National Institute of Technology, Karnataka




# Numerical Solution of Ordinary Differential Equations

Presented by-
- Gavali Deshabhakt Nagnath (202CD005)

Course Instructor: Dr. Chandhini G.
Subject: Numerical Methods
Specialization: CDS
Date: 20/05/2021

# Definition & Terminology



**Ordinary differential equation** (ODE)
Differential equations that involve only **ONE** independent variable are called ordinary differential equations.

**Examples:**

$$\frac{dy}{dx} + 5y = e^x \qquad \frac{d^2y}{dx^2} - \frac{dy}{dx} + 6y = 0 \qquad \frac{dx}{dt} + \frac{dy}{dt} = 2x + y$$

➔**only *ordinary* (or *total* ) derivatives**

# Order & Degree



- The order of a differential equation is determined by the highest-order derivative
- the degree of a differential equation is power of highest order derivative

$$\frac{d^2y}{dx^2} + \left(\frac{dy}{dx}\right)^2 = 5$$

$$\text{Order} = 2 \qquad \& \qquad \text{degree} = 1$$

# Initial Value Problem



$$\frac{dy}{dx} = f(x, y) \quad \text{- (1)} \qquad y_0 = f(x_0) \quad \text{-(2)}$$

- Equation (1) and (2) together is called an Initial value Problem
- In order to select a particular integral from the infinite family of solution curves that constitute the general solution to (1), the differential equation will be considered in tandem with an initial condition i.e. (2)

# Method's Of Solving ODE



- Euler's Method
- Runge-kutta methods
  - One-stage RK method
  - Two-stage RK method
    - Modified Euler Method
    - Improved Euler Method
  - Four-stage RK method

# Euler's Method



$$\frac{dy}{dx} = f(x, y) \quad , a \leq x \leq b$$

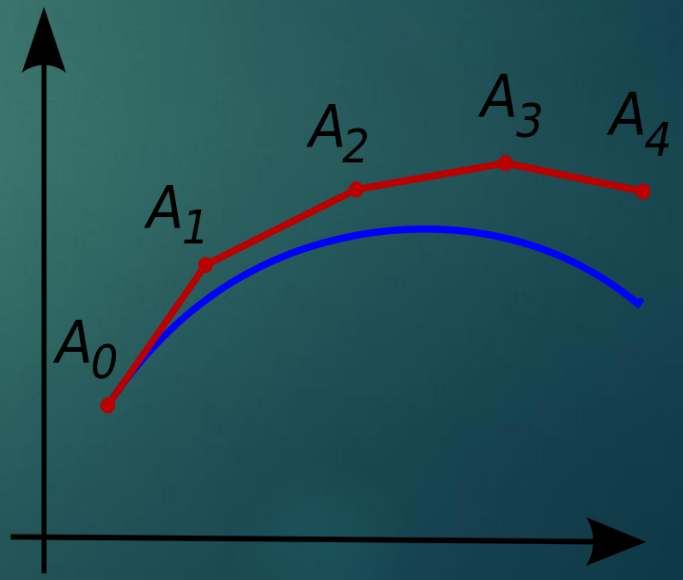
$$y_0 = f(x_0)$$

Iterative Formula

$$y_{n+1} = y_n + hf(x_n, y_n)$$

where, h is step size

- First-order method
- Local Error (error per step) is proportional to the square of the step size
- Global error (error at a given time) is proportional to the step size
- Often serves as basis for constructing more complex methods



# Algorithm of Euler's Method



- Input: Endpoints a, b; integer N ; initial condition a.
- Output: approximation w to y at the N values of x.

- Step 1: Set h = (b − a)/N ;

  x = a; w = $y_0$
- Step 2: For i = 1, 2, . . . , N do steps 3, 4.
  - Step 3: Set w = w + h* f(x, w)  (i.e. compute wI)
  - Step 4: x = x + i*h  (i.e. compute xi)
- Step 5: STOP

# Python function for Euler's Method

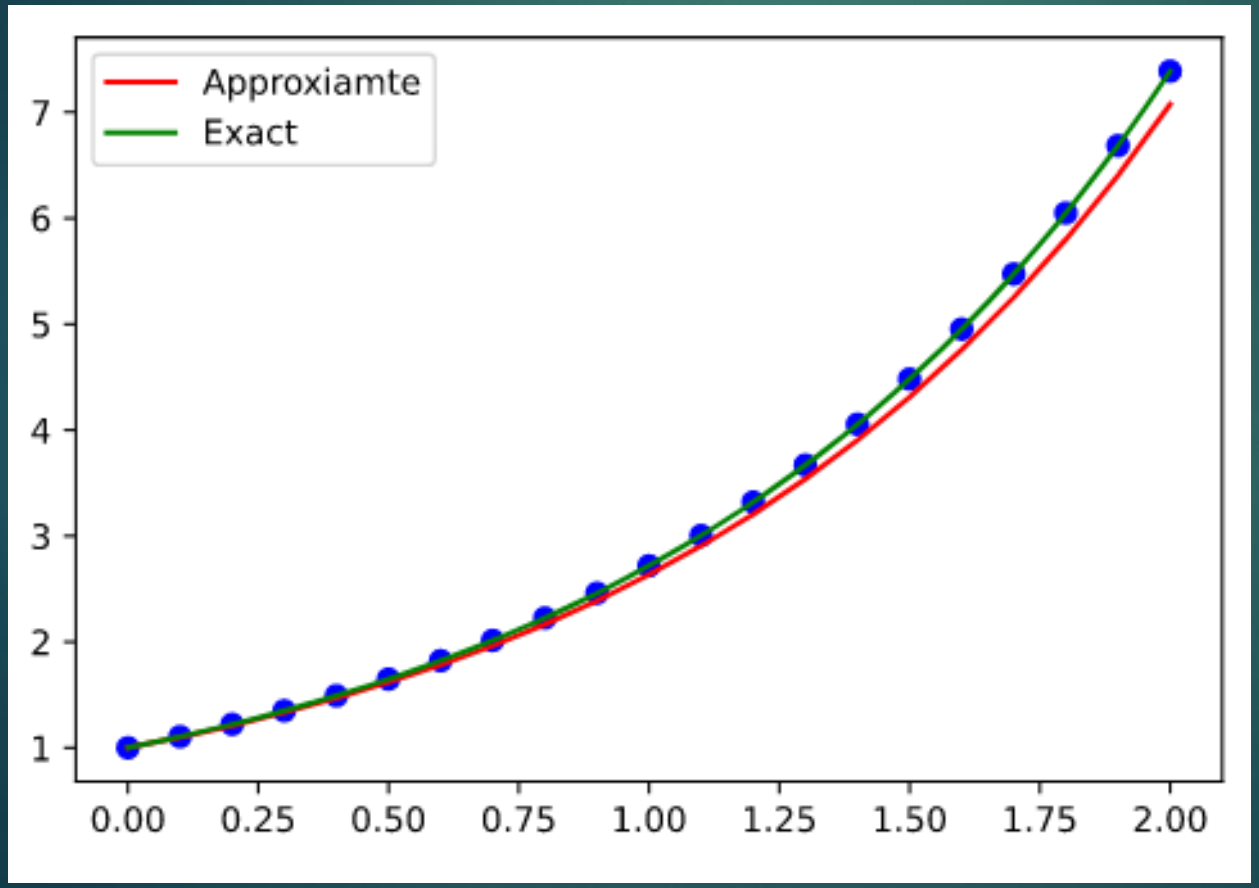


```
def euler(x0,xn,y0,n,f,g):
    lst = [[x0,y0,g(x0)]]
    h = (xn-x0)/n
    print('x0\t\ty_approx\ty_exact\n')
    for i in range(n):
        yn = y0 + h * f(x0, y0)

        print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
        y0 = yn
        x0 = x0+h
        lst.append([x0,y0,g(x0)])
    print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
    return lst
```

$$\frac{dy}{dx} = y \qquad 0 \le x \le 2 \qquad \& \qquad y(0) = 1$$



$solution: y = e^x$ $N = 20$




| x | y | yt |
|---|---|---|
| 0 | 1 | 1 |
| 0.1 | 1.1 | 1.105171 |
| 0.2 | 1.210517 | 1.221403 |
| 0.3 | 1.332657 | 1.349859 |
| 0.4 | 1.467643 | 1.491825 |
| 0.5 | 1.616826 | 1.648721 |
| 0.6 | 1.781698 | 1.822119 |
| 0.7 | 1.96391 | 2.013753 |
| 0.8 | 2.165285 | 2.225541 |
| 0.9 | 2.387839 | 2.459603 |
| 1 | 2.633799 | 2.718282 |
| 1.1 | 2.905628 | 3.004166 |
| 1.2 | 3.206044 | 3.320117 |
| 1.3 | 3.538056 | 3.669297 |
| 1.4 | 3.904986 | 4.0552 |
| 1.5 | 4.310506 | 4.481689 |
| 1.6 | 4.758674 | 4.953032 |
| 1.7 | 5.253978 | 5.473947 |
| 1.8 | 5.801372 | 6.049647 |
| 1.9 | 6.406337 | 6.685894 |
| 2 | 7.074927 | 7.389056 |

# Runge-Kutta Methods



- Runge–Kutta methods aim to achieve higher accuracy by sacrificing the efficiency of Euler's method through re-evaluating f (·, ·) at points intermediate between (x n , y(x n )) and (x n+1 , y(x n+1 )).
- The general R-stage Runge–Kutta family is defined by

$$y_{n+1} = y_n + h * \emptyset(x_n, y_n; h)$$

*Where,*

$$\emptyset(x, y; h) = \sum_{r=1}^{R} c_r k_r$$

$$k_1 = f(x, y)$$

$$k_r = f(x + h * a_r , y + h * \sum_{s=1}^{r-1} c_{rs} k_s) , \qquad r = 2, \dots, R$$

$$a_r = \sum_{s=1}^{r-1} c_{rs} k_s$$

# Runge-Kutta Methods



- For different value of R, there are different methods with corresponding number of stages
- We will be discussing only following here
  1. One Stage Runge-Kutta method
  2. Two Stage Runge-Kutta Method
  3. Four Stage Runge-kutta method

# One Stage Runge-Kutta Method



- This is Simply Euler's explicit Method.

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y) \ , a \leq x \ \leq b \quad \& \quad y_0 = f(x_0)$$

Iterative Formula

$$y_{n+1} = y_n + hf(x_n, y_n)$$

where, h is step size

# Two Stage Runge-Kutta Methods



- $y_{n+1} = y_n + h * (c_1\ k_1\ , c_2 k_2)$

Where,

- $k_1 = f(x_n\ , y_n)$
- $k_2 = f(x_n + a_2\ * h\ , y_n + b_{21}\ * h * k_1\ )$

Where,

$$c_1 + c_2 = 1$$

$$a_2 c_2 = b_{12} c_2 = \frac{1}{2}$$

For $b_{12} = a_2$, the method is second-order accurate. (with $a_2$ as free parameter).

# Examples of Second order Runge-Kutta Method



- *The Modified Euler Method:* $(a_2 = ½)$

$$y_{n+1} = y_n + h * f\left(x_n + \frac{1}{2} * h, y_n + \frac{1}{2} * k\right)$$

- *The Improved Euler Method:* $(a_2 = 1)$

$$y_{n+1} = y_n + \frac{1}{2} * h * [f(x_n, y_n) + f(x_n + h, y_n + \mathrm{k})]$$

where,

$$\mathrm{k} = h * f(x_n, y_n)$$

# Python function for Modified Euler's Method



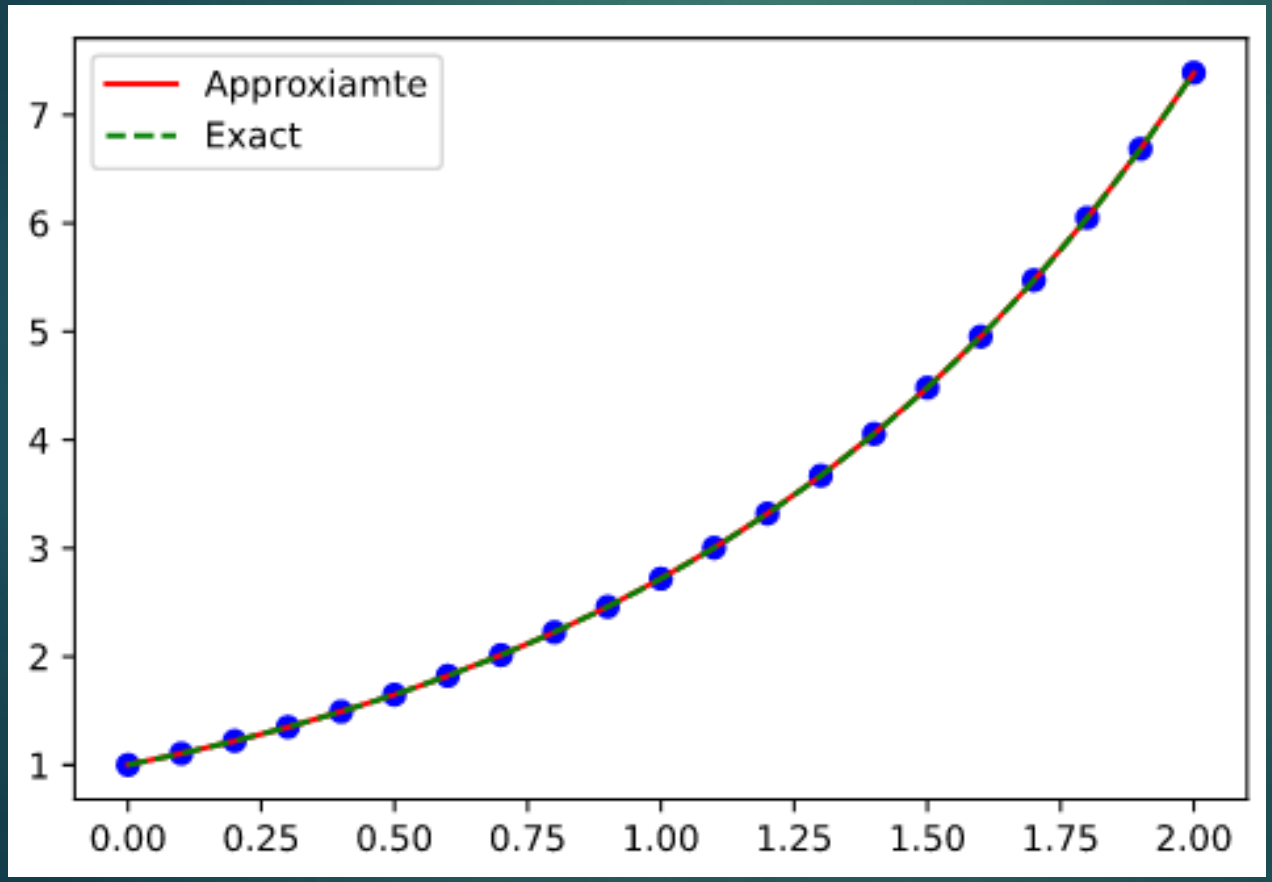
```
def eulerModified(x0,xn,y0,n,f,g):
    lst = [[x0,y0,g(x0)]]
    h = (xn-x0)/n
    print('x0\t\ty_approx\ty_exact\n')
    for i in range(n):
        k = h*f(x0, y0)
        yn = y0 + h * f(x0 +h/2, y0+k/2)

        print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
        y0 = yn
        x0 = x0+h
        lst.append([x0,y0,g(x0)])
    print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
    return lst
```

# Modified Euler Method



$$\frac{dy}{dx} = y \qquad 0 \le x \le 2 \qquad \& \qquad y(0) = 1$$

$$solution: y = e^x \qquad N = 20$$




| x | y | yt |
|---|---|---|
| 0 | 1 | 1 |
| 0.1 | 1.105127 | 1.105171 |
| 0.2 | 1.221311 | 1.221403 |
| 0.3 | 1.349713 | 1.349859 |
| 0.4 | 1.49162 | 1.491825 |
| 0.5 | 1.648451 | 1.648721 |
| 0.6 | 1.821776 | 1.822119 |
| 0.7 | 2.01333 | 2.013753 |
| 0.8 | 2.22503 | 2.225541 |
| 0.9 | 2.458995 | 2.459603 |
| 1 | 2.717566 | 2.718282 |
| 1.1 | 3.003331 | 3.004166 |
| 1.2 | 3.31915 | 3.320117 |
| 1.3 | 3.668185 | 3.669297 |
| 1.4 | 4.053927 | 4.0552 |
| 1.5 | 4.480239 | 4.481689 |
| 1.6 | 4.951386 | 4.953032 |
| 1.7 | 5.472084 | 5.473947 |
| 1.8 | 6.047544 | 6.049647 |
| 1.9 | 6.683526 | 6.685894 |
| 2 | 7.386395 | 7.389056 |

# Python function for Improved Euler's Method



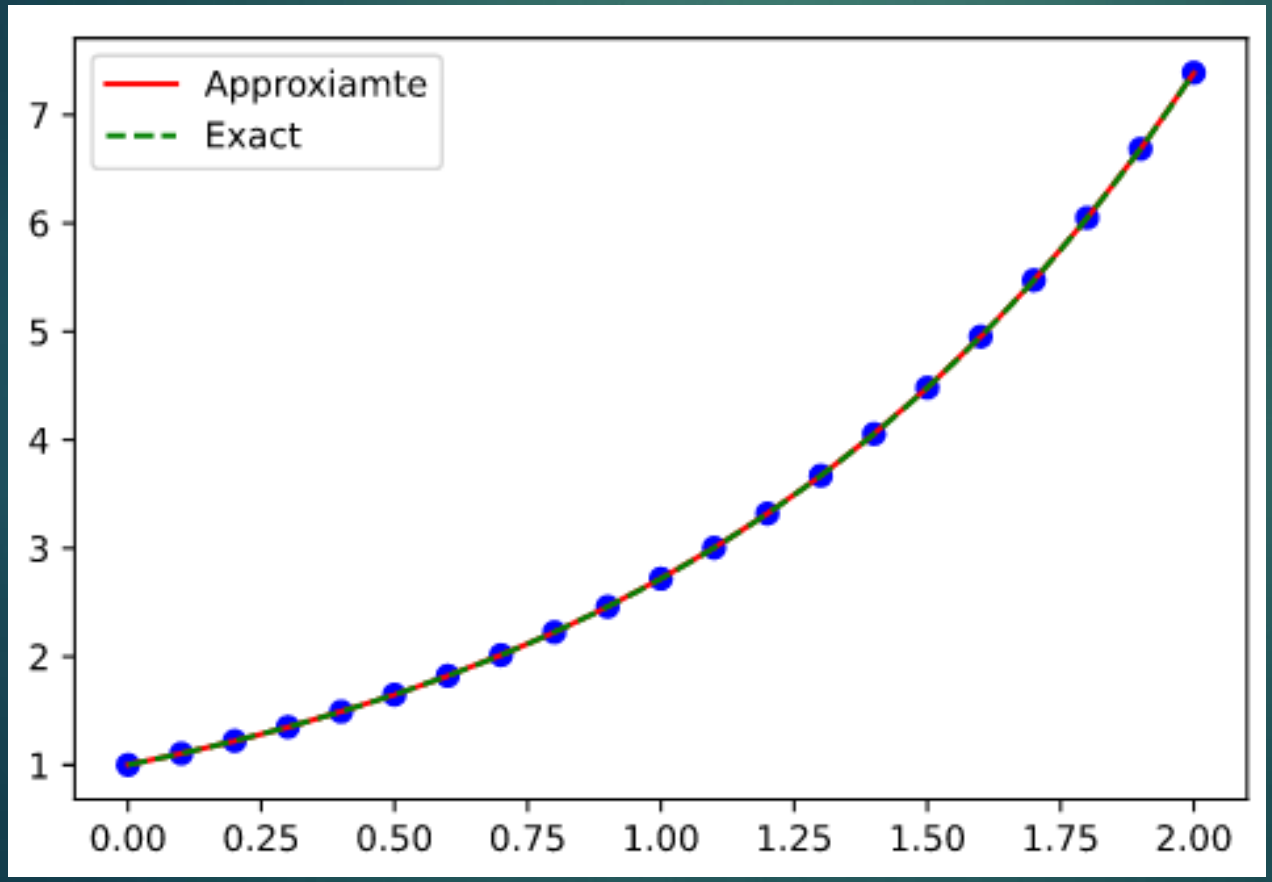
```
def eulerImproved(x0,xn,y0,n,f,g):
    lst = [[x0,y0,g(x0)]]
    h = (xn-x0)/n
    print('x0\t\ty_approx\ty_exact\n')
    for i in range(n):
        k = h*f(x0, y0)
        yn = y0 + (h/2) *( f(x0, y0) + f(x0 + h, y0 + k) )

        print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
        y0 = yn
        x0 = x0+h
        lst.append([x0,y0,g(x0)])
    print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
    return lst
```

# Improved Euler Method



$$\frac{dy}{dx} = y \qquad 0 \le x \le 2 \qquad \& \qquad y(0) = 1$$

$$solution: y = e^x \qquad N = 20$$




| x | y | yt |
|---|---|---|
| 0 | 1 | 1 |
| 0.1 | 1.105259 | 1.105171 |
| 0.2 | 1.221587 | 1.221403 |
| 0.3 | 1.35015 | 1.349859 |
| 0.4 | 1.492234 | 1.491825 |
| 0.5 | 1.649262 | 1.648721 |
| 0.6 | 1.822804 | 1.822119 |
| 0.7 | 2.014597 | 2.013753 |
| 0.8 | 2.226562 | 2.225541 |
| 0.9 | 2.460819 | 2.459603 |
| 1 | 2.719713 | 2.718282 |
| 1.1 | 3.005836 | 3.004166 |
| 1.2 | 3.32205 | 3.320117 |
| 1.3 | 3.671521 | 3.669297 |
| 1.4 | 4.057746 | 4.0552 |
| 1.5 | 4.48459 | 4.481689 |
| 1.6 | 4.956326 | 4.953032 |
| 1.7 | 5.477675 | 5.473947 |
| 1.8 | 6.053855 | 6.049647 |
| 1.9 | 6.690632 | 6.685894 |
| 2 | 7.394379 | 7.389056 |

# Four Stage Runge-Kutta Method



- $y_{n+1} = y_n + \frac{h}{6} * (k_1 + 2 * k_2 + 2 * k_3 + k_4)$

  where,

  $$k_1 = f(x_n, y_n)$$

  $$k_2 = f\left(x_n + \frac{h}{2},\ y_n + \frac{1}{2} * h * k_1\right)$$

  $$k_3 = f\left(x_n + \frac{h}{2},\ y_n + \frac{1}{2} * h * k_2\right)$$

  $$k_4 = f(x_n + h,\ y_n + h * k_3)$$

# Python function for Fourth-Order Runge Kutta Method



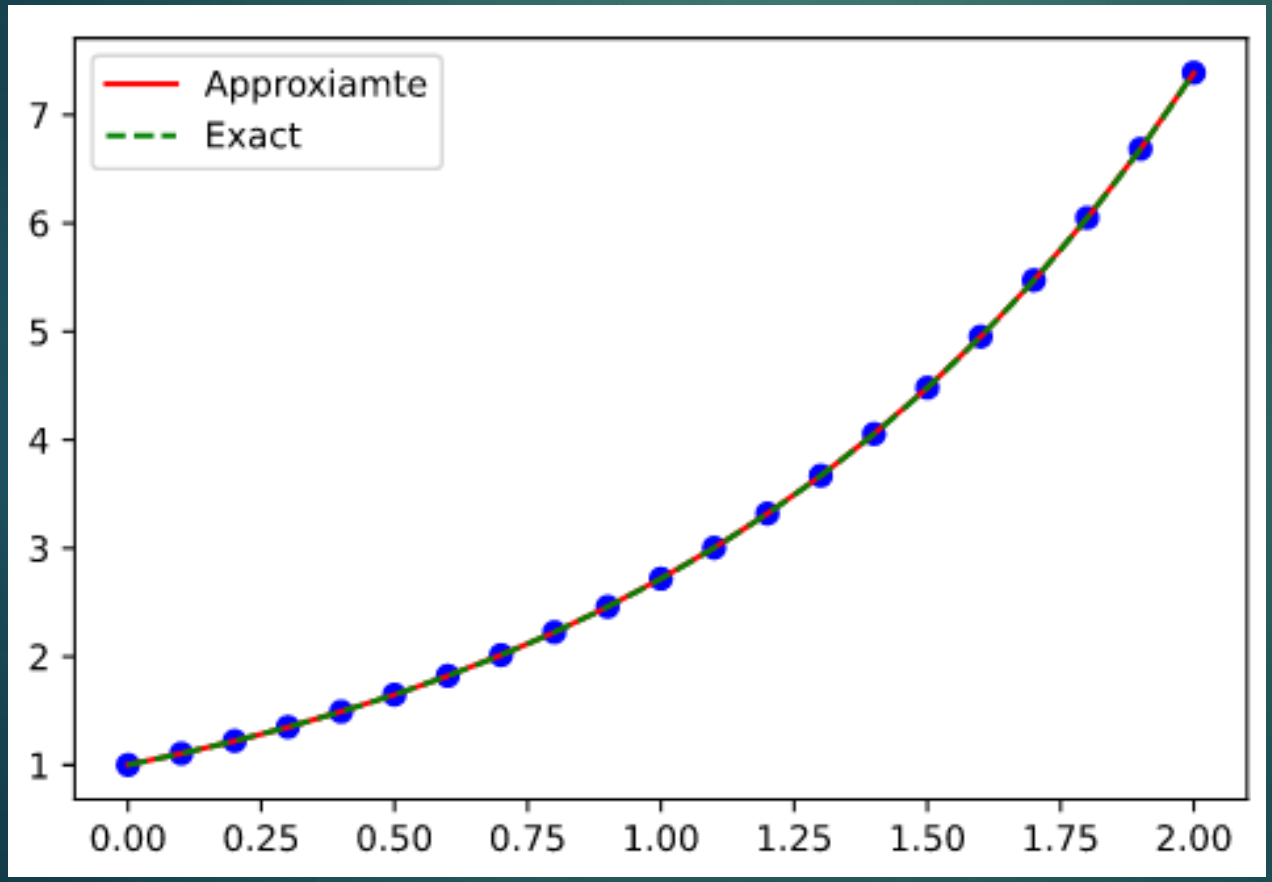
```
def rk4(x0,xn,y0,n,f,g):
    lst = [[x0,y0,g(x0)]]
    h = (xn-x0)/n
    print('x0\t\ty_approx\ty_exact\n')
    for i in range(n):
        k1 = h*f(x0, y0)
        k2 = h*f(x0+h/2,y0+k1/2)
        k3 = h*f(x0+h/2,y0+k2/2)
        k4 = h*f(x0+h,y0+k3)
        yn = y0 + (1/6) *(k1 + 2*k2 + 2*k3 + k4)

        print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
        y0 = yn
        x0 = x0+h
        lst.append([x0,y0,g(x0)])
    print(f'{x0:f}\t{y0:f}\t{g(x0):f}')
    return lst
```

# Runge-Kutta Fourth Order



$$\frac{dy}{dx} = y \qquad 0 \le x \le 2 \qquad \& \qquad y(0) = 1$$

$$solution: y = e^x \qquad N = 20$$




| x | y | yt |
|---|---|---|
| 0 | 1 | 1 |
| 0.1 | 1.105171 | 1.105171 |
| 0.2 | 1.221403 | 1.221403 |
| 0.3 | 1.349859 | 1.349859 |
| 0.4 | 1.491825 | 1.491825 |
| 0.5 | 1.648721 | 1.648721 |
| 0.6 | 1.822119 | 1.822119 |
| 0.7 | 2.013753 | 2.013753 |
| 0.8 | 2.225541 | 2.225541 |
| 0.9 | 2.459603 | 2.459603 |
| 1 | 2.718282 | 2.718282 |
| 1.1 | 3.004166 | 3.004166 |
| 1.2 | 3.320117 | 3.320117 |
| 1.3 | 3.669297 | 3.669297 |
| 1.4 | 4.0552 | 4.0552 |
| 1.5 | 4.481689 | 4.481689 |
| 1.6 | 4.953033 | 4.953032 |
| 1.7 | 5.473948 | 5.473947 |
| 1.8 | 6.049648 | 6.049647 |
| 1.9 | 6.685895 | 6.685894 |
| 2 | 7.389056 | 7.389056 |

# Comparison of different methods at common mesh points



$y' = y - x^2 + 1 \qquad 0 \leq x \leq 2 \qquad \& \qquad y(0) = 0.5$

| $x_i$ | Exact | Euler h = 0.025 | Modified Euler h = 0.05 | RK order four h = 0.1 |
|---|---|---|---|---|
| 0.0 | 0.5000000 | 0.5000000 | 0.5000000 | 0.5000000 |
| 0.1 | 0.6574145 | 0.6554982 | 0.6573085 | 0.6574144 |
| 0.2 | 0.8292986 | 0.8253385 | 0.8290778 | 0.8292983 |
| 0.3 | 1.0150706 | 1.0089334 | 1.0147254 | 1.0150701 |
| 0.4 | 1.2140877 | 1.2056345 | 1.2136079 | 1.2140869 |
| 0.5 | 1.4256394 | 1.4147264 | 1.4250141 | 1.4256384 |

# References




- Numerical Analysis, by Richard L. Burden & J. Douglas Faires
- Numerical Solution of Ordinary Differential Equations, by E. Süli
- Wikipedia

# Thank You