IVP Module

SGTB Khalsa College, University of Delhi

Preetpal Singh(2020PHY1140)(20068567043)

Unique Paper Code: 32221401

Paper Title: Mathematical Physics III

Submitted on: April 11, 2022 2022

B.Sc(H) Physics Sem IV

Submitted to: Dr. Mamta

Given Equations are:

$$y_1' = y_2 - y_3 + x$$
$$y_2' = 3x^2$$
$$y_3' = y_2 + \mathrm{e}^{-x}$$

Initial Conditions : $y_1(0) = 1, y_2(0) = 1, y_3(0) = -1$ and compute y_i for $0 \leq x \leq x_f$ with $x_f = 1$.
Analytical Solution:-

$$y_1(x) = -0.05x^5 + 0.25x^4 + x + 2 - \mathrm{e}^{-x}$$
$$y_2(x) = x^3 + 1$$
$$y_3(x) = 0.25x^4 + x - \mathrm{e}^{-x}$$

Variation with step size $h = (x_f - x_0)/N$ by taking $N = 10^k$ with $k = 1, 2, \ldots, 6$ and Error Plots

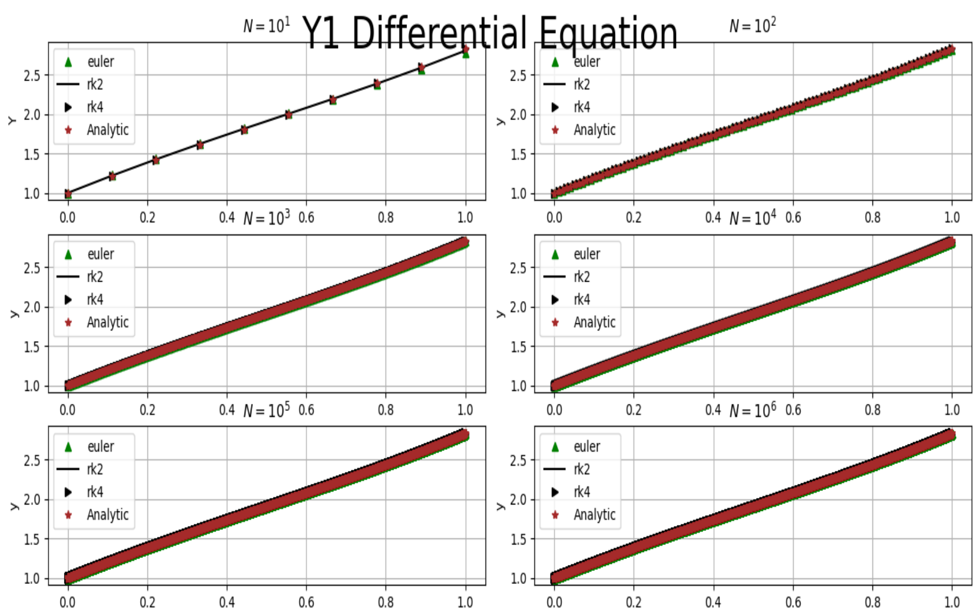

Figure 1: Y v/s x

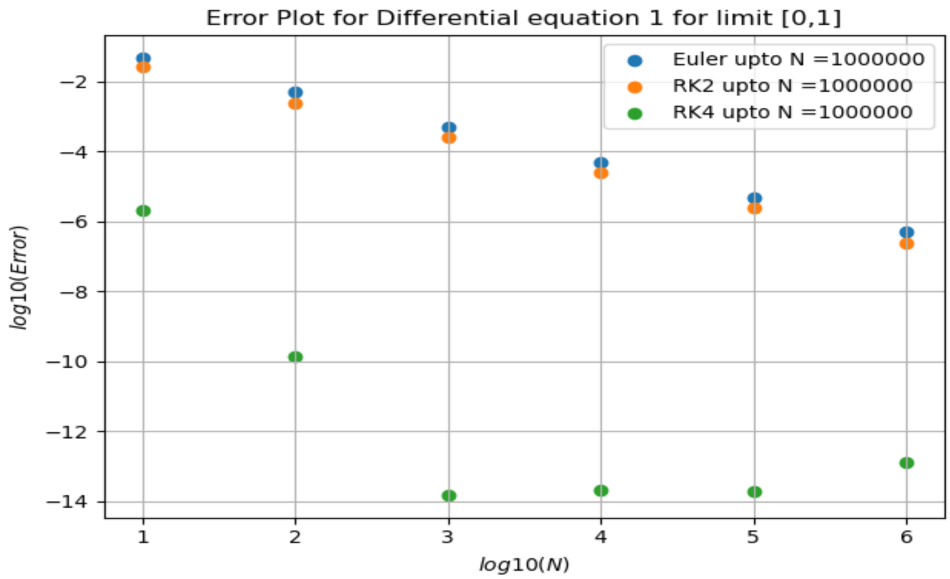

Figure 2: Error Plot

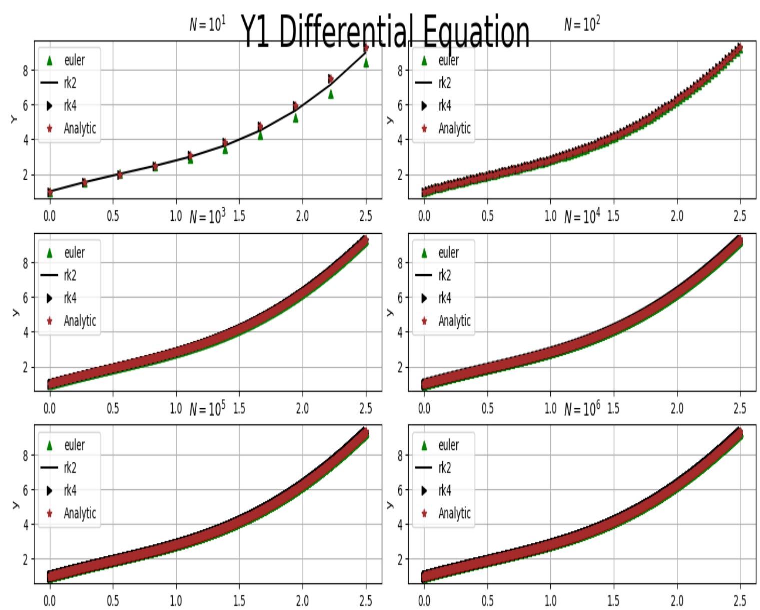

Figure 3: Y v/s x

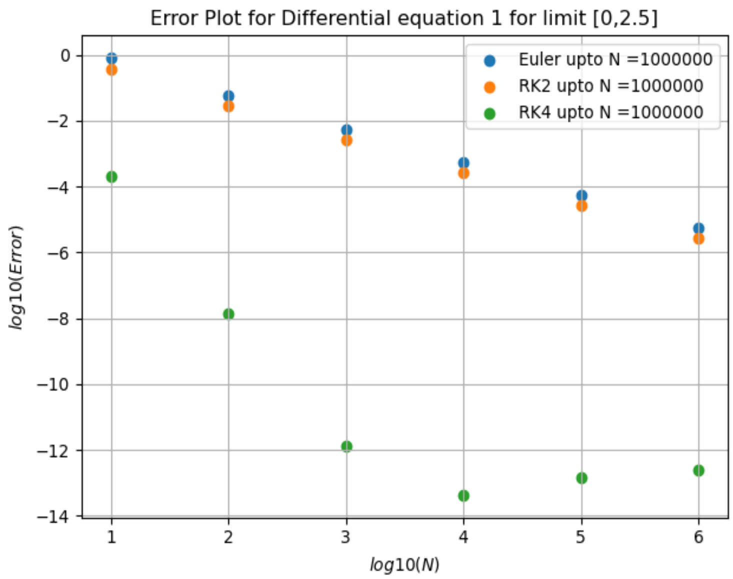

Figure 4: Error Plot

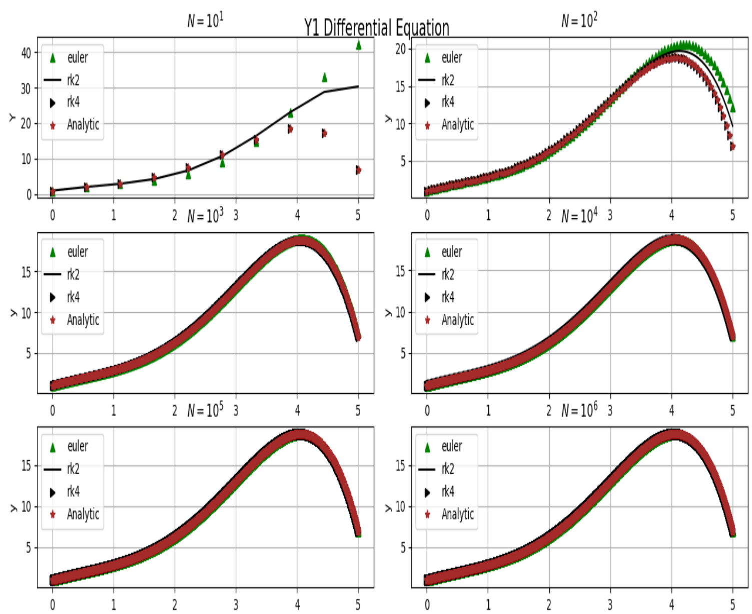

Figure 5: Y v/s x

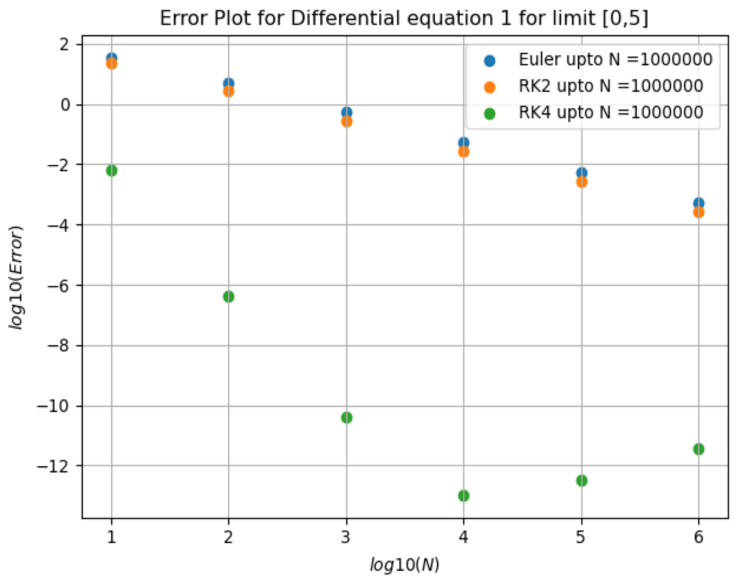

Figure 6: Error Plot

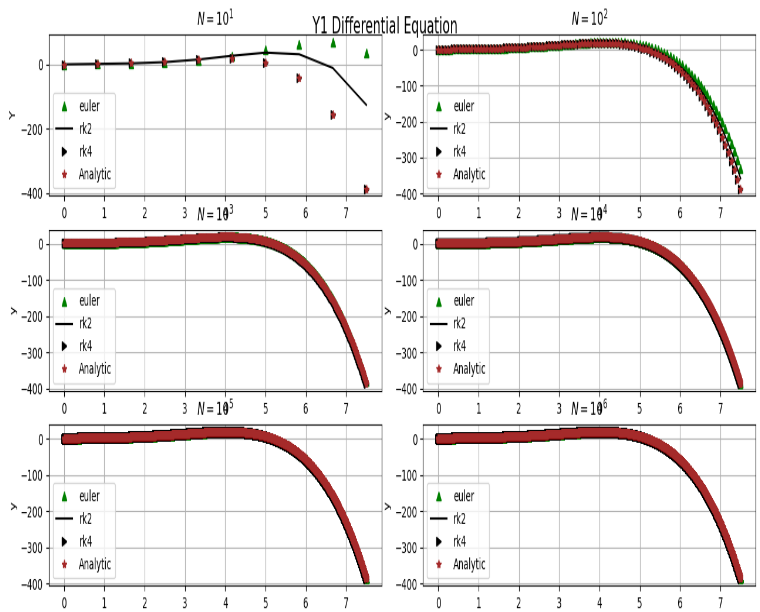

Figure 7: Y v/s x

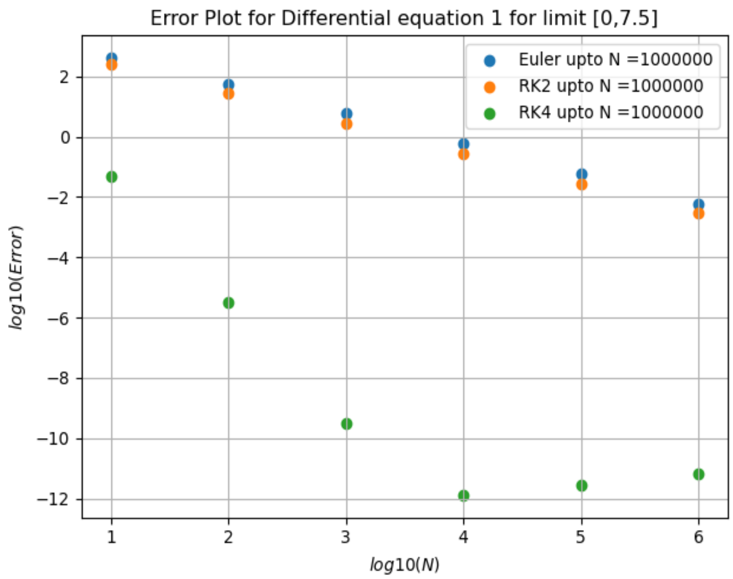

Figure 8: Error Plot

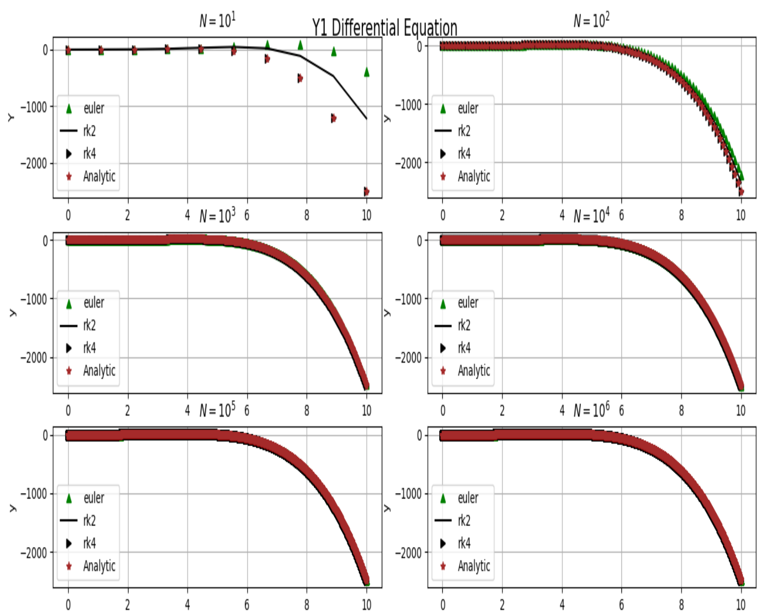

Figure 9: Y v/s x

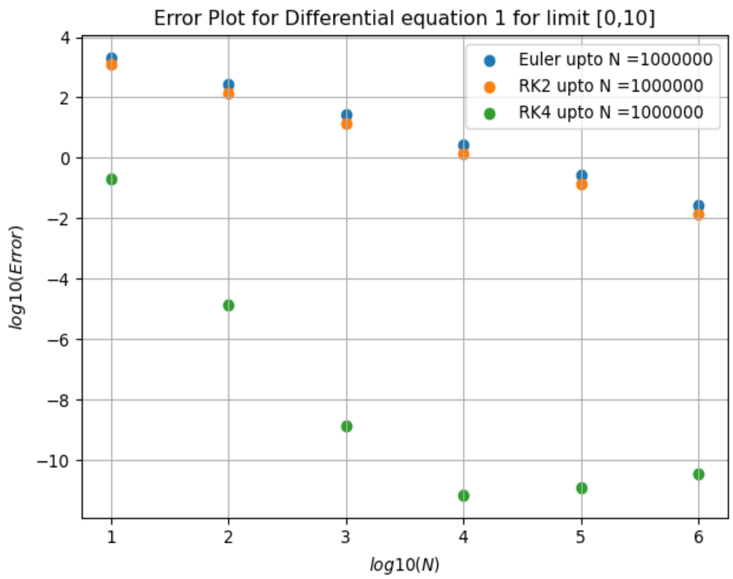

Figure 10: Error Plot

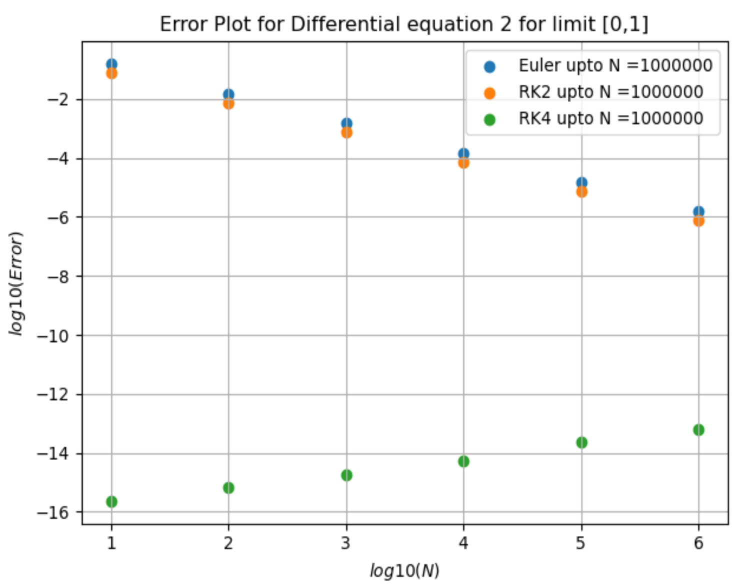

Figure 11: Y v/s x

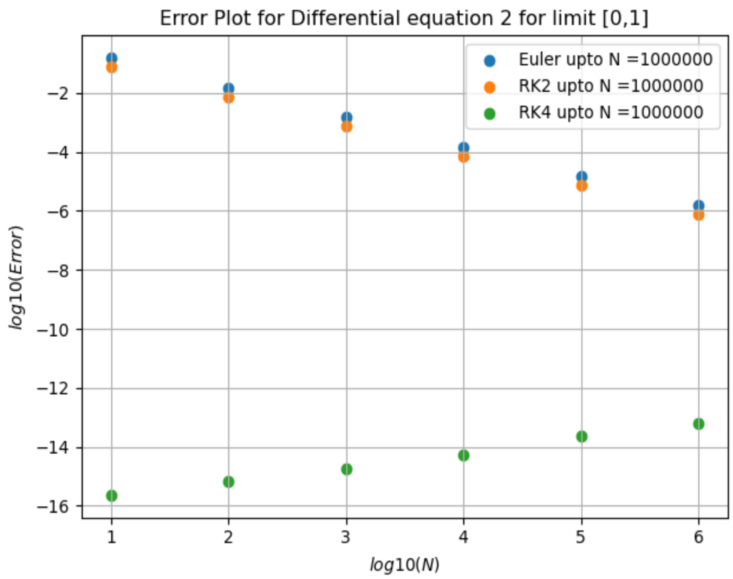

Figure 12: Error Plot

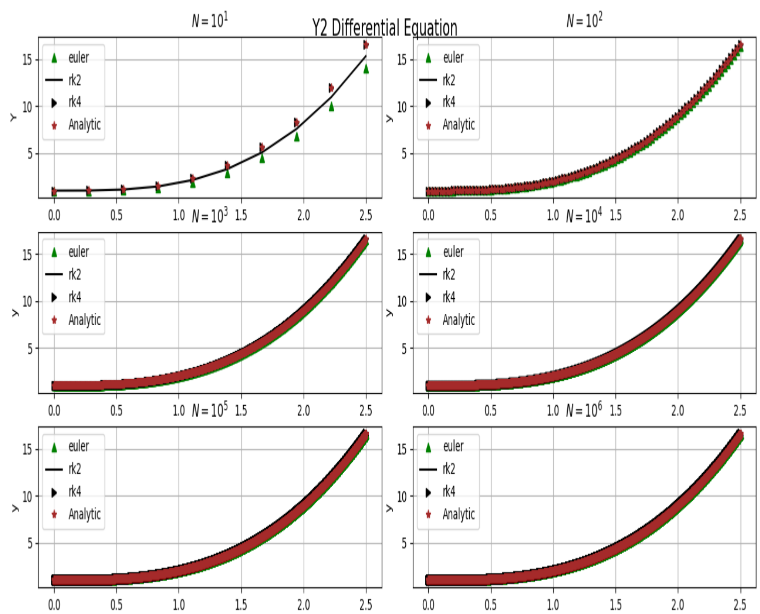

Figure 13: Y v/s x

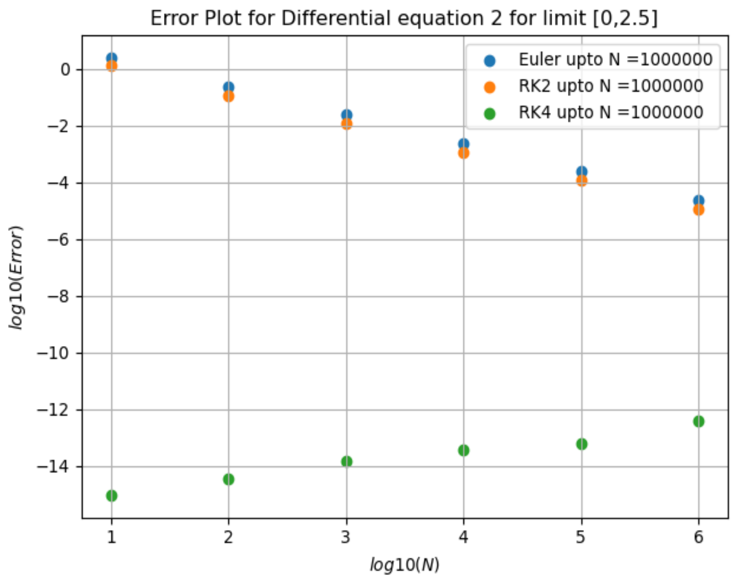

Figure 14: Error Plot

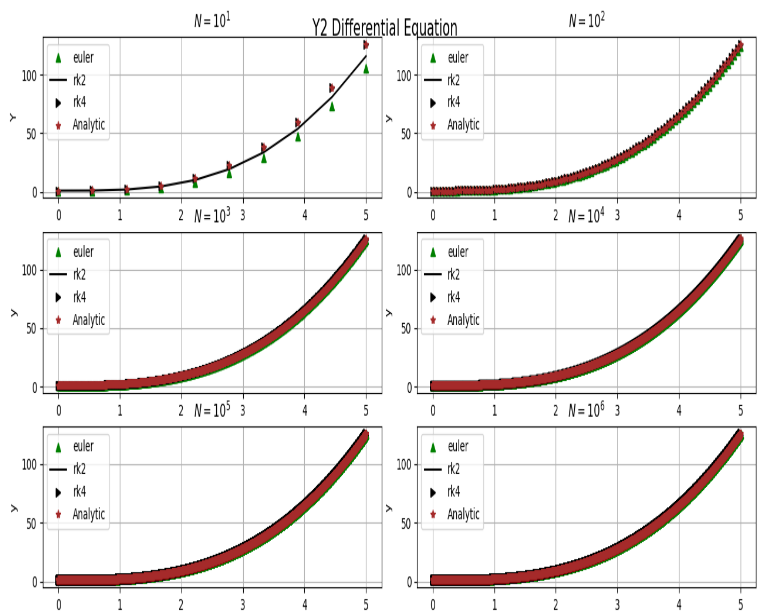

Figure 15: Y v/s x

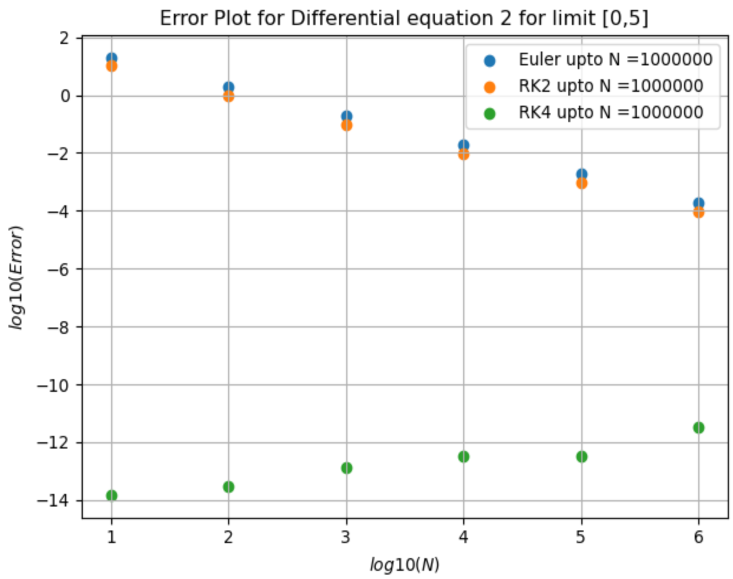

Figure 16: Error Plot

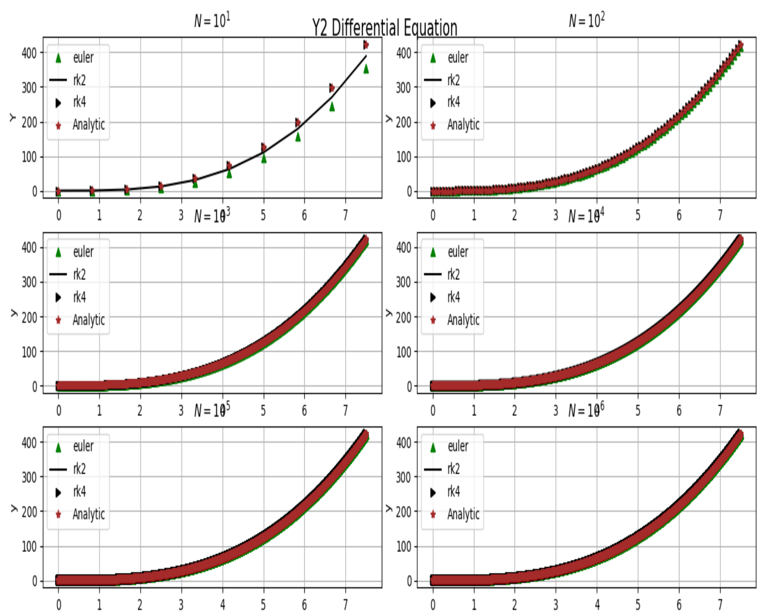

Figure 17: Y v/s x

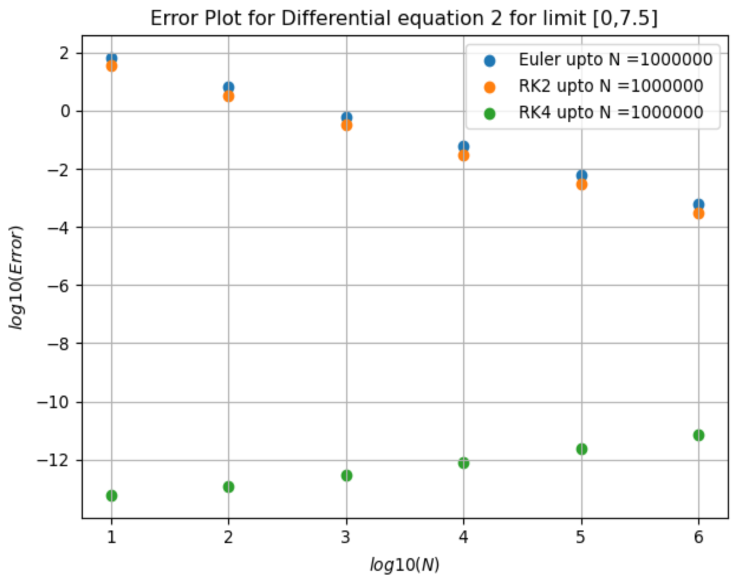

Figure 18: Error Plot

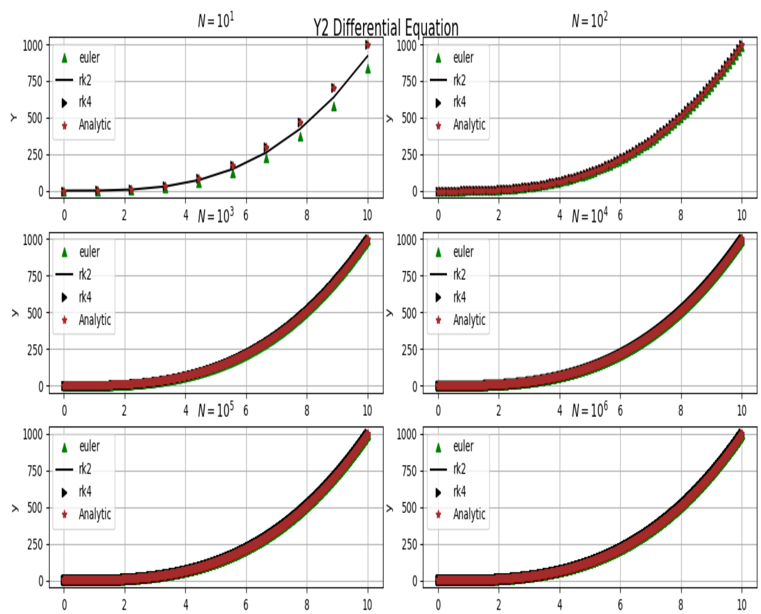

Figure 19: Y v/s x

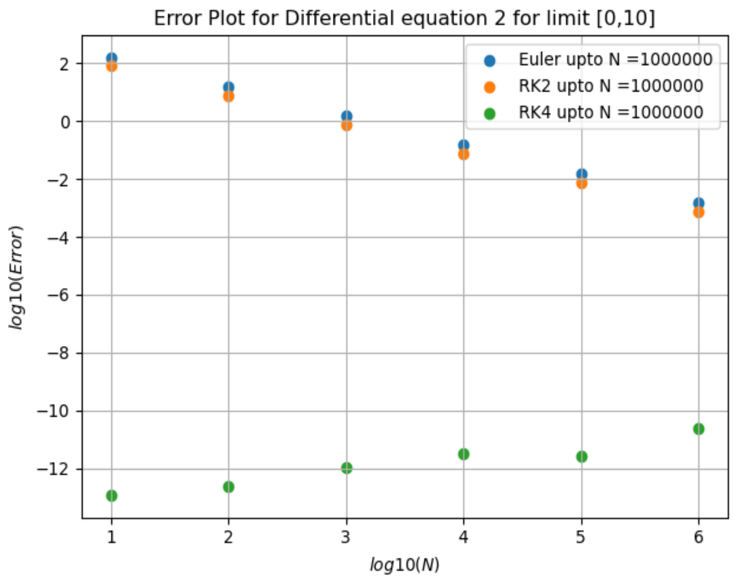

Figure 20: Error Plot

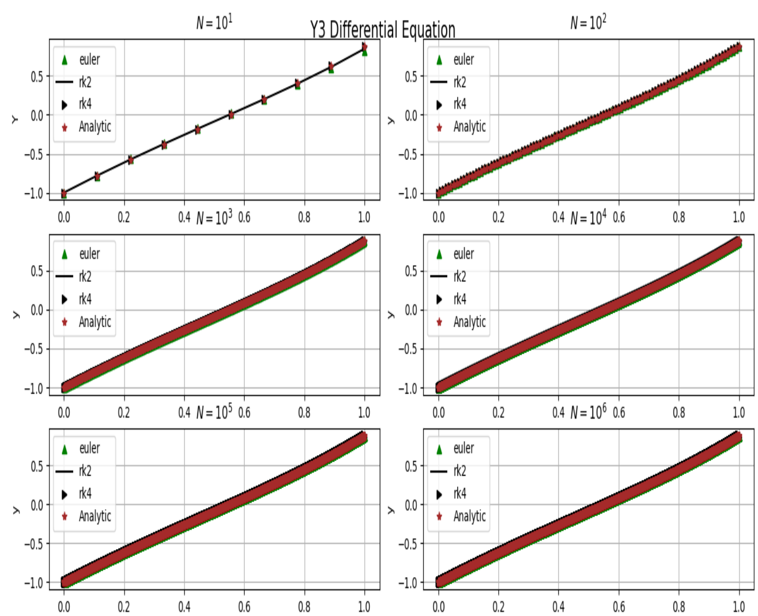

Figure 21: Y v/s x

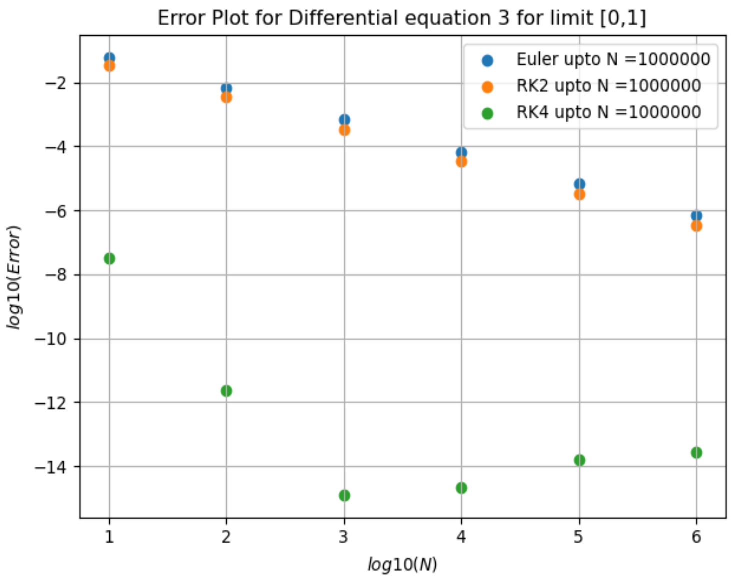

Figure 22: Error Plot

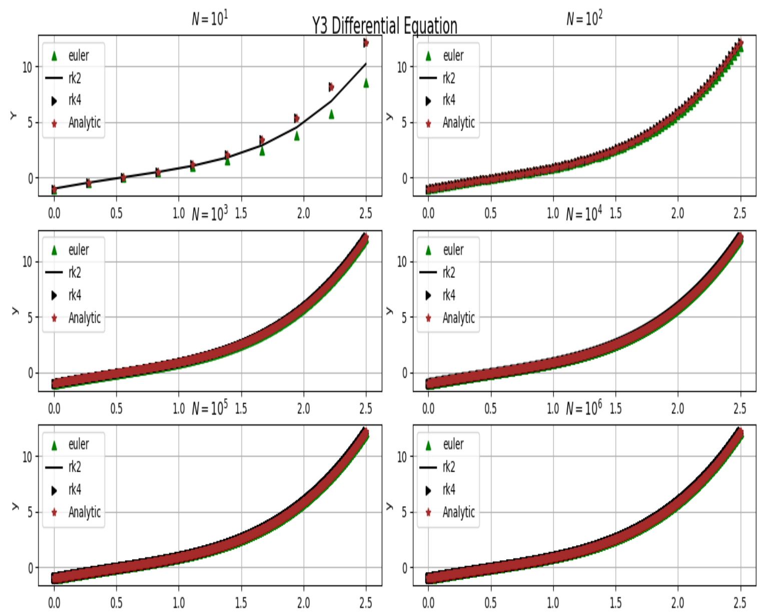

Figure 23: Y v/s x

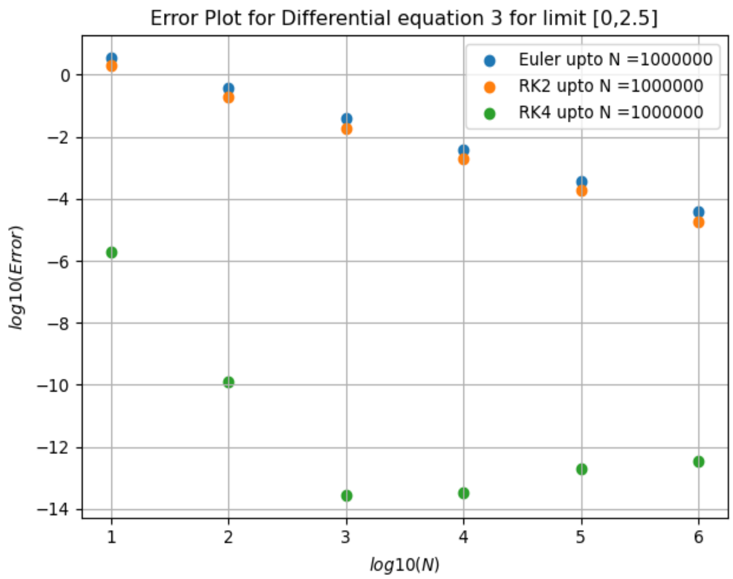

Figure 24: Error Plot

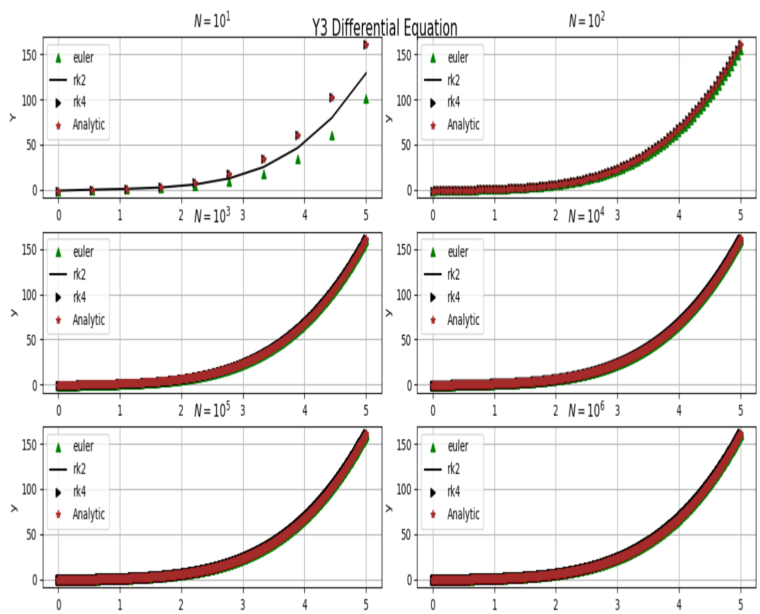

Figure 25: Y v/s x

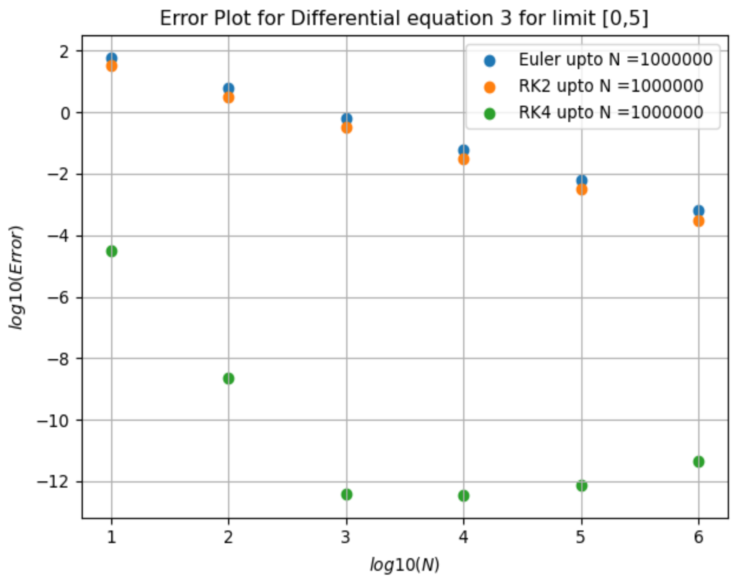

Figure 26: Error Plot

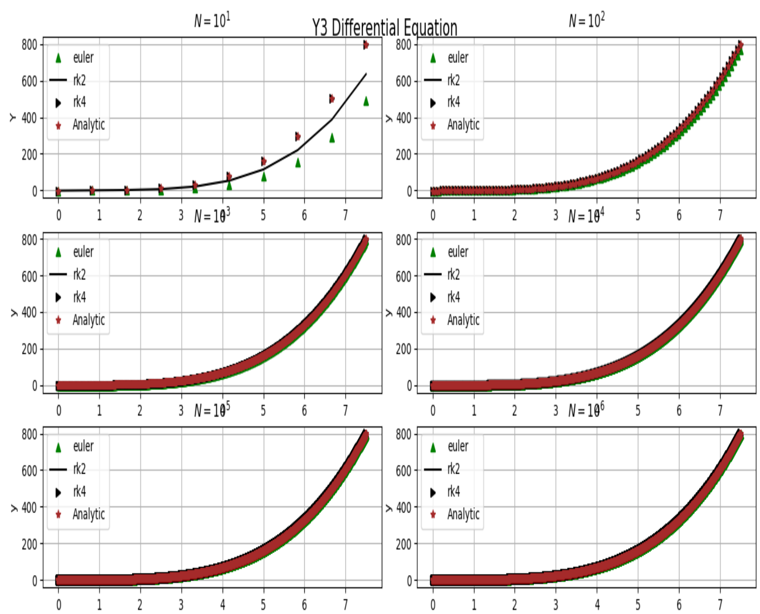

Figure 27: Y v/s x

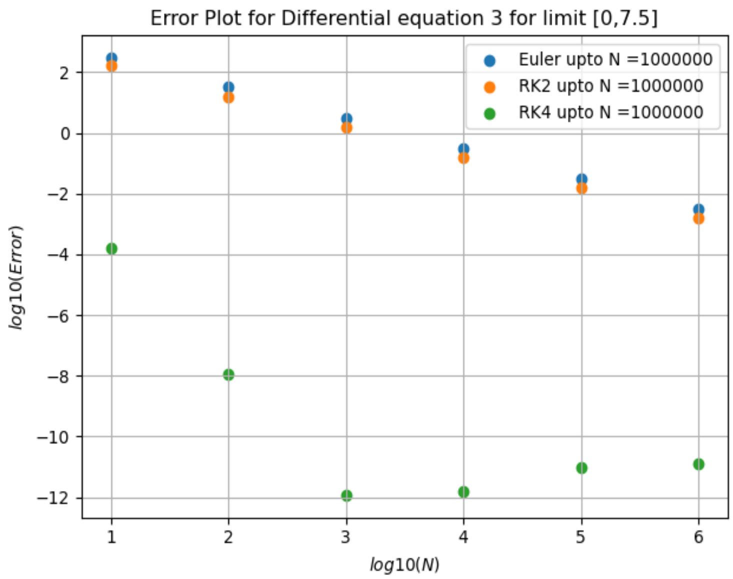

Figure 28: Error Plot

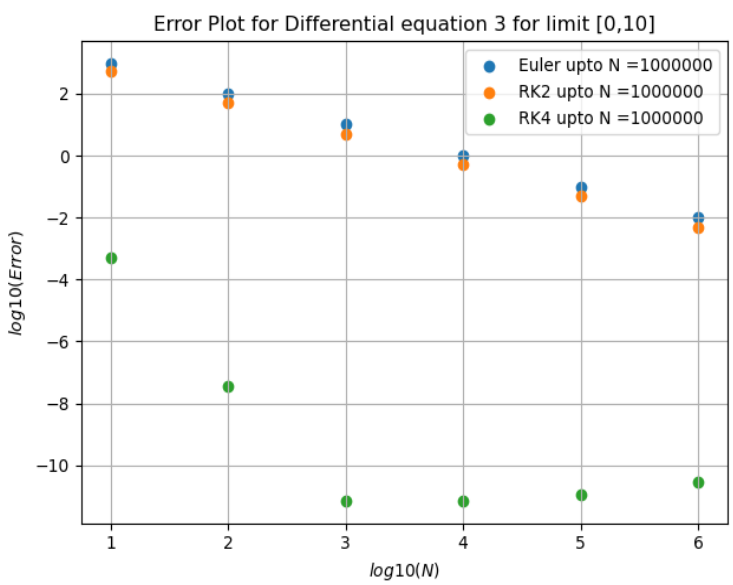

Figure 29: Y v/s x

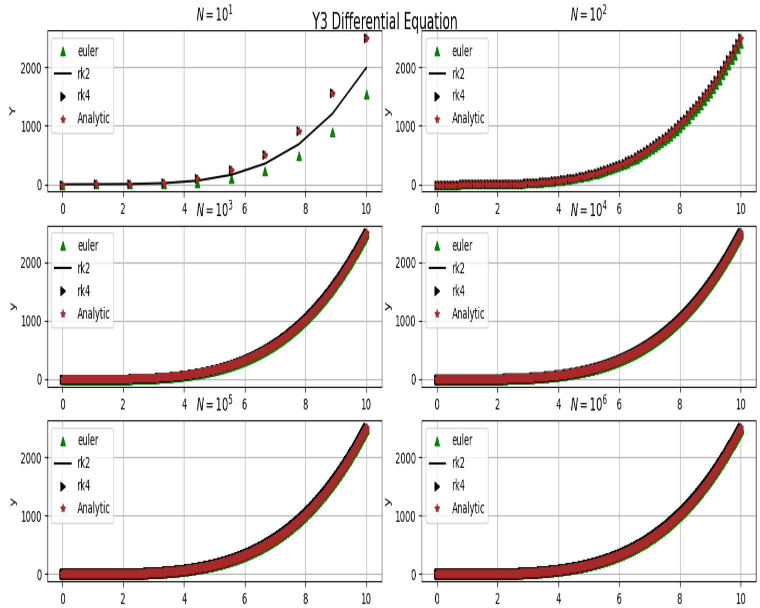

Figure 30: Error Plot

Analysis

The given figures shows $log_{10}Error$ reduces linearly w.r.t $log_{10}N$ for Euler and RK2 Method as we increase the Value of N from 10^1 to 10^6 Whereas the Error for RK4 is minimum(*i.e.* of order 10^{-12} to 10^{-16}) for most of the cases. But in cases when error is less than of the order 10^{-16}(which is epsilon of computer) it start giving some junk values so the graph shows increase in error(which is unexpected because usually Error decreases with increase in number of intervals(N)) when we take bigger values of N.When we increase xf from 1-10 the numerical methods requires more intervals to better approximate it.

Algorithm

Algorithm 1 Euler Method

function INPUT(f, *initial conditions*, t)
Calculate dt ← step size
▷ *Difference of two consecutive elements of time array* ◁
Define X ← Empty array
▷ *Empty array to store output* ◁
$X \leftarrow X_0$
▷ *Assigning Initial conditions to output array* ◁
for all $i \in \{1, \ldots, N\}$ **do**
$X_{i+1} = X_i + f\,(t_i, x_i, paramteres)\,dt$
return X, t

Algorithm 2 Rk2 Method

function INPUT(f, *initial conditions*, t)
Calculate dt ← step size
▷ *Difference of two consecutive elements of time array* ◁
Define X ← Empty array
▷ *Empty array to store output* ◁
$X \leftarrow X_0$
▷ *Assigning Initial conditions to output array* ◁
for all $i \in \{1, \ldots, N\}$ **do**

$k_1 = dt(f\,(t_i, X_i, paramteres))$
$k_2 = dt(f\,(t_i + dt, x_i + k_1, paramteres))$
$X_{i+1} = X_i + \frac{(k_1+k_2)}{2}$
return X, t

Algorithm 3 Rk4 Method

function INPUT(f, *initial conditions*, t)
Calculate dt ← step size
▷ *Difference of two consecutive elements of time array* ◁
Define X ← Empty array
▷ *Empty array to store output* ◁
$X \leftarrow X_0$
▷ *Assigning Initial conditions to output array* ◁
for all $i \in \{1, \ldots, N\}$ **do**

$k_1 = f\left(t_i, X_i, paramteres\right)$
$k_2 = dt(f\left(t_i + dt/2, x_i + dt/2 \times k_1, paramteres\right))$
$k_3 = hf\left(t_i + \frac{dt}{2}, X_i + dt \times k_3, parameters\right)$
$k_4 = f\left(t_i + dt, X_i + dt \times k_3, parameters\right)$
$X_{i+1} = X_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$

return X, t

Programmes

```python
import numpy as np
def euler(f, initial_cond , t):
    """
    Finds the solution of a Differential Equation using Euler Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth order
    Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond

    ele = np.array([])

    for i in range(0 , len(t)-1):
        ele = mat[i,:] + np.multiply(h, f(t[i], mat[i,:]))
        mat[i+1,:] = ele

    return mat


def RK_2(f, initial_cond ,t):
```

```
    """
    Finds the solution of a Differential Equation using RK-2 Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth order Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond

    k1 = np.array([])
    k2 = np.array([])

    for i in range(0 , len(t)-1):
        k1 = np.multiply(h, f(t[i], mat[i,:]))
        k2 = np.multiply(h, f(t[i]+h/2, mat[i,:]+ k1/2))
        sum = np.multiply((k1+k2),1/2)

        ele = mat[i,:] + sum
        mat[i+1,:] = ele

    return mat



def RK_4(f, initial_cond, t):
    """
    Finds the solution of a Differential Equation using RK-4 Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth order Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond
```

```python

    k1 = np.array([])
    k2 = np.array([])
    k3 = np.array([])
    k4 = np.array([])
    ele = np.array([])

    for i in range(0 , len(t)-1):

        k1 = f(t[i], mat[i,:])
        k2 = f(t[i]+(h/2),(mat[i,:]+np.multiply(k1, (h/2))))
        k3 = f(t[i]+(h/2),(mat[i,:]+np.multiply(k2, (h/2))))
        k4 = f(t[i]+(h/1),(mat[i,:]+np.multiply(k3, (h/1))))
        sum = np.multiply((k1+np.multiply(k2,2)+np.multiply(k3,2)+k4), (1/6))

        ele = mat[i,:] + np.multiply((sum), h)
        mat[i+1,:] = ele

    return mat
```

```python
from IVP import euler, RK_2, RK_4
import numpy as np
import matplotlib.pyplot as plt
from  scipy.integrate import RK45
from prettytable import PrettyTable
#Funcion To Be Defined(Not to be included in Module)
def func1(x,x_vec):
    ans_vec = np.zeros((3))
    ans_vec[0] = x_vec[1] - x_vec[2] + x
    ans_vec[1] = 3*x**2
    ans_vec[2] = x_vec[1] + np.exp(-x)
    return ans_vec
def graph(x,analytic,euler_final,rk2_final,rk4_final,title):
    fig,axs=plt.subplots(3,2,figsize=(15,15))
    fig.suptitle(title, fontsize=15)
    ax11,ax12,ax21,ax22,ax31,ax32=axs[0][0],axs[0][1],axs[1][0],axs[1][1],axs[2][0],axs[2][1]
    ax11.plot(x[0],euler_final[0],'^', color='green',label="euler"),ax11.plot(x[0],rk2_final[0],'-', color='black',label="rk2")
    ax11.plot(x[0],rk4_final[0],'>',color='black',label="rk4"),ax11.plot(x[0],analytic[0],'*',color='brown',label="Analytic")
    ax11.set_title("$N= 10^1$"),ax11.set_ylabel("Y"),ax11.set_xlabel("x")
    ax12.plot(x[1],euler_final[1],'^', color='green',label="euler"),ax12.plot(x[1],rk2_final[1],'-', color='black',label="rk2")
    ax12.plot(x[1],rk4_final[1],'>',color='black',label="rk4"),ax12.plot(x[1],analytic[1],'*',color='brown',label="Analytic")
    ax12.set_title("$N=10^2$"),ax12.set_ylabel("y"),ax12.set_xlabel("x")
    ax21.plot(x[2],euler_final[2],'^', color='green',label="euler"),ax21.plot(x[2],rk2_final[2],'-', color='black',label="rk2")
    ax21.plot(x[2],rk4_final[2],'>',color='black',label="rk4"),ax21.plot(x[2],analytic[2],'*',color='brown',label="Analytic")
    ax21.set_title("$N=10^3$"),ax21.set_ylabel("y"),ax21.set_xlabel("x")
    ax22.plot(x[3],euler_final[3],'^', color='green',label="euler"),ax22.plot(x[3],rk2_final[3],'-', color='black',label="rk2")
    ax22.plot(x[3],rk4_final[3],'>',color='black',label="rk4"),ax22.plot(x[3],analytic[3],'*',color='brown',label="Analytic")
    ax22.set_title("$N=10^4$"),ax22.set_ylabel("y"),ax22.set_xlabel("x")
    ax31.plot(x[4],euler_final[4],'^', color='green',label="euler"),ax31.plot(x[4],rk2_final[4],'-', color='black',label="rk2")
    ax31.plot(x[4],rk4_final[4],'>',color='black',label="rk4"),ax31.plot(x[4],analytic[4],'*',color='brown',label="Analytic")
    ax31.set_title("$N=10^5$"),ax31.set_ylabel("y"),ax31.set_xlabel("x")
    ax32.plot(x[5],euler_final[5],'^', color='green',label="euler"),ax32.plot(x[5],rk2_final[5],'-', color='black',label="rk2")
    ax32.plot(x[5],rk4_final[5],'>',color='black',label="rk4"),ax32.plot(x[5],analytic[5],'*
```

```
    ',color='brown',label="Analytic")
    ax32.set_title("$N=10^6$"),ax32.set_ylabel("y"),ax32.set_xlabel("x")
    ax11.legend(),ax11.grid(True),ax12.legend(),ax12.grid(True),ax21.legend(),ax21.grid(True
    ),ax22.legend(),ax22.grid(True)
    ax31.legend(),ax31.grid(True),ax32.legend(),ax32.grid(True);plt.tight_layout()
    plt.show()
x=np.linspace(0,1,100)
initial_conds = [1,1,-1]
euler_final,rk2_final,rk4_final,analytic_final,x_final=[],[],[],[],[]
# for j in np.arange(1,11,1.5):
E1,E2,E3=[],[],[];p=[]
for i in np.arange(1,3,1):
    x = np.linspace(0,2.5,10**i)
    analytic = [-0.05*x**5+0.25*x**4+x+2-np.exp(-x),x**3+1,0.25*x**4+x-np.exp(-x)]
    euler_1 = euler(func1,initial_conds,x).T[1]
    rk2 = RK_2(func1,initial_conds,x).T[1]
    rk4= RK_4(func1,initial_conds,x).T[1]
    euler_final.append(euler_1)
    rk2_final.append(rk2)
    rk4_final.append(rk4)
    analytic_final.append(analytic[1])
    x_final.append(x)
    #print(RK_4(func1,initial_conds, x))
    euler_error = np.max(abs(analytic[1]-(euler(func1,initial_conds, x)).T[1]))
    rk2error = np.max(abs(analytic[1]-(RK_2(func1,initial_conds, x)).T[1]))
    rk4error = np.max(abs(analytic[1]-(RK_4(func1,initial_conds, x)).T[1]))
    E1.append(rk2error);E2.append(rk4error);E3.append(euler_error)

    p.append(10**i)
# plt.scatter(np.log10(p),np.log10(E3),label="Euler upto N ={}".format(10**i))
# plt.scatter(np.log10(p),np.log10(E1),label="RK2 upto N ={}".format(10**i))
# plt.scatter(np.log10(p),np.log10(E2),label="RK4 upto N ={}".format(10**i))
# plt.legend();plt.grid(True);plt.xlabel("$log10(N)$");plt.ylabel("$log10(Error)$");
# plt.title("Error Plot for Differential equation 2 for limit [0,2.5]");plt.tight_layout()
# plt.show()
# graph(x_final,analytic_final,euler_final,rk2_final,rk4_final,"Y3 Differential Equation")

def func2(x,x_vec):
    ans_vec = np.zeros((2))
    ans_vec[0] = x_vec[1]
    ans_vec[1] = 2*x_vec[1]-2*x_vec[0]+np.exp(2*x)*np.sin(x)
    return ans_vec
initial_conds = [-0.4,-0.6]
x=np.linspace(0,1,6)
print(x)
print(RK_2(func2,initial_conds,x))
```

Modify your program to compute $E = \max\left(|y_{\mathrm{anal}_\mathrm{i}} - y_{\mathrm{num\ i}}|\right)$ (where $y_i = y\left(x_i\right)$ and plot $\log_{10}(E)$ as a function of $\log_{10}(N)$ or $\log_{10}(h)$.