Module 4

Lecture 5

More accurate numerical solvers for ODEs

Sarah Groves, Ph.D.

Shayn Peirce-Cottler, Ph.D.



Why do we have numerical error?

- When we can't solve an ODE analytically, we must approximate a solution numerically
- We can think back to calculus of how to approximate a function: Taylor series!
- F(x) is **equal to** an **infinite sum** of terms including the **derivatives** of the function:

$f(x) = y_{i+1}$ $\quad f(a) = y_i$

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

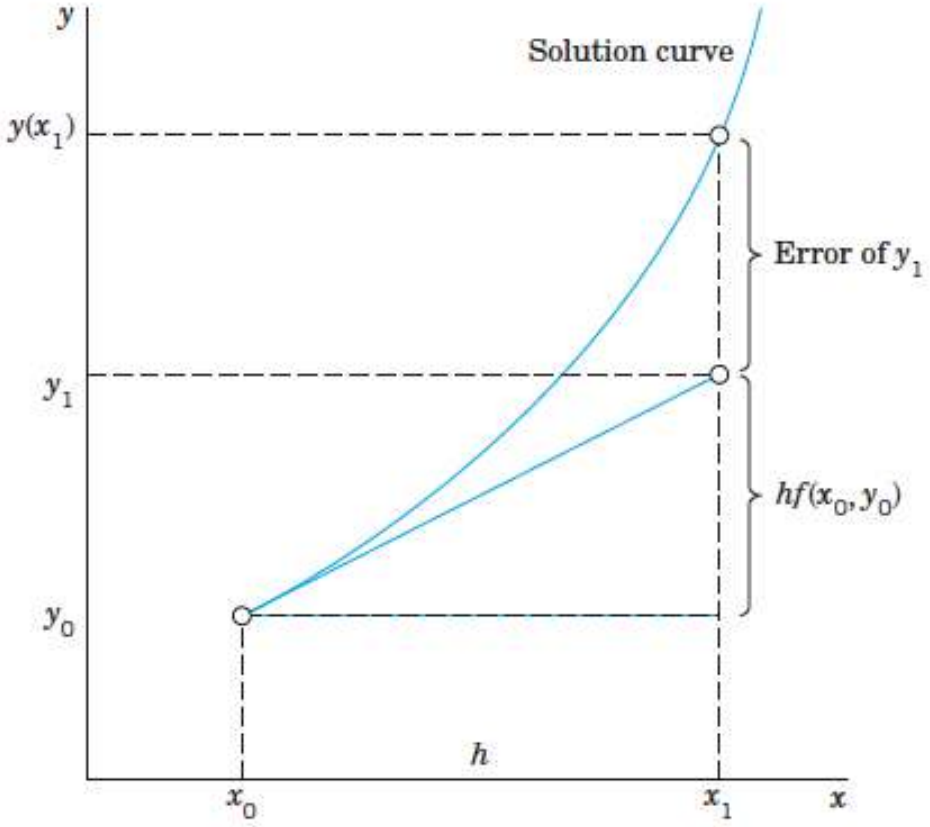
Interval ($h=x_{i+1}-x_i$)

$$y_{i+1} = y_i + hf'(t_i, y_i)$$

- Better approximating higher order terms = lower (truncation) error from numerical method

Predicting the future

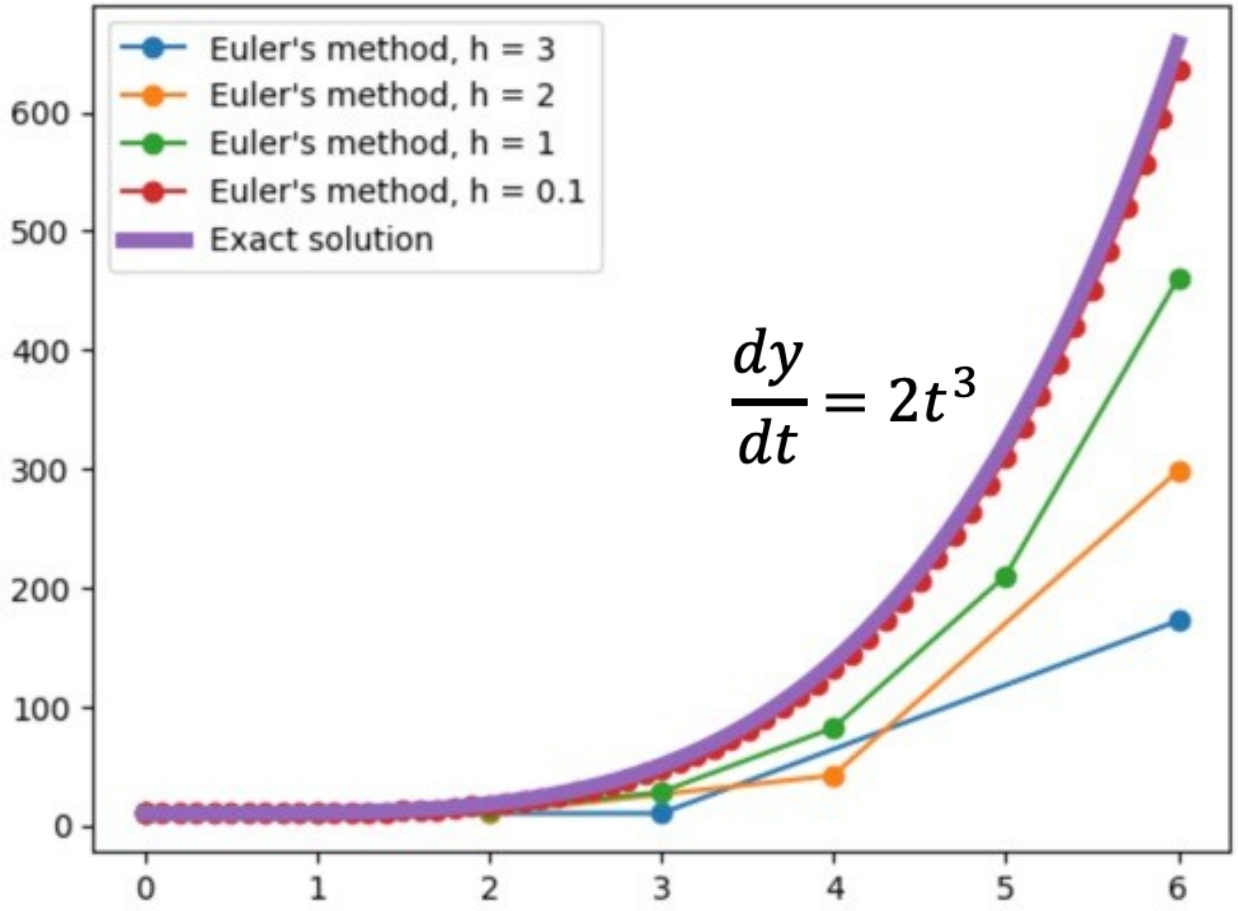
- What is the error associated with our predictions?

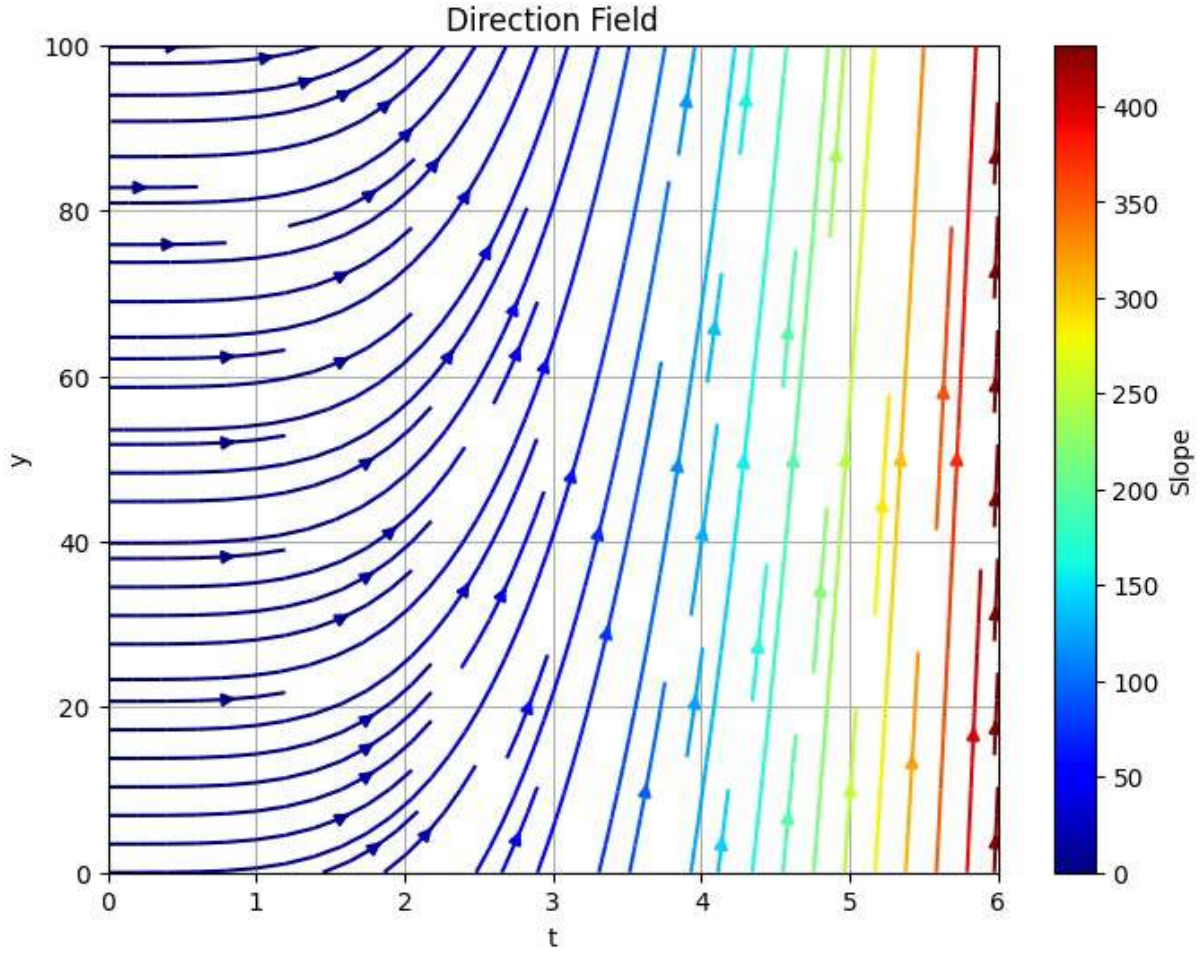


Euler's Method

- Approximate solution **using slope at current location**
- Error depends on step size, and we decrease error with smaller step size

Choosing step size and comparing error





Why are the predictions for y_{i+1} underestimating the true

We can get a better estimate by approximating the slope differently

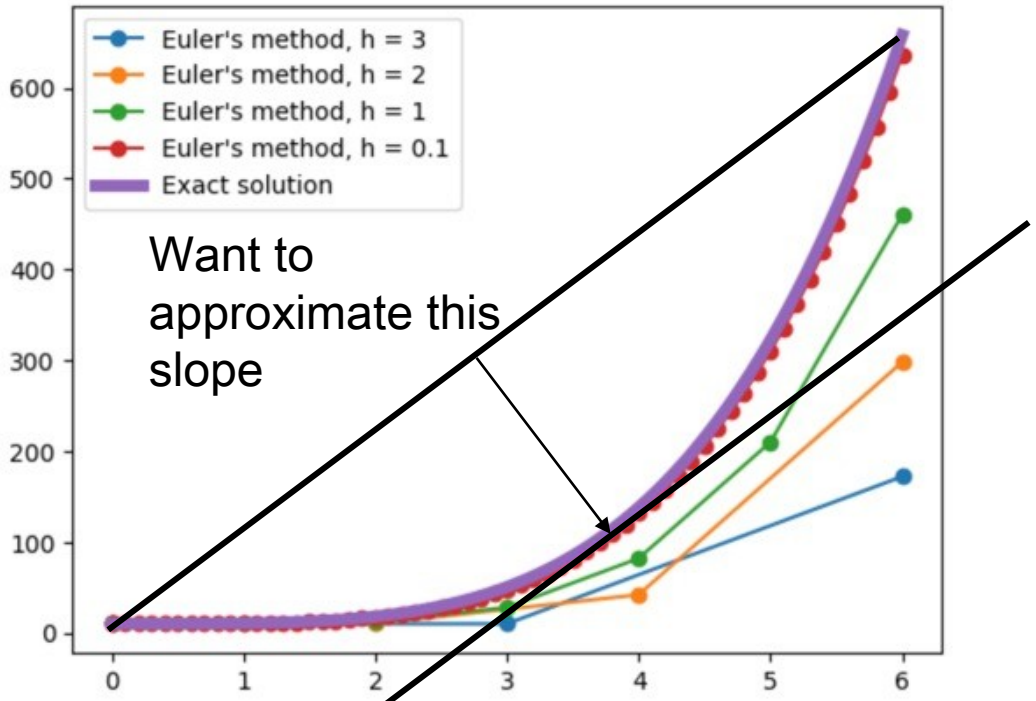
ODE Form: $\frac{dy}{dt} = f(t, y)$

General Form:

$$y_{i+1} = y_i + \phi h$$

$$\phi = f(t_i, y_i)$$

Let's use a different function ϕ

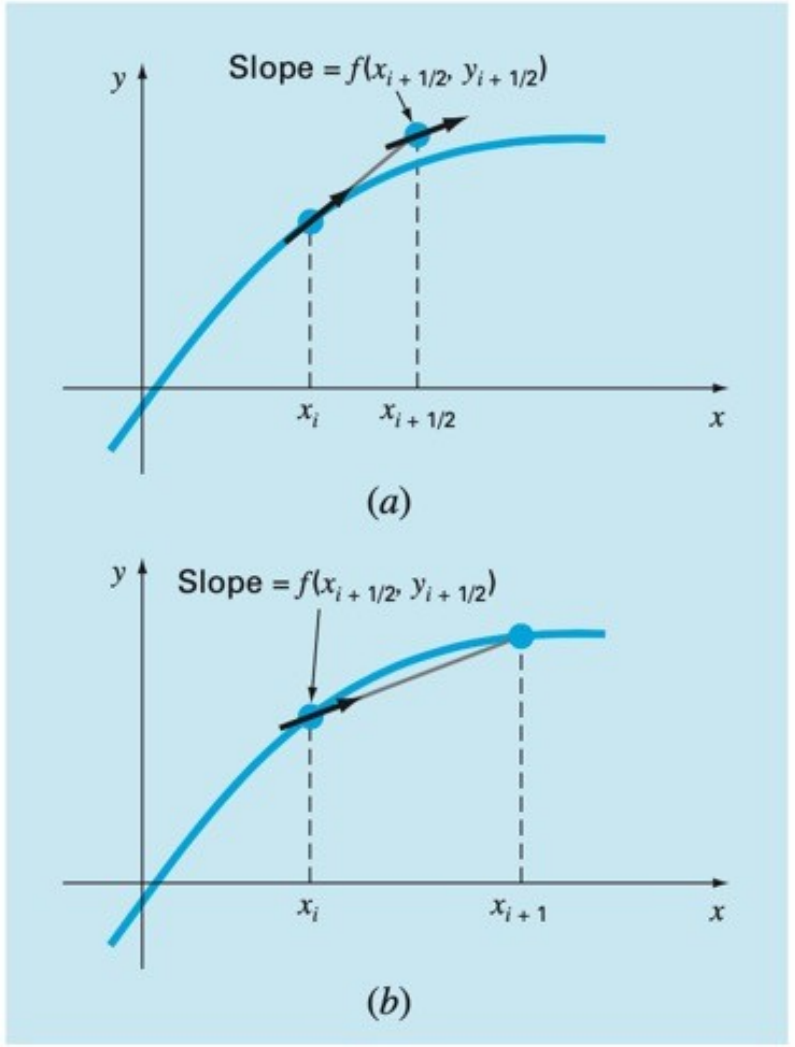


But we usually only know (x_i, y_i) and x_{i+1}

$$s = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

In this case,
S_i is too small
S_{i+1} is too big

Midpoint Method (2nd order Runge Kutta)



Recap: Euler's method (RK1) and Midpoint method (RK2)

- **Euler's Method**
  - Approximate solution **using slope at current location**
  - Error depends on step size
    - Throws out Taylor series terms higher than 1st order
    - Decrease step size by 2 → decrease error by 2

ODE Form: $\frac{dy}{dt} = f(t, y)$

General Form:

$$y_{i+1} = y_i + \phi h$$
$$\phi = f(t_i, y_i)$$

- **Midpoint Method**
  - Approximate solution **using slope at current location** and **approximate slope at midpoint of h interval**
  - Decrease step size by 2 → decrease error by 4!

ODE Form: $\frac{dy}{dx} = f(x, y)$

General Form: $\phi = f\left(x_{i+1/2}, y_{i+1/2}\right)$

$$y_{i+1/2} = y_i + f(x_i, y_i)\frac{h}{2}$$

$$y_{i+1} = y_i + f\left(x_i + \frac{h}{2}, y_{i+1/2}\right)h$$

Runge Kutta Methods: generalized form

$$y_{i+1} = y_i + \phi(x_i, y_i, h)h$$

n = 1 → Euler's method $y_{i+1} = y_i + \phi h$, $\phi = f(t_i, y_i)$

n = 2 with a_2=1 → Midpoint method

$$\phi = a_1k_1 + a_2k_2 + \cdots + a_nk_n$$

$$k_1 = f(x_i, y_i)$$
$$k_2 = f(x_i + p_1h, y_i + q_{11}k_1h)$$
$$k_3 = f(x_i + p_2h, y_i + q_{21}k_1h + q_{22}k_2h)$$
$$\vdots$$
$$k_n = f(x_i + p_{n-1}h, y_i + q_{n-1,1}k_1h + q_{n-1,2}k_2h + \cdots + q_{n-1,n-1}k_{n-1}h)$$

a = weight
k = derivative evaluated at some point

$$y_{i+1} = y_i + k_2h$$

where

$$k_1 = f(x_i, y_i)$$
$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1h\right)$$

4th order Runge Kutta Method

- Most popular method
- Infinite number of versions, but the most common is the **classical 4th order RK method**
- a_i weights and k functions are chosen to cancel out terms in the Taylor series, giving lower error for the approximation

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

where

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_i + h, y_i + k_3 h)$$

Runge-Kutta 4th Order

ODE Form: $\frac{dy}{dt} = f(t, y)$

General Form:

$$y_{i+1} = y_i + \phi h$$

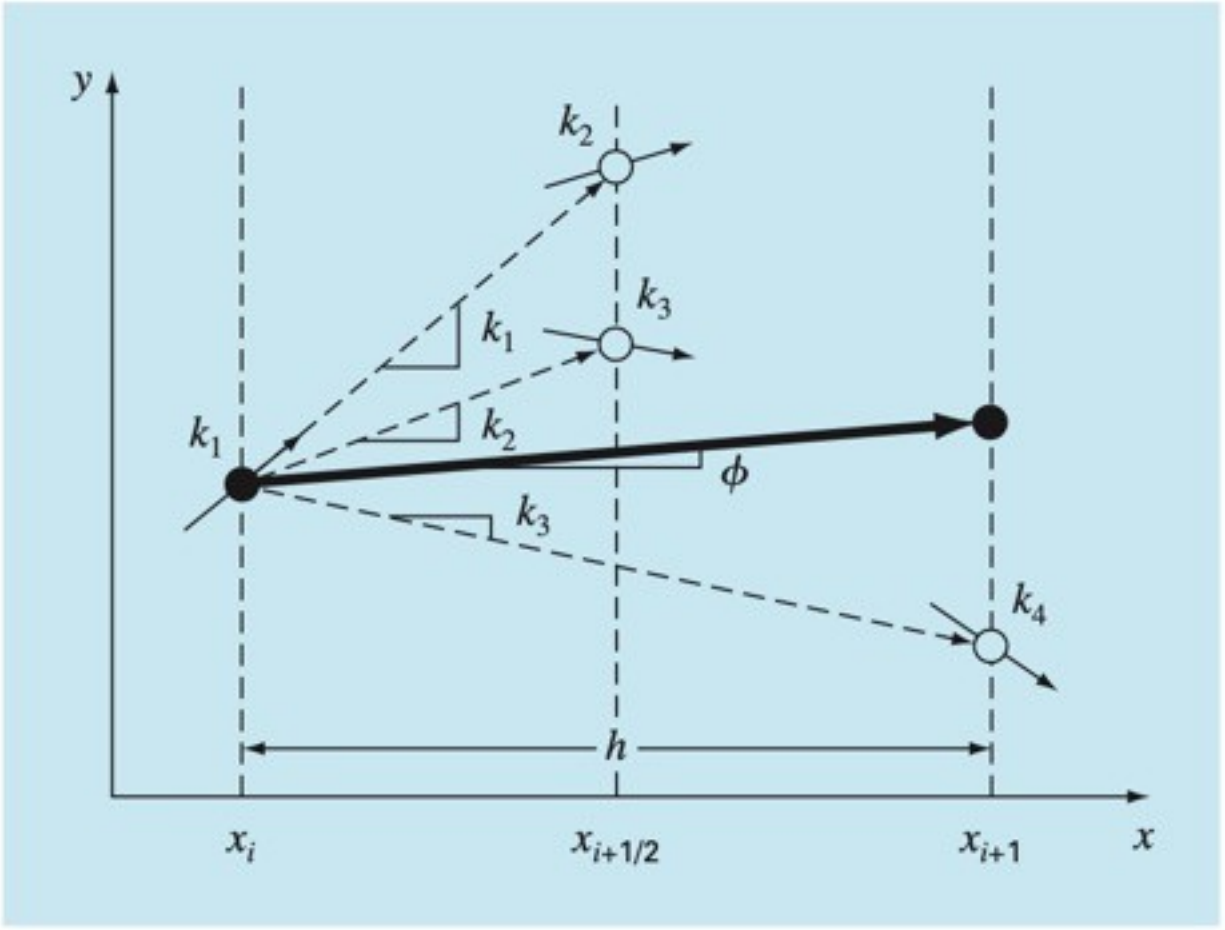
$$\phi = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(t_i, y_i)$$

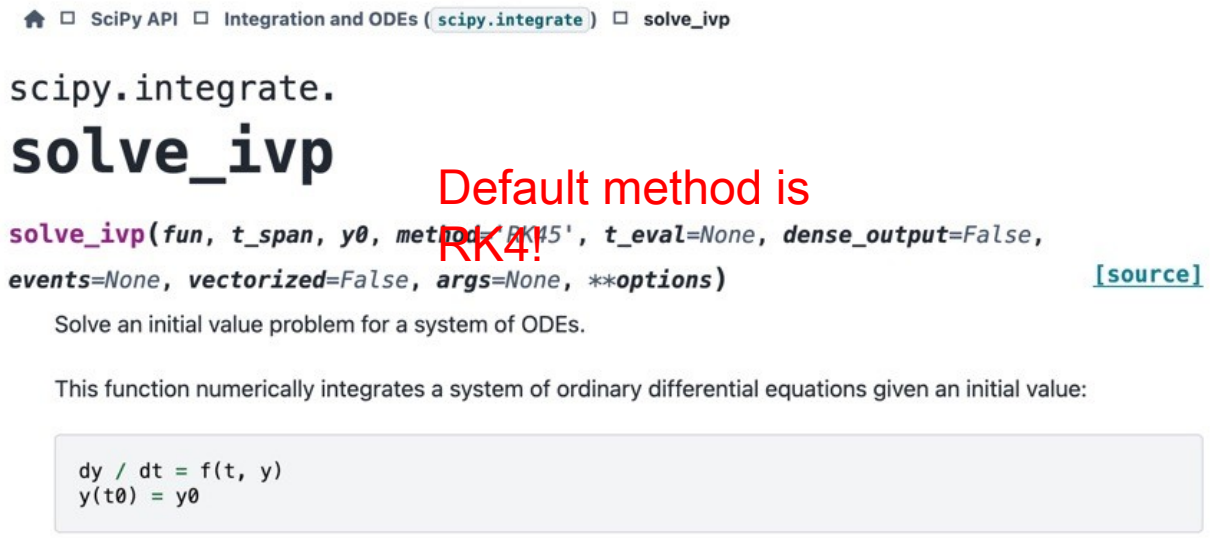
$$k_2 = f\left(t_i + \frac{h}{2}, y_i + k_1\frac{h}{2}\right)$$

$$k_3 = f\left(t_i + \frac{h}{2}, y_i + k_2\frac{h}{2}\right)$$

$$k_4 = f(t_i + h, y_i + k_3 h)$$



Implementing RK4 in Python



- fun: system of ODEs
- t_span: Interval of integration (two numbers, t0 and tf)
- y0: Initial state for system (S0, I0, and R0) in an array

- We could call solve_ivp a functional: function of a function
- fun is a function that takes in t, S, I and, R and outputs dS, dI, and dR (i.e. the system of ODEs)

```
#EXAMPLE
import numpy as np
from scipy.integrate import solve_ivp
def exponential_decay(t, y): return -0.5 * y
sol = solve_ivp(exponential_decay, [0, 10], [2])
print(sol.t)
print(sol.y)
```

t_span

y0

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

In class work time

- **Tuesday at midnight:** All code for your final project (ipynb and pdf) should be SUBMITTED TO CANVAS (both file types)
  - You do not need to submit other python files, but everything should already be run so we can see your output
- This module “tells a story” rather than explores a single aim, and you can structure your project report that way as well
  - Ensure your conclusions recommend interventions, report any errors calculated, guess viral family, and report epidemic parameters from your best fit model