-1}
t_j


y_j from Euler's method
estimate of slope at t_j
y_j from Heun's method
true solution for the given y_{j-1}
y_{j-1}
h
t_{j-1}
t_j

RK4



Compute slope at four places within each step

$$k_1 = f(t_j, y_j)$$
$$k_2 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_1)$$
$$k_3 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_2)$$
$$k_4 = f(t_j + h, y_j + hk_3)$$

Use weighted average of slopes to obtain y_{j+1}

$$y_{j+1} = y_j + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)$$

$$\text{LDE} = \text{GDE} = \mathcal{O}(h^4)$$

The values of the calculated temperature at $t = 480\,\text{s}$ as a function of step size are plotted in Figure 5.

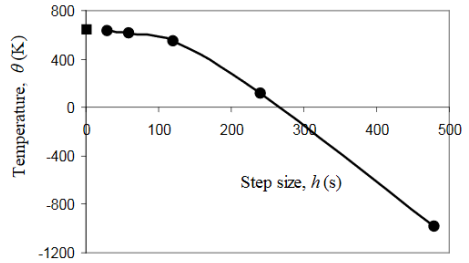


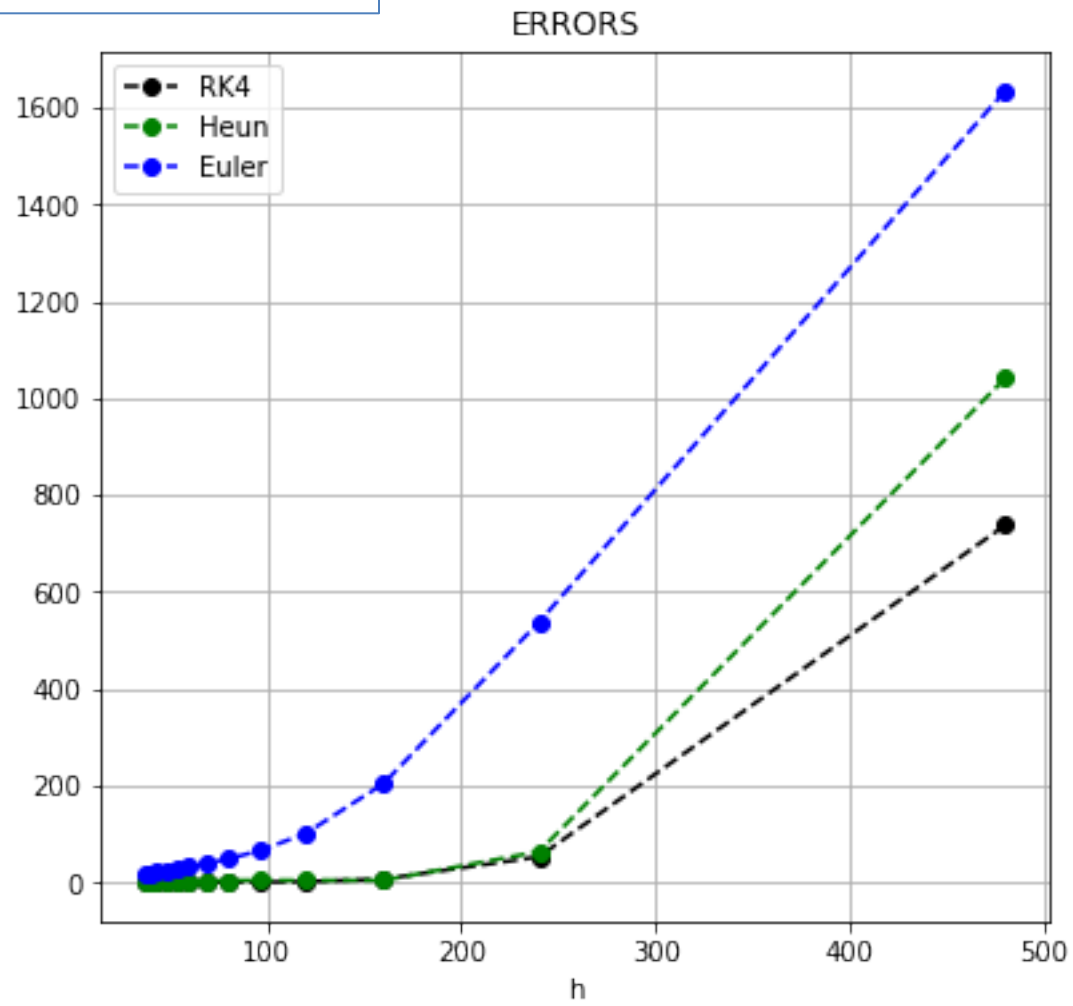
Figure 5 Effect of step size in Euler's method.

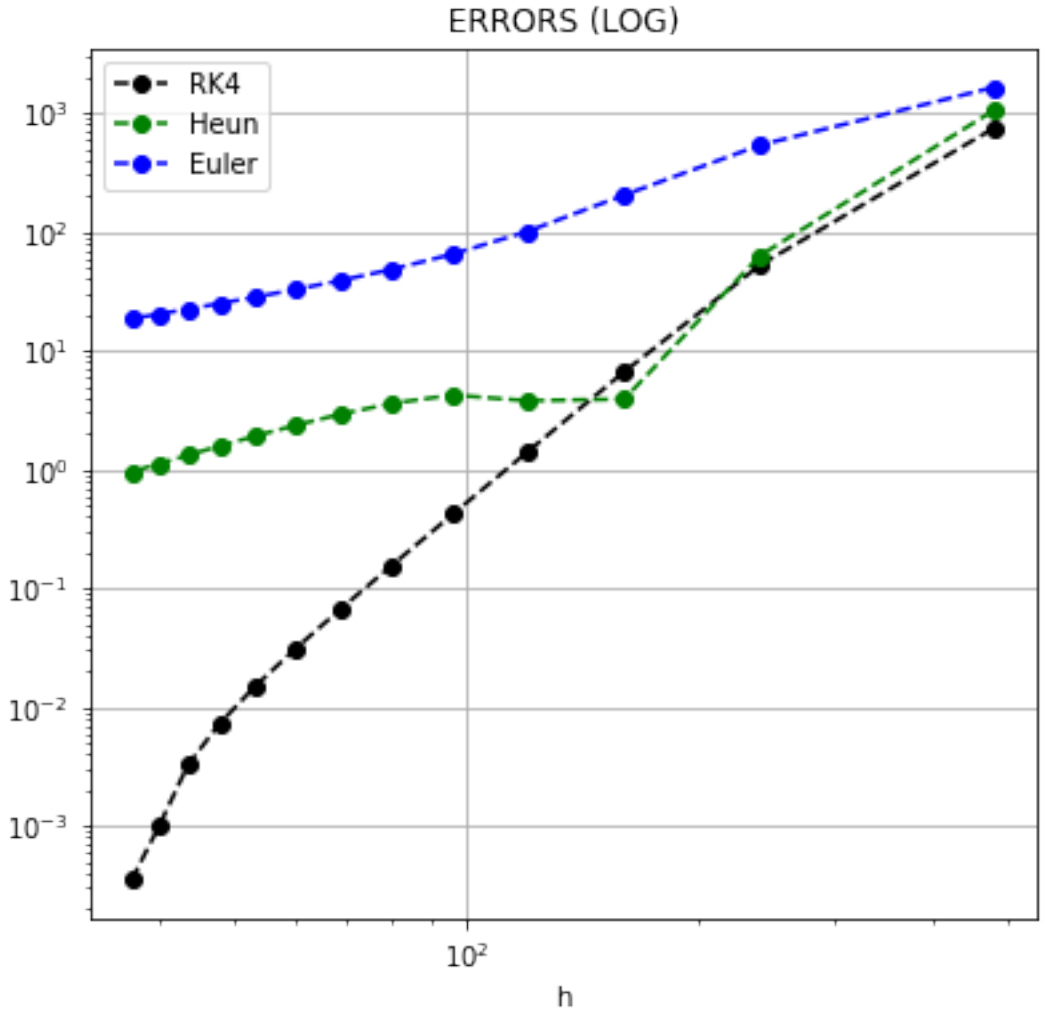
The exact solution of the ordinary differential equation is given by the solution of a non-linear equation as

$$0.92593\ln\frac{\theta-300}{\theta+300}-1.8519\tan^{-1}\left(0.333\times10^{-2}\,\theta\right)=-0.22067\times10^{-3}t-2.9282 \qquad (4)$$

The solution to this nonlinear equation is

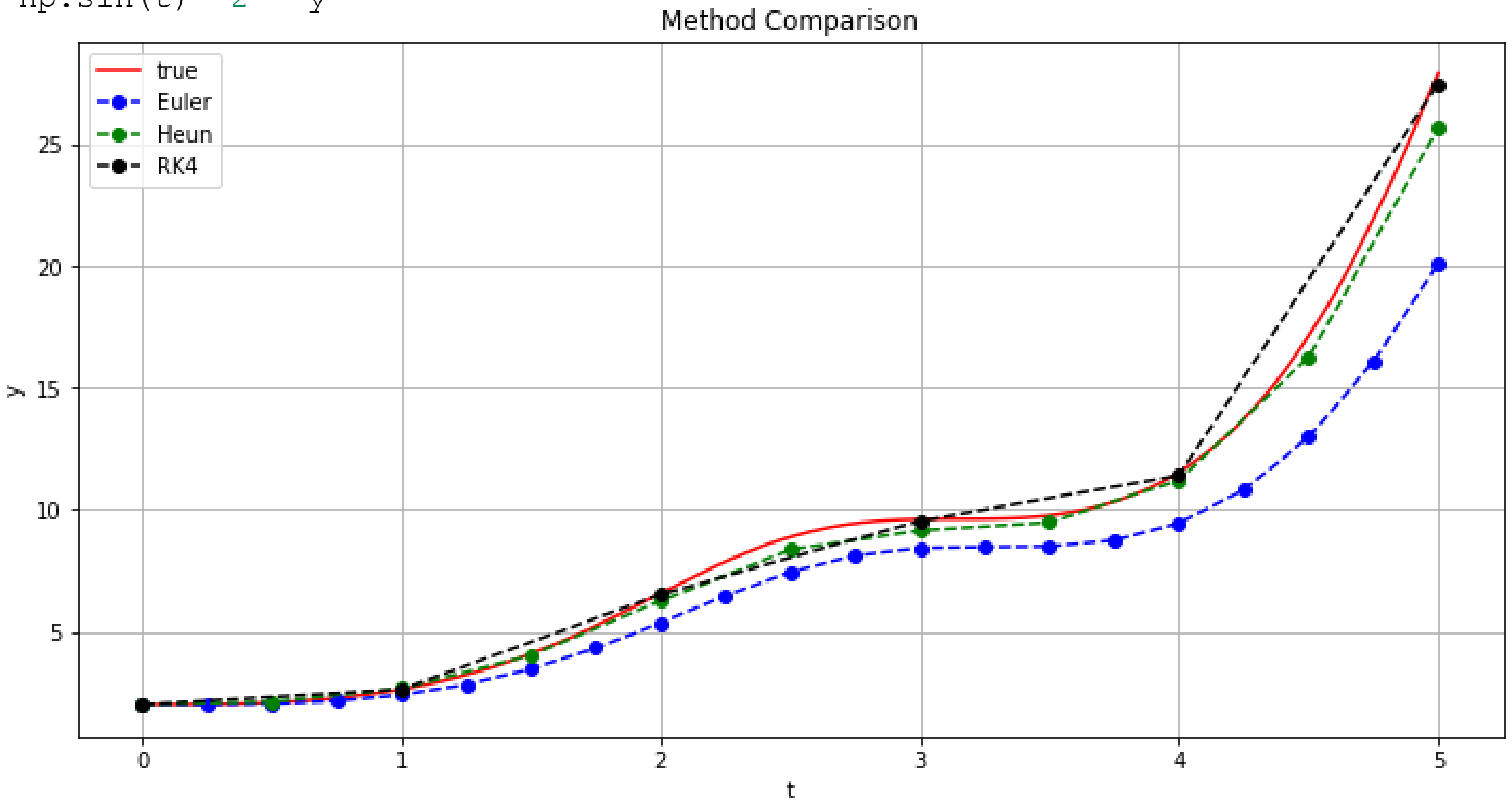
$$\theta = 647.57\,\text{K}$$



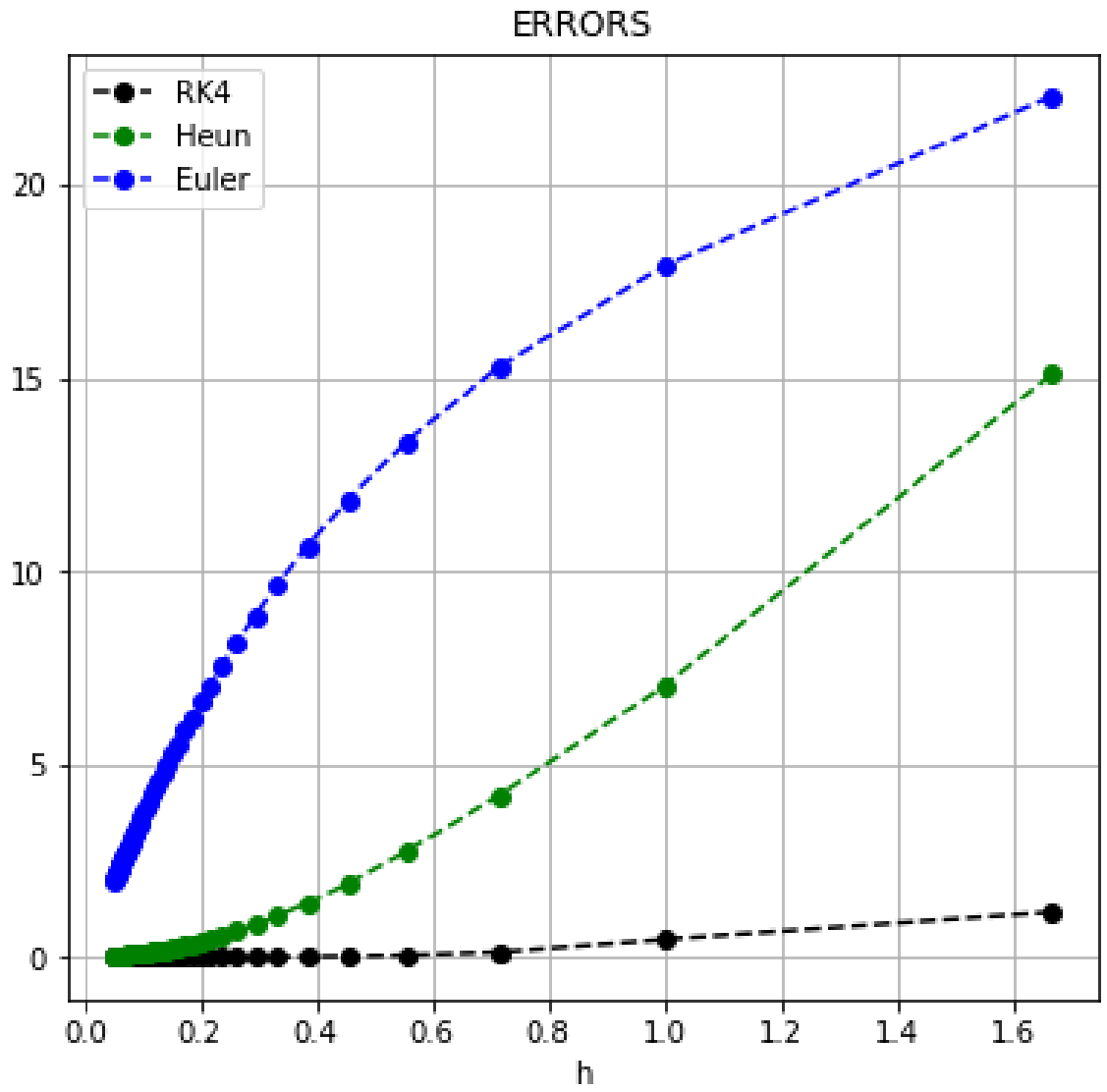


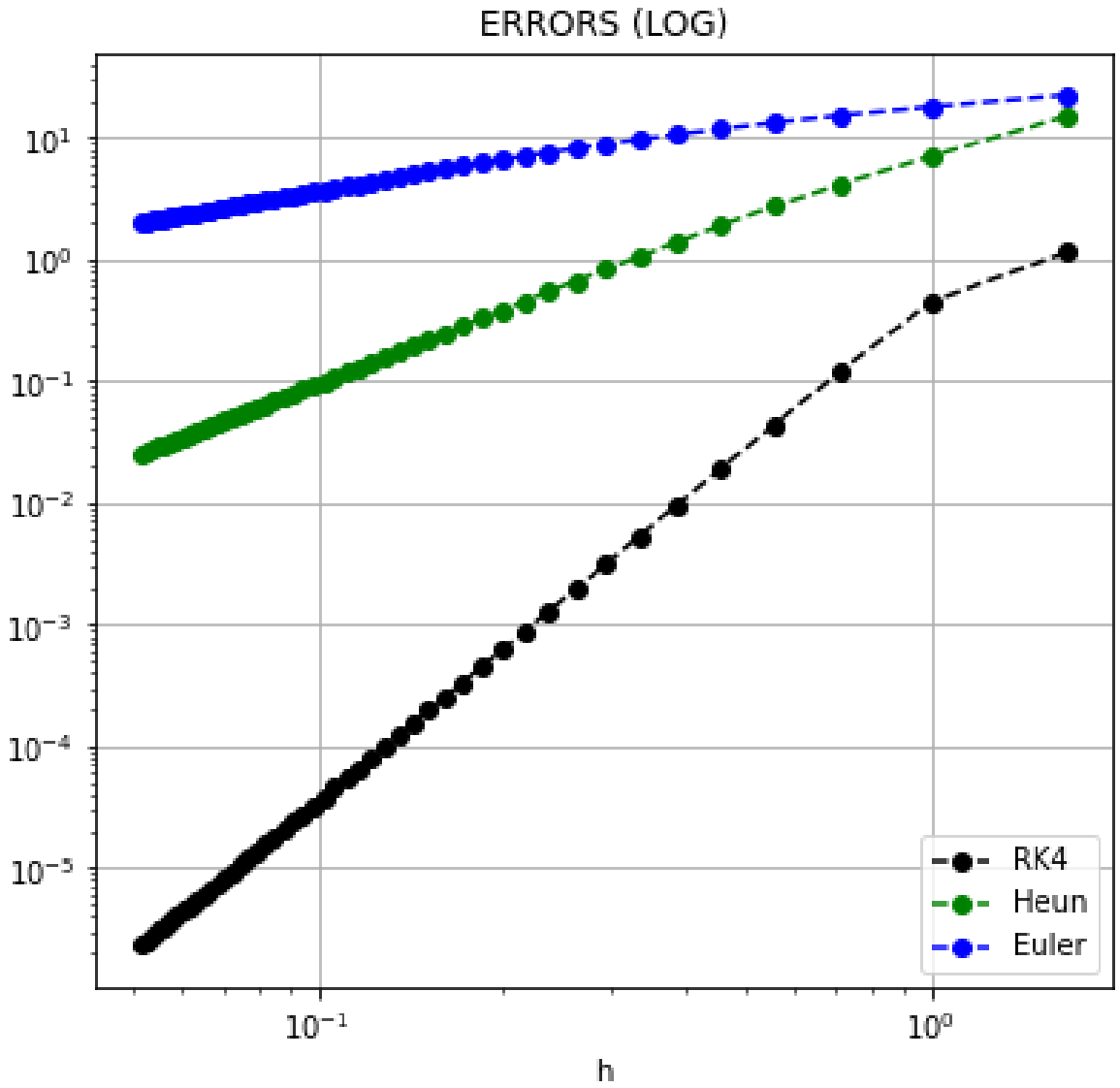
Method Comparison

```
def f(t,y):
   return np.sin(t)**2 * y
```



```python
def f(t,y):
  return np.sin(t)**2 * y
```





Butcher Tableau

https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods#Adaptive_Runge%E2%80%93Kutta_methods

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i,$$

where[6]

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f(t_n + c_2 h, y_n + (a_{21} k_1)h),\\
k_3 &= f(t_n + c_3 h, y_n + (a_{31} k_1 + a_{32} k_2)h),\\
&\vdots\\
k_s &= f(t_n + c_s h, y_n + (a_{s1} k_1 + a_{s2} k_2 + \cdots + a_{s,s-1} k_{s-1})h).
\end{aligned}$$

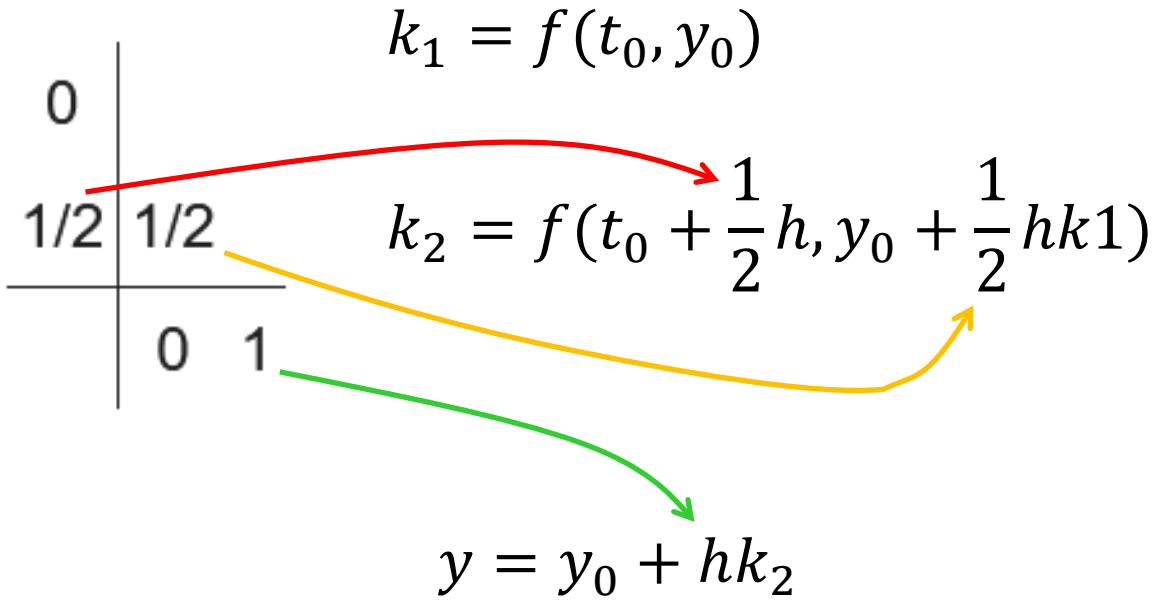
steps

$$\begin{array}{c|cccc}
0 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \vdots & & \ddots & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} \quad b_s
\end{array}$$

Remember that f is the derivative function $f = y'$

Butcher Tableau

Midpoint



RK4

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

$$k_1 = f(t_j, y_j)$$

$$k_2 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_1)$$

$$k_3 = f(t_j + \frac{h}{2}, y_j + \frac{h}{2}k_2)$$

$$k_4 = f(t_j + h, y_j + hk_3)$$

$$y_{j+1} = y_j + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)$$

Remember that f is the derivative function $f = y'$

Adaptive Steps RK45

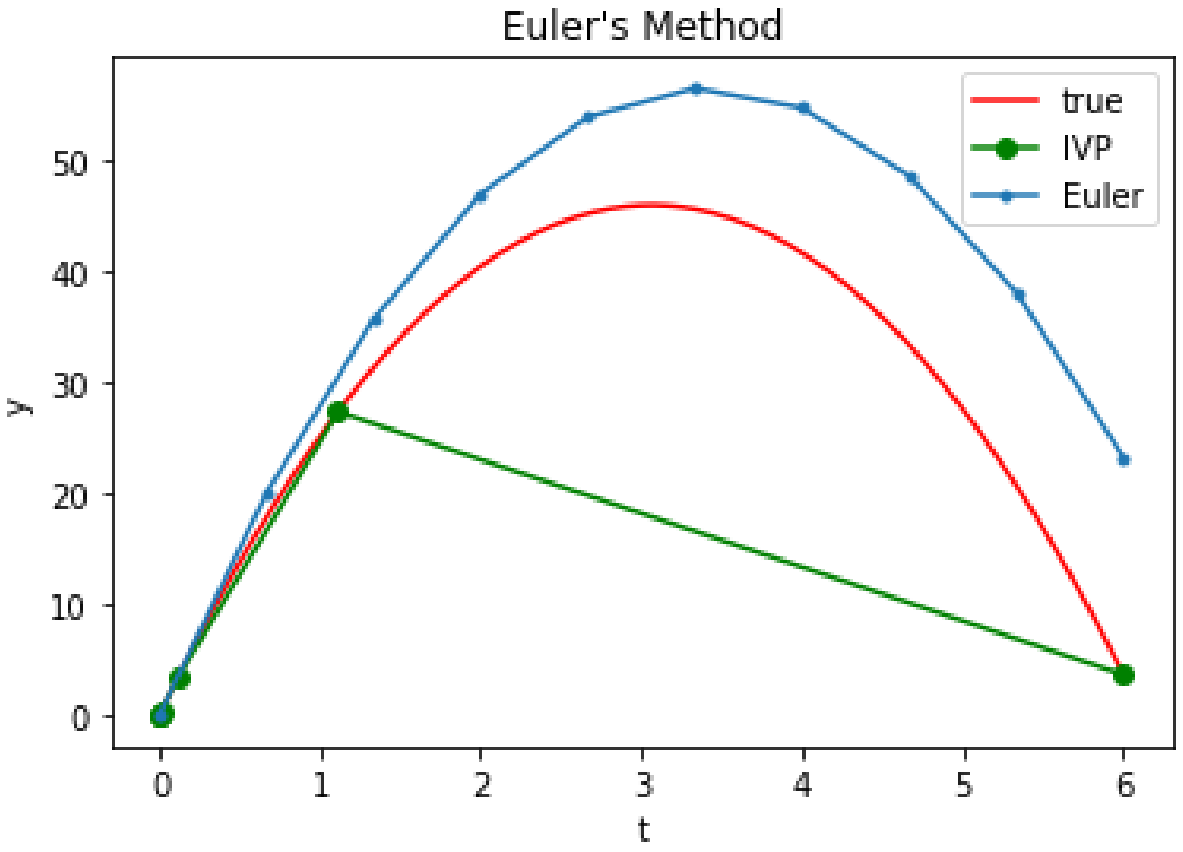
There are multiple ways of choosing coefficients. Scipy's solve_ivp uses Dormand-Prince (1980)

RK45

0							
1/5	1/5						
3/10	3/40	9/40					
4/5	44/45	−56/15	32/9				
8/9	19372/6561	−25360/2187	64448/6561	−212/729			
1	9017/3168	−355/33	46732/5247	49/176	−5103/18656		
1	35/384	0	500/1113	125/192	−2187/6784	11/84	
5^{th} order	35/384	0	500/1113	125/192	−2187/6784	11/84	0
4^{th} order	5179/57600	0	7571/16695	393/640	−92097/339200	187/2100	1/40

For each step, evaluate the function 5 times. Using a very clever combination of coefficients we can simultaneously find a 4^{th} order and 5^{th} order step. Comparing the difference gives an estimate of the step error. If the error is too large then reduce the step size and try again. Then use the estimate to calculate optimal location of next step.

RK45



Remember that solve_ivp uses adaptive steps! But they also make it easy to do 4th order interpolation between the points they've calculated.

If you really need to make a smooth graph you can turn this off and specify your own steps.

Application

SOLVE IVP

solve_ivp

https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

scipy.integrate.solve_ivp

scipy.integrate.solve_ivp(*fun*, *t_span*, *y0*, *method='RK45'*, *t_eval=None*, *dense_output=False*, *events=None*, *vectorized=False*, *args=None*, ***options*) [source]

Solve an initial value problem for a system of ODEs.

This function numerically integrates a system of ordinary differential equations given an initial value:

```
dy / dt = f(t, y)
y(t0) = y0
```

Here t is a 1-D independent variable (time), y(t) is an N-D vector-valued function (state), and an N-D vector-valued function f(t, y) determines the differential equations. The goal is to find y(t) approximately satisfying the differential equations, given an initial value y(t0)=y0.

Explicit Runge-Kutta methods ('RK23', 'RK45', 'DOP853') should be used for non-stiff problems and implicit methods ('Radau', 'BDF') for stiff problems [9]. Among Runge-Kutta methods, 'DOP853' is recommended for solving with high precision (low values of *rtol* and *atol*).
If not sure, first try to run 'RK45'. If it makes unusually many iterations, diverges, or fails, your problem is likely to be stiff and you should use 'Radau' or 'BDF'. 'LSODA' can also be a good universal choice, but it might be somewhat less convenient to work with as it wraps old Fortran code.

Example Code

Look at solveivp demos on Github:

https://github.com/mdaugherity/Numerical2024/blob/main/ode/Week_9_solveivp.ipynb

```
# Define derivative function
def yprime(t,y):
  k = 2.2067e-12
  a = 81e8
  return -k*(y**4 - a)

ivp = solve_ivp(yprime, [0,480],[1200])
t_ivp = ivp.t
y_ivp = ivp.y[0]

plt.title('Solve_ivp Example')
plt.plot(t_ivp,y_ivp,'s-',label='IVP')
plt.show()
```

Practice Problem

Given:

$$y' + 4y = t^2$$

with initial value $y(0) = 1$.

What is y at $t = 1.5$?

Exact solution:

$$y(t) = \frac{31}{32}e^{-4t} + \frac{1}{4}t^2 - \frac{1}{8}t + \frac{1}{32}$$

Two or more functions

COUPLED SYSTEMS

$$\frac{dp_1}{dt} = \alpha_1 p_1 - \delta_1 p_1 p_2$$

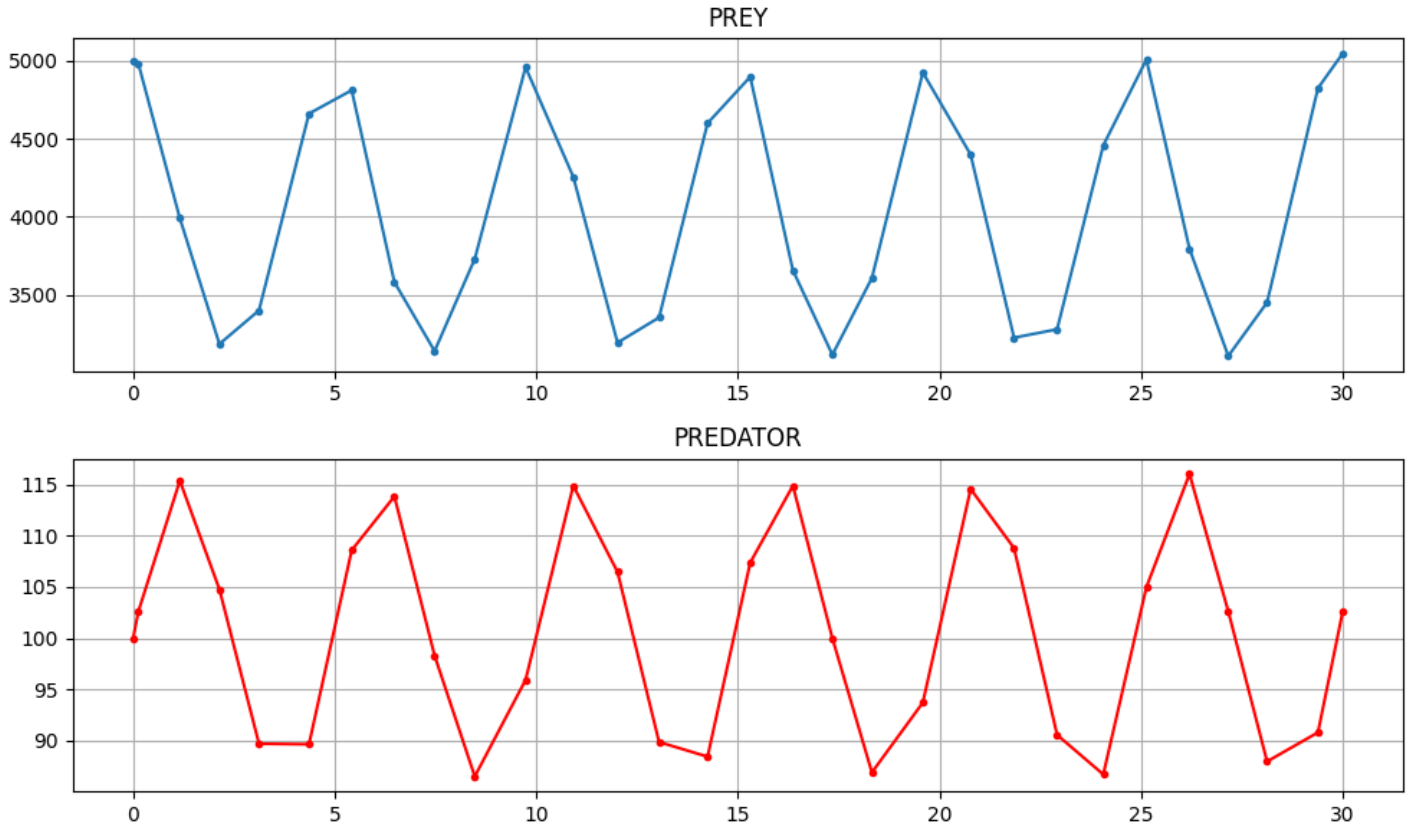
prey

$$\frac{dp_2}{dt} = \alpha_2 p_1 p_2 - \delta_2 p_2$$

predator

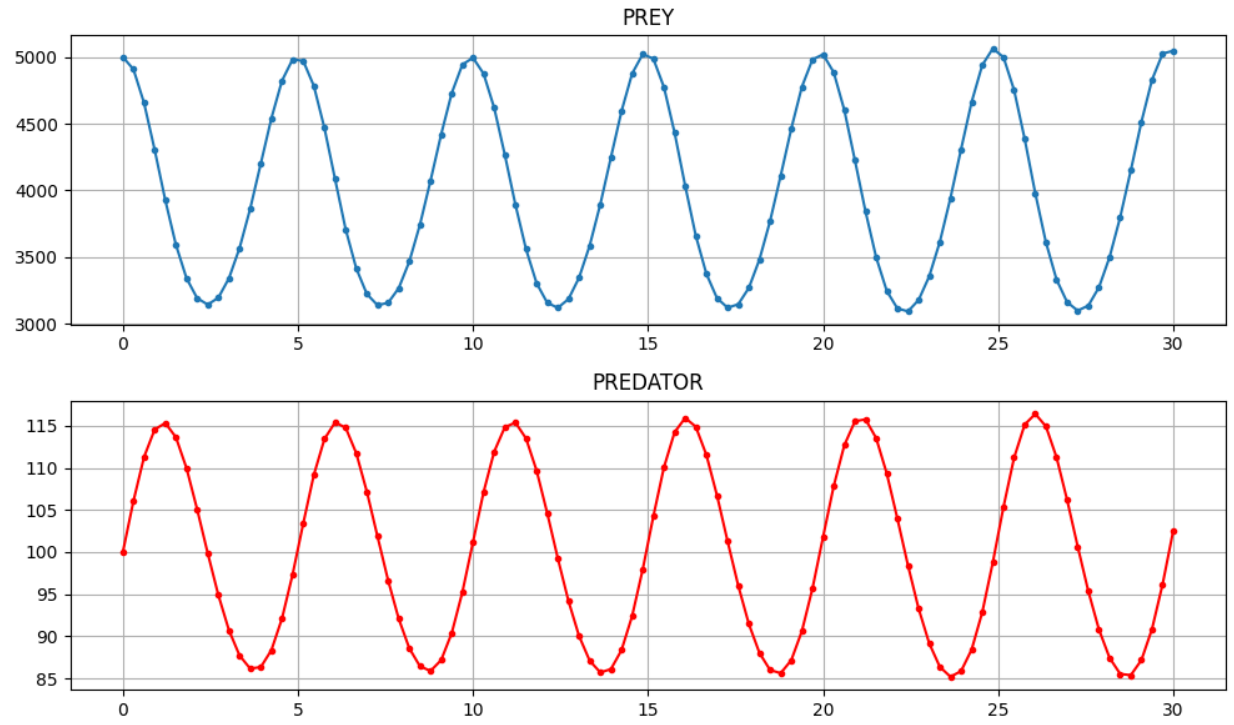
```
# PRED-PREY
# p1 = prey population, p2 = predator population
a1 = 2.0; a2 = 0.0002; # growth rates
d1 = 0.02; d2 = 0.8; # death rates

def f(t,yvec):
  y1 = yvec[0]  # define yvec=[y1,y2]=[prey, predator]
  y2 = yvec[1]
  return [a1*y1-d1*y1*y2,  a2*y1*y2 - d2*y2];
```



dense_output

```
1 ivp = solve_ivp(yprime, [0,TMAX],y0,dense_output=1)
2 print(ivp.message)
3 print('nfev = ',ivp.nfev)
4
5 # Interpolate values for plots
6 t = np.linspace(0, TMAX, num=100)
7 yint = ivp.sol(t)
8 prey_int = yint[0]
9 pred_int = yint[1]
```



Want a smooth graph?
Set **dense_output=True**

Then solve_ivp will return a function called sol that you can use to get interpolated values.

Any derivatives

HIGHER-ORDER PROBLEMS

Higher Order

Use “unrolling” trick:

any Nth order problem can be written as N first-order equations

Need N initial values to get started.

Practice Problem

Solve:

$$y'' + 4y = 4t$$

with $y(0) = 0$ and $y'(0) = 0$ from $t = 0$ to 2

2nd Order Example

Simple Harmonic Oscillator: $F = -\mathrm{kx} = \mathrm{ma} \longrightarrow \dfrac{d^2x}{\mathrm{dt}^2} = -\left(\dfrac{k}{m}\right)x$

$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ where $y_1 = x$ and $y_2 = \dfrac{\mathrm{dy}_1}{\mathrm{dt}}$(or velocity)

The ODE functions are $\vec{f} = \begin{bmatrix} \mathrm{dy}_1/\mathrm{dt} \\ \mathrm{dy}_2/\mathrm{dt} \end{bmatrix} = \begin{bmatrix} y_2 \\ -\dfrac{k}{m}y_1 \end{bmatrix}$

Obviously (you better know this!) the right solution is $y = A\cos(\omega t)$ where $\omega^2 = \dfrac{k}{m}$

Just for fun, a coupled second-order system!

Consider a solar system with a (stationary) sun at the origin being orbited by a single planet. Can we integrate Newton's Laws and see if we get an elliptical orbit?

Define $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x \\ v_x \\ y \\ v_y \end{bmatrix}$ remembering that $v_x = \frac{\mathrm{dx}}{\mathrm{dt}}$ and $v_y = \frac{\mathrm{dy}}{\mathrm{dt}}$

Gravitational force $F = \frac{G\,m\,M}{r^2}$ with distance is $r = \sqrt{x^2 + y^2}$.

Acceleration $a = \frac{F}{m} = \frac{G\,M}{r^2}$ pointing towards the origin. The x any components are:

$$a_x = -a\cos\theta = -a\left(\frac{x}{r}\right) = -\frac{G\,M\,x}{r^3} \quad a_y = -a\sin\theta = -a\left(\frac{y}{r}\right) = -\frac{G\,M\,y}{r^3}$$

Finally, our derivative function is $\vec{f} = \frac{d\vec{y}}{\mathrm{dt}} = \begin{bmatrix} v_x \\ -\frac{G\,M\,x}{r^3} \\ v_y \\ -\frac{G\,M\,y}{r^3} \end{bmatrix}$

Practice Problem

An object is in free-fall. Assume the force of air resistance is:

$$\boldsymbol{F_D = Av^2 e^{-By}}$$

The object's acceleration is: $\boldsymbol{a = -g + \frac{F_D}{m}}$

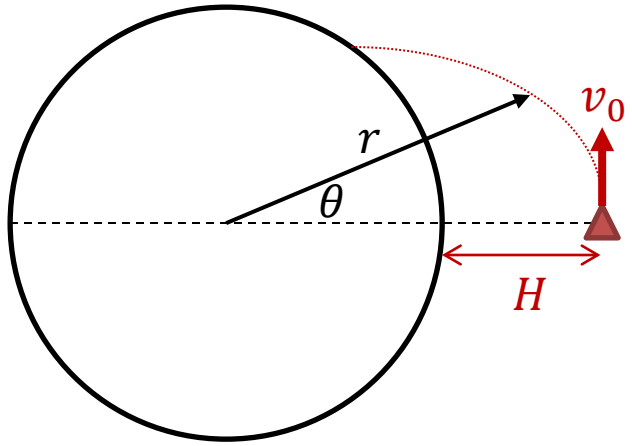
Use constants $A = 7.45\,\frac{kg}{m}$, $B = 1.053 \times 10^{-4}\,m^{-1}$, $m = 114\,kg$.

1) Given initial conditions $y(0) = 9{,}000\,m$ and $v(0) = 0$, how far does it fall in 10 seconds with no air resistance?
2) How far with air resistance?

Practice Problem

The 2D equations of motion for a spacecraft in flight are:

$$\ddot{r} = r\dot{\theta}^2 - \frac{GM}{r^2} \qquad \ddot{\theta} = -\frac{2\dot{r}\dot{\theta}}{r}$$



Constants are:

$G = 6.672 \times 10^{-11}$ $m^3kg^{-1}s^{-2}$	Newton's Gravitational Constant
$M = 5.9742 \times 10^{24}$ kg	Mass of Earth
$R_e = 6378.14$ km	Radius of Earth at sea level

Initial Conditions:

$H = 772$ km, $\mathrm{r} = \mathrm{R_e} + \mathrm{H}$, $\theta = 0$, $v_0 = 6700$ m/s, $\dot{\theta} = \frac{v_0}{r}$

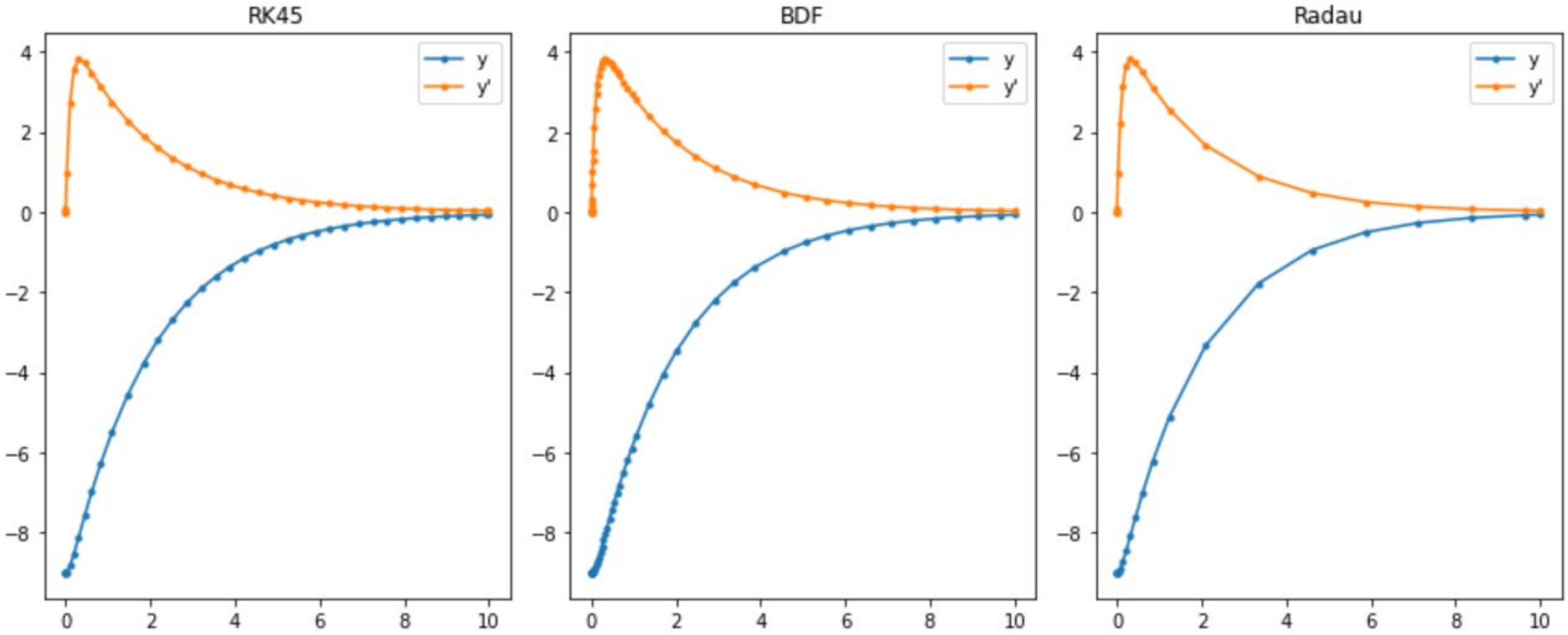
When and where does it "land"?

What to do when things break

STIFF SYSTEMS

```
def f(t,y):
  return [y[1], -4.75*y[0] - 10.0*y[1]]
```

Example 7.7

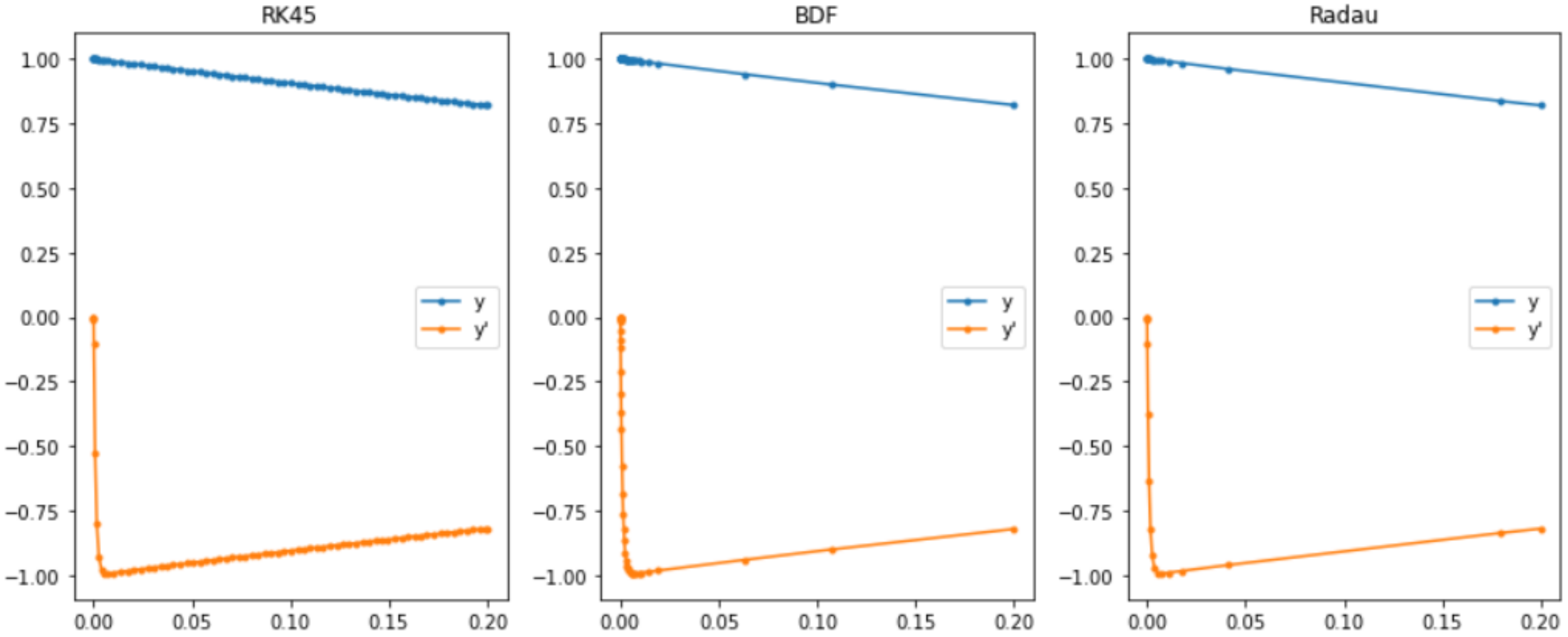


- RK45 uses way too many points during the smooth region x>2
- Notice how BDF makes extra points when y' is changing quickly, then it speeds up
- Radau makes fewer overall points but more function evaluations than BDF

```
def f(t,y):
  return [y[1], -1001*y[1] - 1000*y[0]]

y0 = [1,0]
```

Problem 7.2.9



RK45 is really struggling! It took 428 function evals compared to 74 for BDF

Richardson Equations

In 1966 a chemist published a set of reaction rate equations which later became a famous example as a difficult numerical system. Consider 3 chemicals A,B, and C in reactions:

$$\begin{aligned} A &\rightarrow B \\ 2B &\rightarrow B + C \\ B + C &\rightarrow A + C \end{aligned}$$

where the reactions proceed at rates k_1, k_2, and k_3. The rate of change is

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} -k_1 A + k_3 BC \\ k_1 A - k_2 B^2 - k_3 BC \\ k_2 B^2 \end{bmatrix}$$

The rates are measured to be $k_1 = 0.04, k_2 = 3 \cdot 10^7, \text{and } k_3 = 10^4$. The initial conditions are $A = 1, B = 0, C = 0$. Note that the chemicals will have different y-axis ranges. The time range is [0, 400]

RK45 will fail miserably, but BDF and Radau do great!

Things to Know

- Understand concepts behind Euler and higher-order methods
- “Unrolling” for higher-orders
- Coupled systems
- Know how to use `scipy.integrate.solve_ivp`
  - dense_output to get interpolation
  - RK45 normal, DOP853 for high precision, BDF for stiff
  - events!