Initial value problem using Euler,RK2 and RK4 methods
Lab Report for Assignment No. 8

Harsh saxena
(2020PHY1162)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

April 11, 2022

Submitted to Dr. Mamta
"32221401 - MATHEMATICAL PHYSICS III"

Contents

1 Theory

We can convert a differential equation of order n into n first order differential equation -

$$\frac{d^n y}{x^n} = f(x, y, y', y", y^{(3)}, ...y^{(n-1)})$$

can be modified as follows

$$y = Y_0, y' = Y_1, y" = Y_2, y^{(n-1)} = Y_{n-1} \tag{1}$$

$$\frac{dY_0}{dx} = Y_1 \tag{2}$$

$$\frac{dY_1}{dx} = Y_2 \tag{3}$$

$$. \tag{4}$$

$$. \tag{5}$$

$$. \tag{6}$$

$$\frac{dY_{n-1}}{dx} = f(x, y, y', y", y^{(3)}, ...y^{(n-1)}) \tag{7}$$

THEORY

b) we have -

$$y'' - 2y' + 2y = e^{2x} \sin x \quad \text{—— (1)}$$

first convert this second order differential equation in two first order differential equation
let,

$$\frac{dy}{dx} = u \quad \text{—— (2)}$$

using (2), eqn becomes

$$\frac{du}{dx} - 2u + 2y = e^{2x} \sin x$$

$$\frac{du}{dx} = e^{2x} \sin x + 2u - 2y \quad \text{— (3)}$$

$$f(x, y, u) \Rightarrow \frac{dy}{dx} = u$$

$$f(x, y, u) \Rightarrow \frac{du}{dx} = e^{2x} \sin x + 2u - 2y$$

Initial Conditions

$$y(0) = -0.4 \quad , \quad y'(0) = -0.6$$

for,

$$0 \leq x \leq 1$$

Step size,

$$h = \frac{b-a}{N}$$

$$h = \frac{1-0}{5} = 0.2$$

RK 2 Method

$$y_{i+1} = y_i + \left(\frac{K_1 + K_2}{2}\right)$$

$K_1 = hf(x_i, y_i)$, $K_2 = hf(x_i + h, y_i + K_1)$

Step 1

$$K_1 = hf(x_0, y_0, u_0)$$
$$K_1 = 0.2 \times f_1(0, -0.4, -0.6)$$
$$K_1 = 0.2 \times (-0.6)$$
$$K_1 = -0.12$$

and,

$$K_1 = hf_2(x_0, y_0, u_0)$$
$$K_1 = 0.2 f_2(0, -0.4, -0.6)$$
$$K_1 = 0.2(0.8 - 1.2)$$
$$K_1 = -0.08$$

and,

$$K_2 = hf_1(x_0 + h, y_0 + K_1, u_0 + K_1)$$
$$K_2 = -0.136$$
$$K_2 = hf_2(x_0 + h, y_0 + K_1, u_0 + K_1)$$
$$K_2 = -0.00472404$$

so,

using, $y_{i+1} = y_i \left(\frac{K_1 + K_2}{2}\right)$

$$y_1 = y_0 + \left(\frac{-0.12 + (-0.136)}{2}\right)$$

$$\boxed{y_1 = -0.528}$$

for f_2

$i = 0$

$$y_1 = y_0 + \left(\frac{K_1 + K_2}{2}\right)$$

$$y_1 = -0.4 + \left(\frac{-0.08 - 0.00472404}{2}\right)$$

$$y_1 = -0.64236202$$

Now, $y_0(0) = -0.4$, $y_0'(0) = -0.6)$

$y_1 = -0.528$, $u_1 = -0.64236202$

Now, for y_2 & u_2

$x = 0.2$

$y_2 = -0.65511929$, $u_2 = -0.54340149$

$y_3 = -0.74199754$ $u_3 = -0.15923968$

$y_4 = -0.7130458$, $u_4 = 0.74461755$

$y_5 = -0.43474995$, $u_5 = 2.53370418$

2 Programming

IVP solver module

```
import numpy as np

def RK4_vec(IC,a,b,N,n,f,f1=None):
    x = np.linspace(a,b,int(N)+1)
    h = x[1] - x[0]
    S = np.zeros((len(x),n))
    S[0,:] = IC

    k1 = np.zeros([len(x),n])
    k2,k3,k4,K = k1.copy(),k1.copy(),k1.copy(),k1.copy()
    for i in range(len(x)-1):
        k1[i,:] = f(x[i],S[i,:],f1)
        k2[i,:] = f(x[i]+0.5*h,S[i,:]+k1[i,:]*0.5*h,f1)
        k3[i,:] = f(x[i]+0.5*h,S[i,:]+k2[i,:]*0.5*h,f1)
        k4[i,:] = f(x[i]+ h,S[i,:]+k3[i,:]*h,f1)

        K[i,:] = (k1[i,:] + 2*k2[i,:] + 2*k3[i,:] + k4[i,:])/6

        S[i+1,:] = S[i,:] + K[i,:]*h

    return x,S

def RK2_vec(IC,a,b,N,n,f,f1=None):
    x = np.linspace(a,b,int(N)+1)
    h = x[1] - x[0]
    S = np.zeros((len(x),n))
    S[0,:] = IC

    k1 = np.zeros([len(x),n])
    k2,K = k1.copy(),k1.copy()
    for i in range(len(x)-1):
        k1[i,:] = f(x[i],S[i,:],f1)
        k2[i,:] = f(x[i]+h,S[i,:]+k1[i,:]*h,f1)

        K[i,:] = (k1[i,:] + k2[i,:])/2

        S[i+1,:] = S[i,:] + K[i,:]*h

    return x,S


def Euler_vec(IC,a,b,N,n,f,f1=None):

    x = np.linspace(a,b,int(N)+1)
    h = x[1] - x[0]
    S = np.zeros((len(x),n))
    S[0,:] = IC

    K = np.zeros([len(x),n])
    for i in range(len(x)-1):
        K[i,:] = f(x[i],S[i,:],f1)
        S[i+1,:] = S[i,:] + K[i,:]*h

    return x,S


#question specific


```

```python
if __name__ == "__main__":

    import matplotlib.pyplot as plt
    def slope(x,S):
        return -S[0] + S[1] - S[1]**3

    def function(x,S,f1):
        return [*S[1:],f1(x,S)]

    Ic_e = [-1,0]

    X_4,last_4 = RK4_vec(Ic_e, 0, 15, 100, 2, function,slope)
    X_2,last_2= RK2_vec(Ic_e, 0, 15, 100, 2, function,slope)
    X_e,last_e = Euler_vec(Ic_e, 0, 15, 100, 2,function,slope)

    plt.plot(X_4,last_4[:,0],'b--v')
    plt.plot(X_2,last_2[:,0],'r--o')
    plt.plot(X_e,last_e[:,0],'1--')
    plt.show()
```

Specific solution and comparison of various numerical methods

```python
import numpy as np
import matplotlib.pyplot as plt
from MyIVP import *
import pandas as pd
from scipy import stats


def slope(x,S,f1=None):
    temp = np.zeros(len(S))
    temp[0] = S[1] - S[2] + x
    temp[1] = 3*x**2
    temp[2] = S[1] + 1/(np.exp(x))

    return temp

Anfunc_lis   = [lambda x:-0.05*x**5 + 0.25*x**4 + x + 2 - (1/(np.exp(x))), lambda x:
    x**3 + 1,lambda x:0.25*x**4 + x  - 1/(np.exp(x))]

Anfunc_lis[0] = np.vectorize(Anfunc_lis[0])
Anfunc_lis[1] = np.vectorize(Anfunc_lis[1])
Anfunc_lis[2] = np.vectorize(Anfunc_lis[2])

def Plotting(X_l,Y_e,Y_2,Y_4,func,key = None,key2 = None):
    N = len(X_l)
    fig,ax = plt.subplots()
    ax.plot(X_l,Y_e,'r--*',label = 'Euler method')
    ax.plot(X_l,Y_2,'b--o',label = 'RK2 method')
    ax.plot(X_l,Y_4,'g--v',label = 'RK4 method')
    ax.plot(X_l,func(X_l))
    ax.set_title(f'Solution for Y{key} for xf = {key2} with N = {N}')
    ax.set_xlabel('Independent variable X')
    ax.set_ylabel(f'Y{key}')
    ax.legend()
    ax.grid()
    #plt.savefig(f'func{key}-x{key2}.png')
    plt.show()

def Error(a,b,f_an,key,N_d,method):
    N_list = np.logspace(1,N_d,base = 10,num = int(N_d))
    h = (b-a)/N_list
    E = np.zeros(len(N_list))
```

```
    P = []
    for i in range(len(N_list)):
        x,D = method(In_c,a,b,N_list[i],3,slope)
        Y_an = f_an(x)
        E[i] = max(abs(D[:,key] - Y_an))
    return np.log10(N_list),np.log10(h),np.log10(E)

Error = np.vectorize(Error)

In_c = [1,1,-1]
x0 = 0 ; x_e = 1
N1 = 10
x_f = [1,2.5,5,7.5,10]
N_l = [10*i for i in x_f] #  keeping the step size constant

#d part
Y_e,Y_2,Y_4 = [],[],[]
X_e = []
for i in range(len(x_f)):
   x1_e,y_e = Euler_vec(In_c, 0, x_f[i], N_l[i], 3, slope)
   x1_2,y_2 = RK2_vec(In_c, 0, x_f[i], N_l[i], 3, slope)
   x1_4,y_4 = RK4_vec(In_c, 0, x_f[i],N_l[i], 3, slope)
   X_e.append(x1_e) ; Y_e.append(y_e)
   Y_2.append(y_2) ;  Y_4.append(y_4)

'''
for i in range(len(x_f)):
    for j in range(len(Anfunc_lis)):
        Plotting(X_e[i],Y_e[i][:,j],Y_2[i][:,j],Y_4[i][:,j],Anfunc_lis[j],j+1,x_f[i])
'''

# e Part
M_e = []
M_Rk2,M_Rk4 = [],[]

for j in range(len(Anfunc_lis)):
    for i in x_f:
        M_er = Error(0,i,Anfunc_lis[j],j,3,Euler_vec)
        M_rk2 = Error(0,i,Anfunc_lis[j],j,3,RK2_vec)
        M_rk4 = Error(0,i,Anfunc_lis[j],j,3,RK4_vec)
        M_e.append(M_er)
        M_Rk2.append(M_rk2)
        M_Rk4.append(M_rk4)

def Plotting_2(D1,key,title):
    type1 = {0 :'N',1 : 'h'}
    p = type1[key]
    fig,ax = plt.subplots(1,3)
    plt.gca().legend(('y0','y1'))
    fig.suptitle(title)

    for i in range(len(x_f)):
        ax[0].plot(D1[0][key],D1[i][2],'--*',label = f'for xf = {x_f[i]}')
        ax[1].plot(D1[0][key],D1[i+5][2],'--v')
        ax[2].plot(D1[0][key],D1[i+10][2],'--o')

        for j in range(3):
            ax[j].set_title(f'function {j+1}')
            ax[j].set_xlabel(f'log({p})')
            ax[j].set_ylabel('log(E)')
        fig.legend()
    #plt.savefig(f'{title}-for-{p}.png')
```

```
    plt.show()
'''
Plotting_2(M_e,0,'Euler method')
Plotting_2(M_Rk2,0,'RK2 method')
Plotting_2(M_Rk4,0,'RK4 method')
Plotting_2(M_e,1,'Euler method')
Plotting_2(M_Rk2,1,'RK2 method')
Plotting_2(M_Rk4,1,'RK4 method')
'''
def slope_err(D):

    slope = np.zeros(len(x_f))
    slope_2,slope_3 = slope.copy(),slope.copy()
    for i in range(len(x_f)):

        slope[i], intercept, r_value, p_value, std_err = stats.linregress(D[i][1],D[i][2])
        slope_2[i], intercept, r_value, p_value, std_err = stats.linregress(D[i+5][1],D[i+5][2])
        slope_3[i], intercept, r_value, p_value, std_err = stats.linregress(D[i+10][1],D[i+10][2])
    return slope,slope_2,slope_3

'''
f1,f2,f3 = slope_err(M_Rk2)

data2 = {'final x':x_f,'Function 1':f1,'Function 2':f2,'function 3': f3}
df2 = pd.DataFrame(data = data2)

print(df2)
df2.to_csv('slope_rk2.csv')
'''

def diff_tolerence(In_c,a,b,m,f,N_max,tol,var,method,f1 = None):
    max_n = np.floor(np.log2(N_max))
    n_array = np.logspace(2,max_n,base=2,num = int(max_n)-1)
    H = []
    G = []
    for i in range(len(n_array)):
        x,y = method(In_c,a,b,n_array[i],m,f,f1=None)
        H.append(y)
        G.append(x)
    for i in range(len(n_array)-1):
        Y = np.zeros((2,int(n_array[i])+1))
        Y[0] = H[i][:,var-1]
        J = H[i+1][:,var-1]
        Y[1] = J[::2]
        den = np.reciprocal(Y[1])
        ty = abs(Y[1] - Y[0])
        err =  max(np.multiply(ty,den))

        if err <= tol:
           return n_array[i+1]
    return

#Tolerence table
'''
M1 = np.zeros(len(x_f)-2)
M2,M3 = M1.copy(),M1.copy()
for i in range(len(x_f)-2):
    M1[i] = diff_tolerence(In_c,0,x_f[i],3,slope,2**18,0.5*10**(-3),1,Euler_vec)
    M2[i] = diff_tolerence(In_c,0,x_f[i],3,slope,2**16,0.5*10**(-3),1,RK2_vec)
```

```
    M3[i] = diff_tolerence(In_c,0,x_f[i],3,slope,2**16,0.5*10**(-3),1,RK4_vec)

d1 = {'final x' : x_f[:3],'Euler method':M1,'RK2 method':M2,'RK4 method':M3}
df = pd.DataFrame(data = d1)
print(df)
df.to_csv('tolerence.csv')
'''


#Programming part

def slope_2(x,S,f1 = None):
    temp = np.zeros(len(S))
    temp[0] = S[1]
    temp[1] = np.exp(2*x)*np.sin(x) - 2*S[0] + 2*S[1]

    return temp

ic = [-0.4,-0.6]

x,T = RK2_vec(ic, 0, 1, 5, 2, slope_2)

data3 = {'Xi':np.linspace(0,1,6),'Yi':T[:,0]}

df3 = pd.DataFrame(data = data3)

df3.to_csv('last.csv')

x_e,T_e = Euler_vec(ic,0,1,20,2,slope_2)
x_r2,T_r2 = RK2_vec(ic,0,1,20,2,slope_2)
x_r4,T_r4 = RK4_vec(ic,0,1,20,2,slope_2)

plt.plot(x_e,T_e[:,0],label = 'Euler method')
plt.plot(x_r2,T_r2[:,0],label = 'RK2 method')
plt.plot(x_r4,T_r4[:,0],label = 'RK4 method')
plt.title('Numerically calculated y for h = 0.05')
plt.savefig('Last.png')
plt.legend()
plt.show()
```

3 Discussion

3.1 final point x = 1

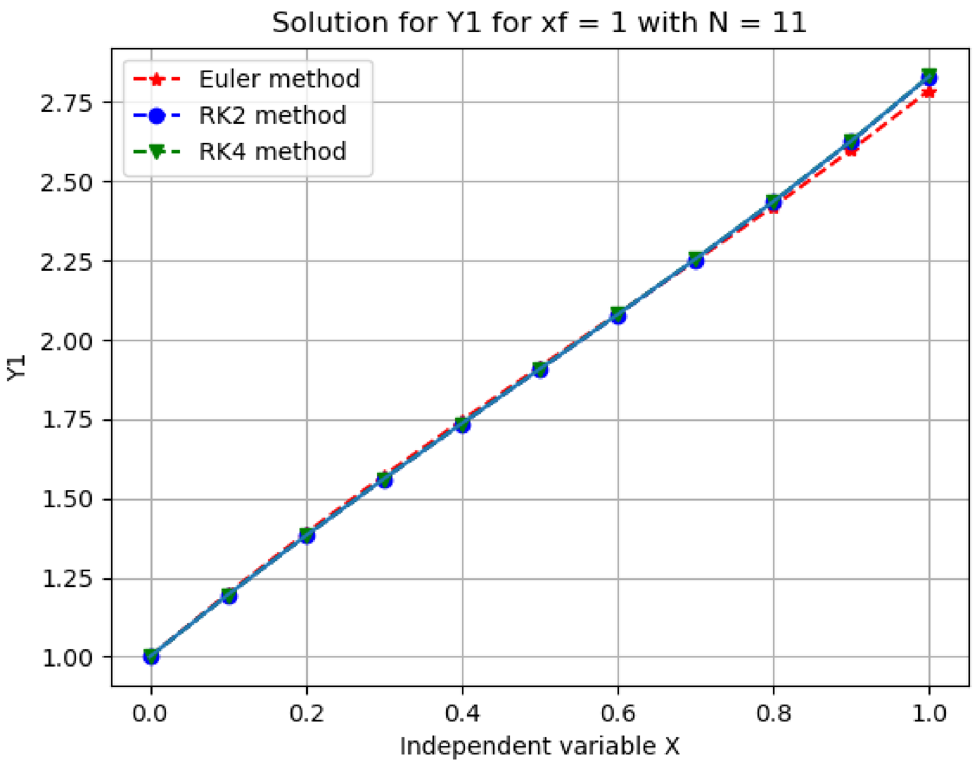

Figure 1: Y1 $= -0.05x^5 + 0.25x^4 + x + 2 + e^{-x}$

Note:
Here we have taken the No. of points at which we compute the Y_{num} to be 10 i.e, with a step size of h $=$ 0.1,hence In further graphs where we plot the function for different upper we keep **h constant**.

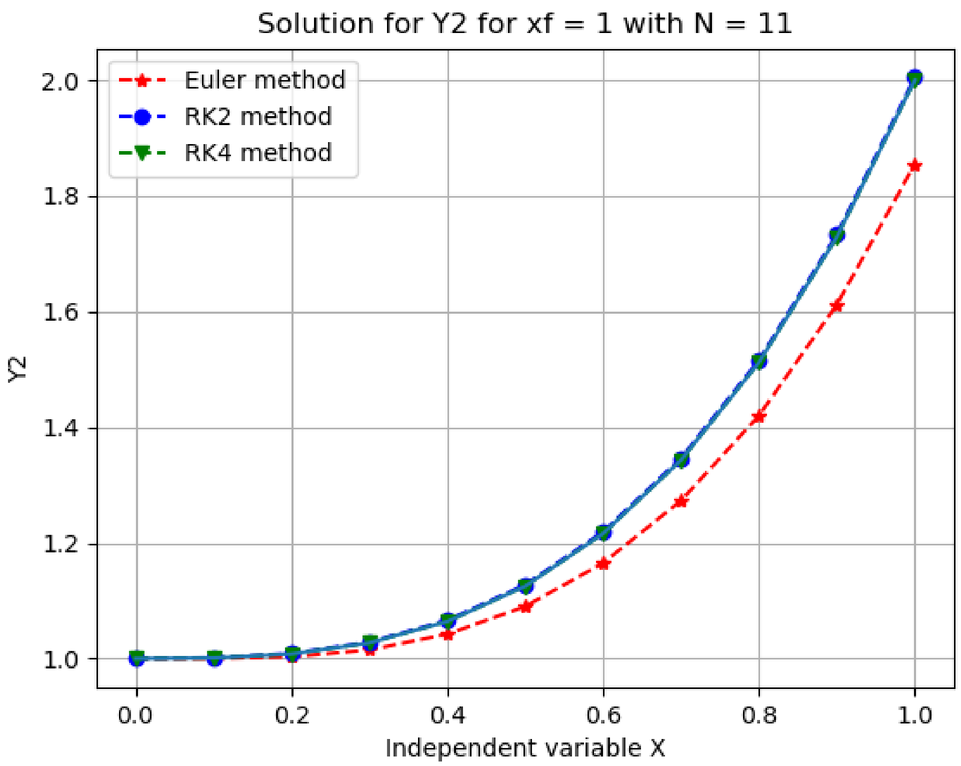

Figure 2: $\mathrm{Y2} = x^3 + 1$

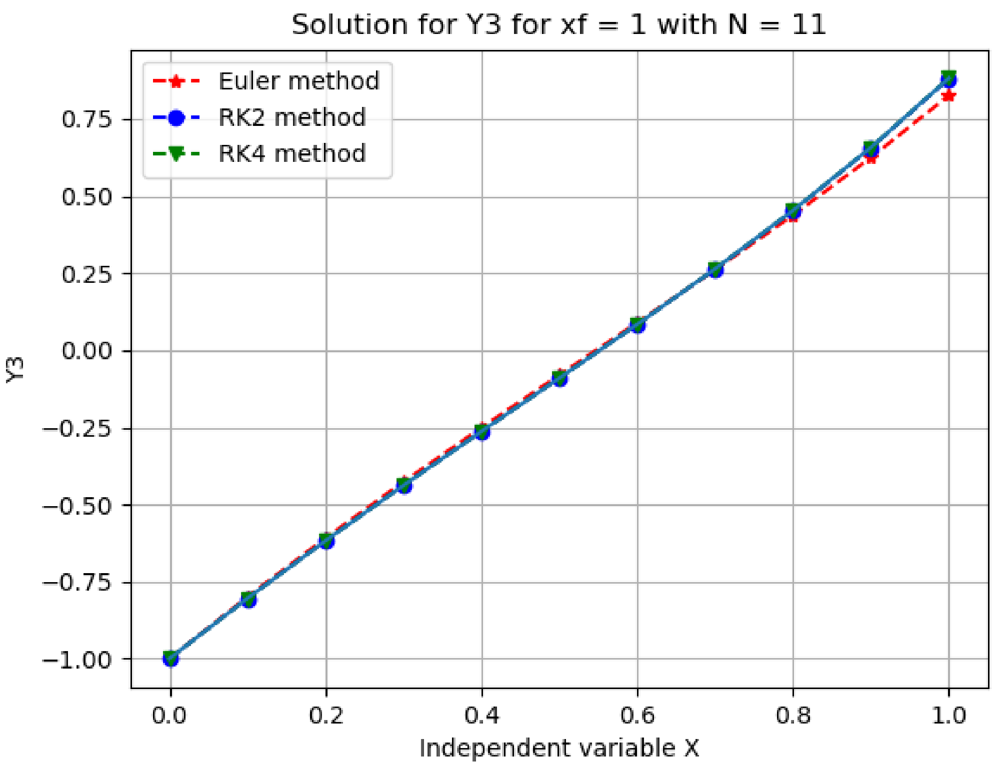

Figure 3: $\mathrm{Y3} = 0.25x^4 + x - e^{-x}$

Further we can see in all the three graphs as we reach the final the accuracy of euler method reduces and deviates from the analytic result

3.2 final point x = 2.5

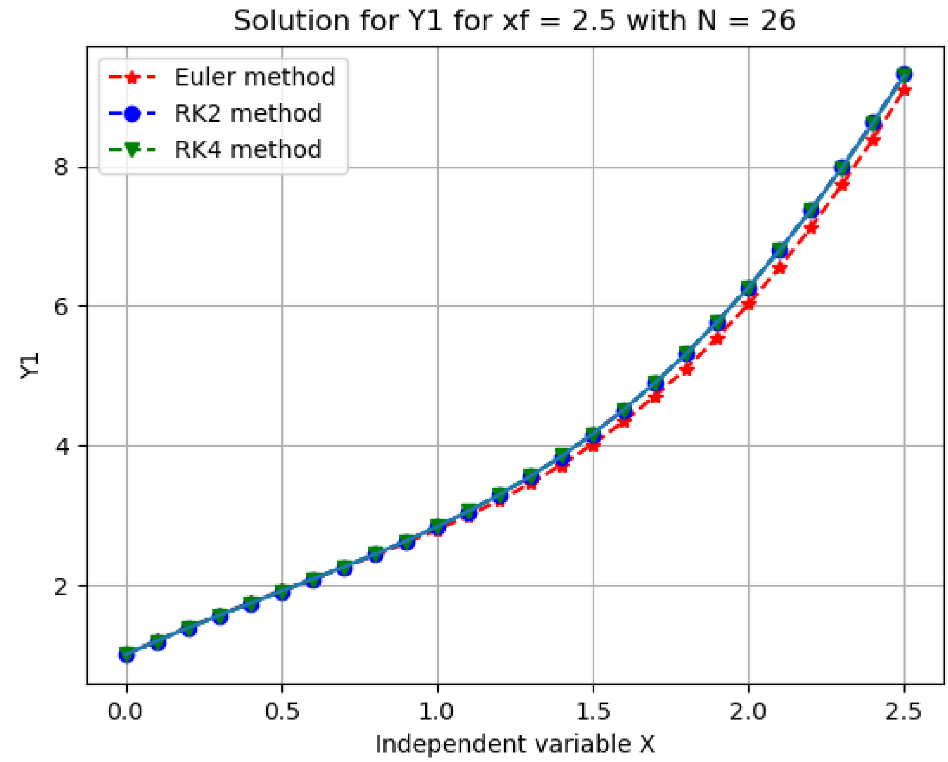

Figure 4: $Y1 = -0.05x^5 + 0.25x^4 + x + 2 + e^{-x}$

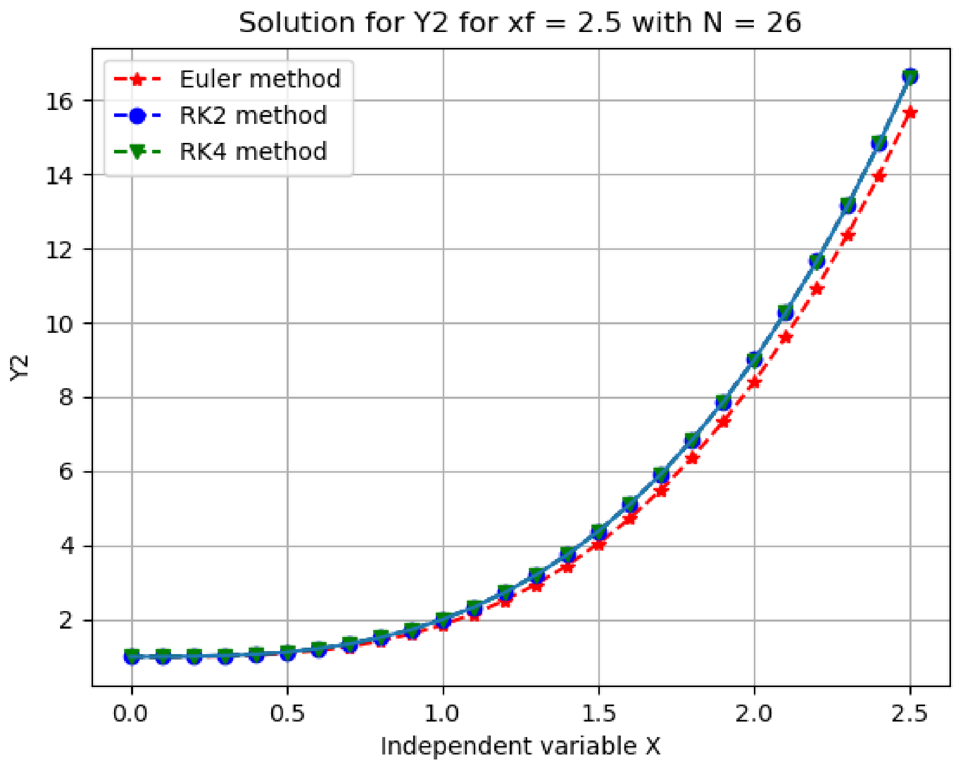

Figure 5: $Y2 = x^3 + 1$

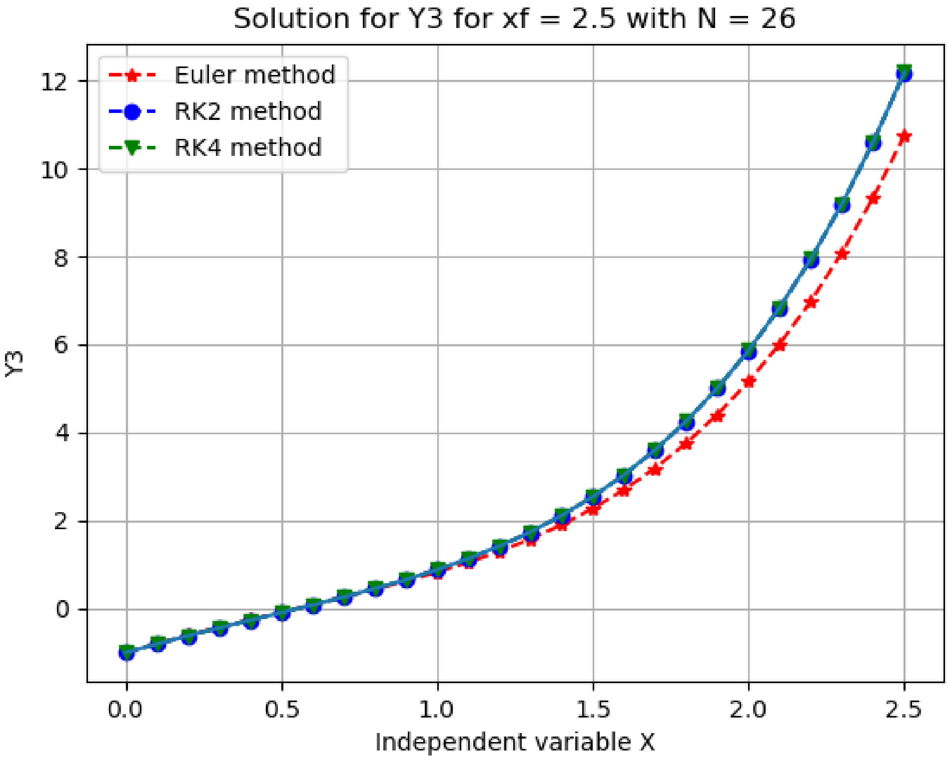

Figure 6: $\mathrm{Y3} = 0.25x^4 + x - e^{-x}$

3.3 final point x = 5

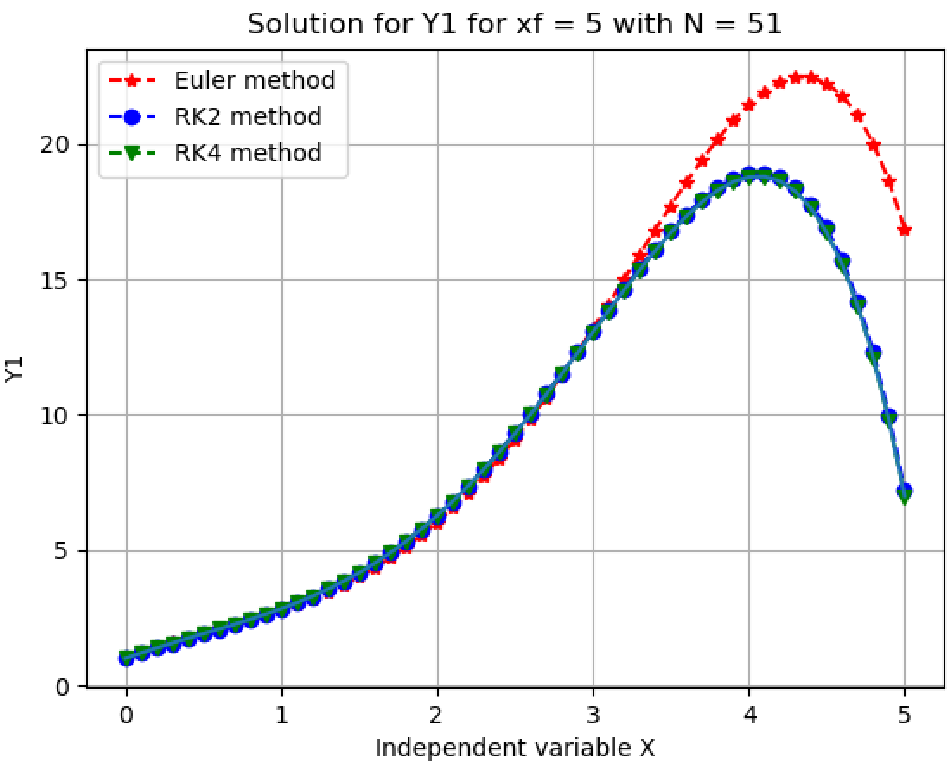

Figure 7: $\mathrm{Y1} = -0.05x^5 + 0.25x^4 + x + 2 + e^{-x}$

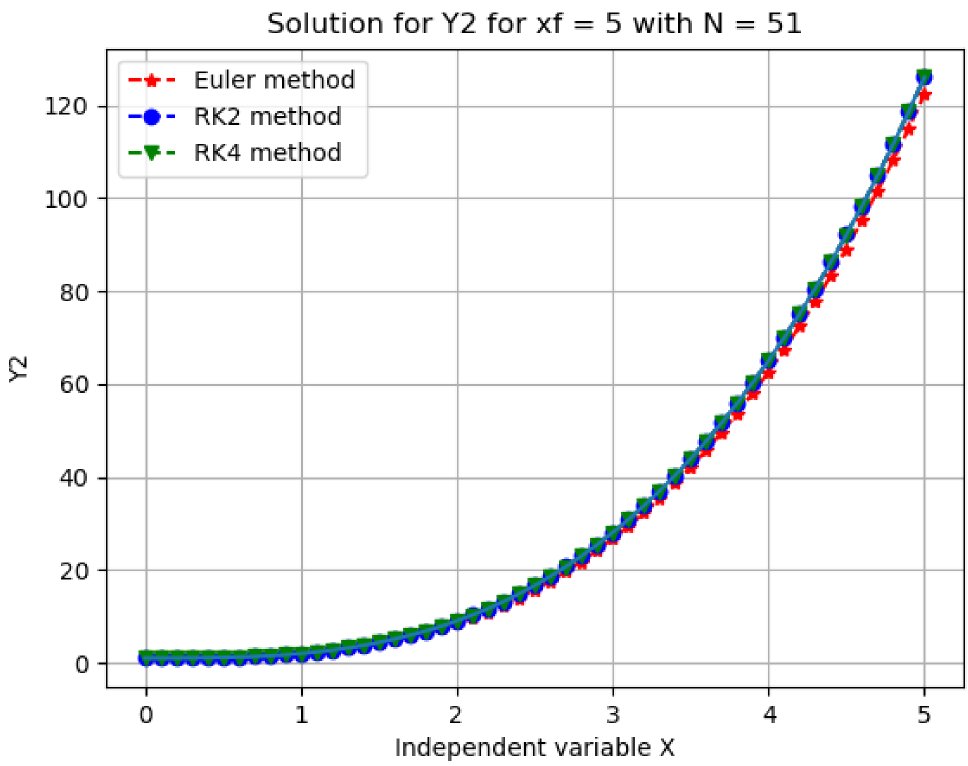

Figure 8: Y2 $= x^3 + 1$

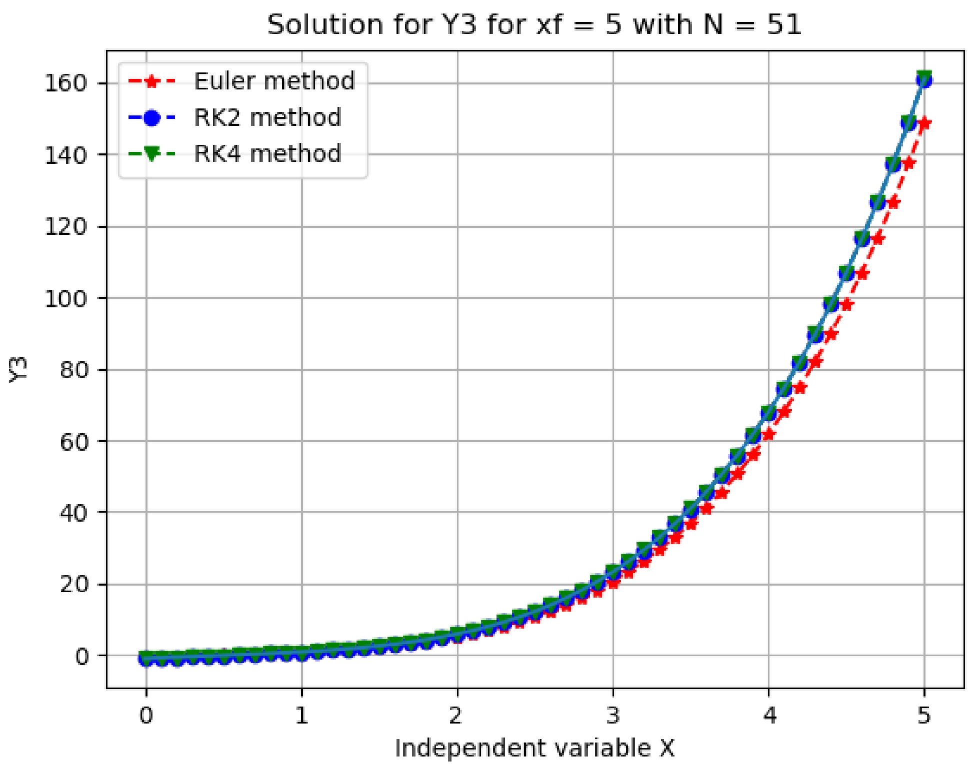

Figure 9: Y3 $= 0.25x^4 + x - e^{-x}$

Here we can see that RK4 and RK2 methods are very consistent with step size = 0.5, but the Euler method either overestimates pf underestimates the Y_{num}

3.4 final point x = 7.5

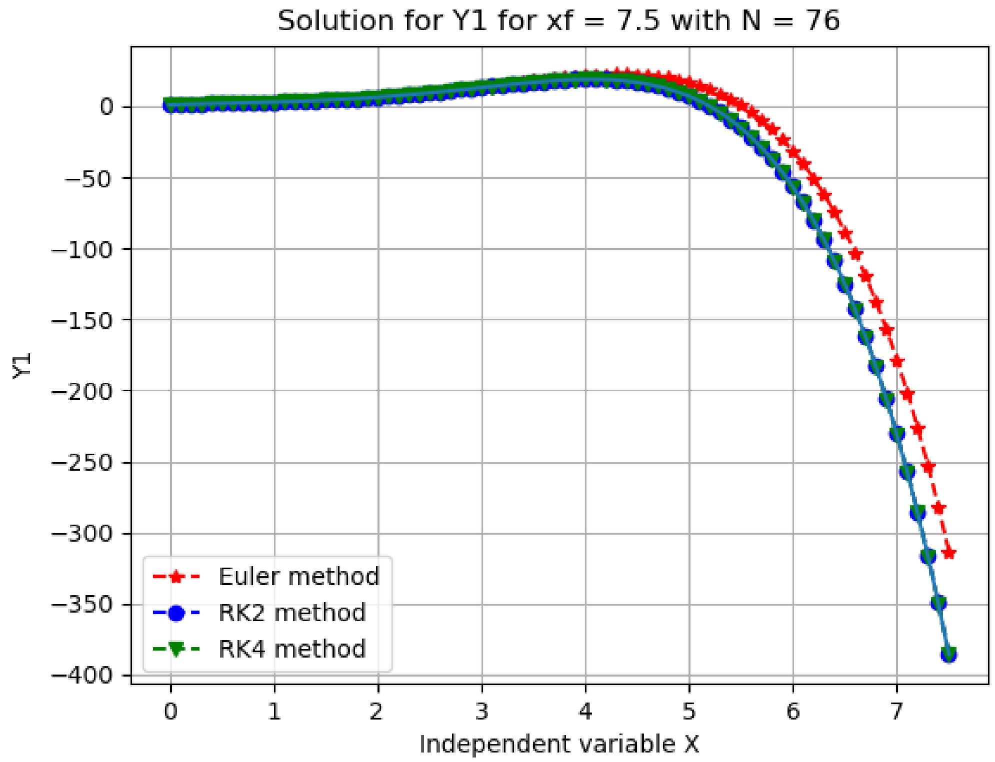

Figure 10: $Y1 = -0.05x^5 + 0.25x^4 + x + 2 + e^{-x}$

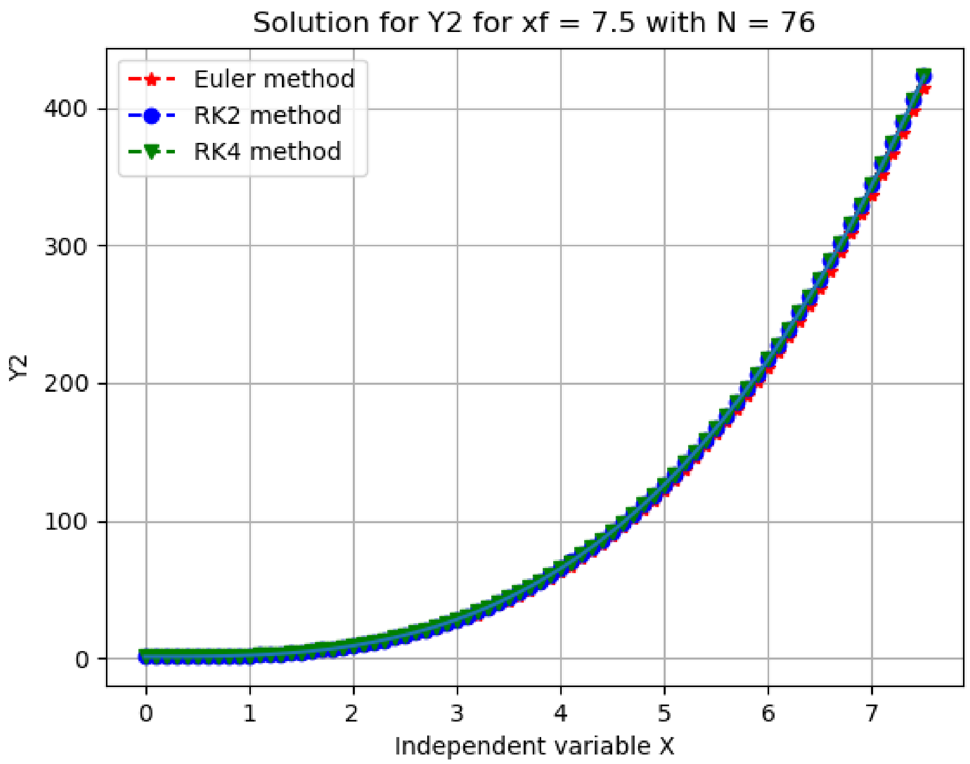

Figure 11: $Y2 = x^3 + 1$

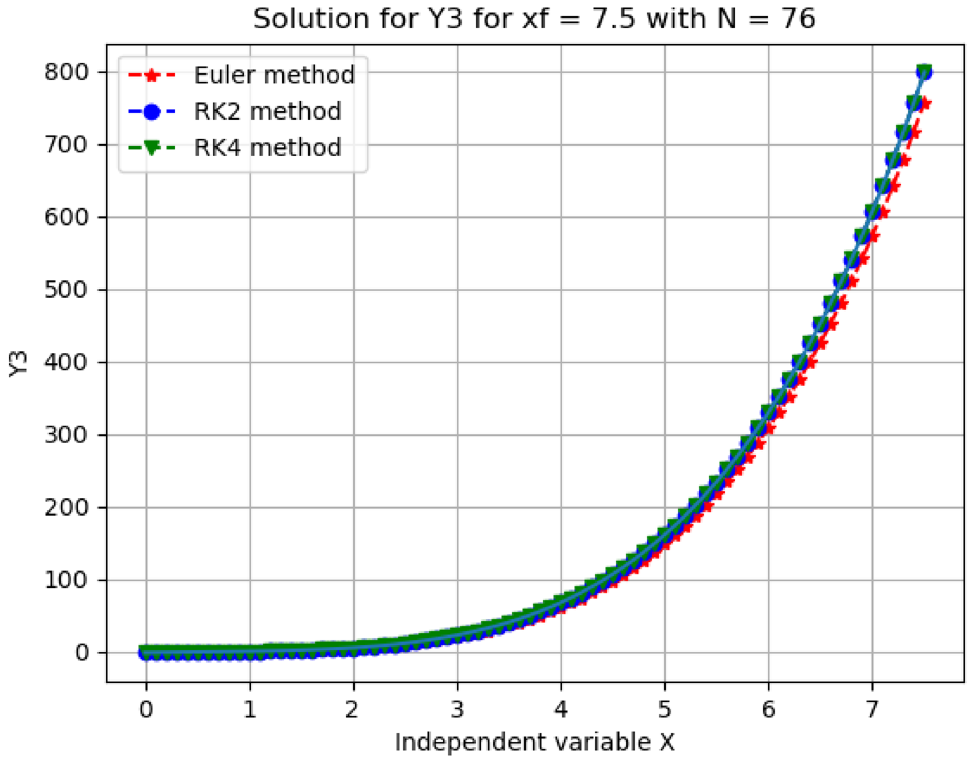

Figure 12: Y3 $= 0.25x^4 + x - e^{-x}$

3.5 final point x = 10

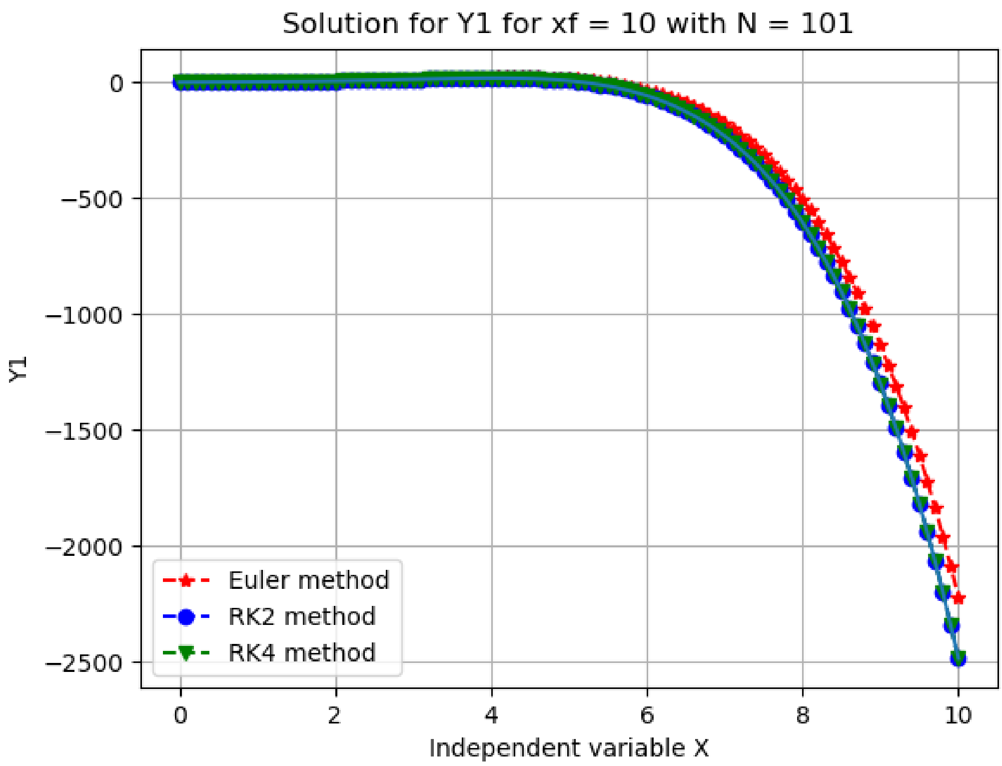

Figure 13: Y1 $= -0.05x^5 + 0.25x^4 + x + 2 + e^{-x}$

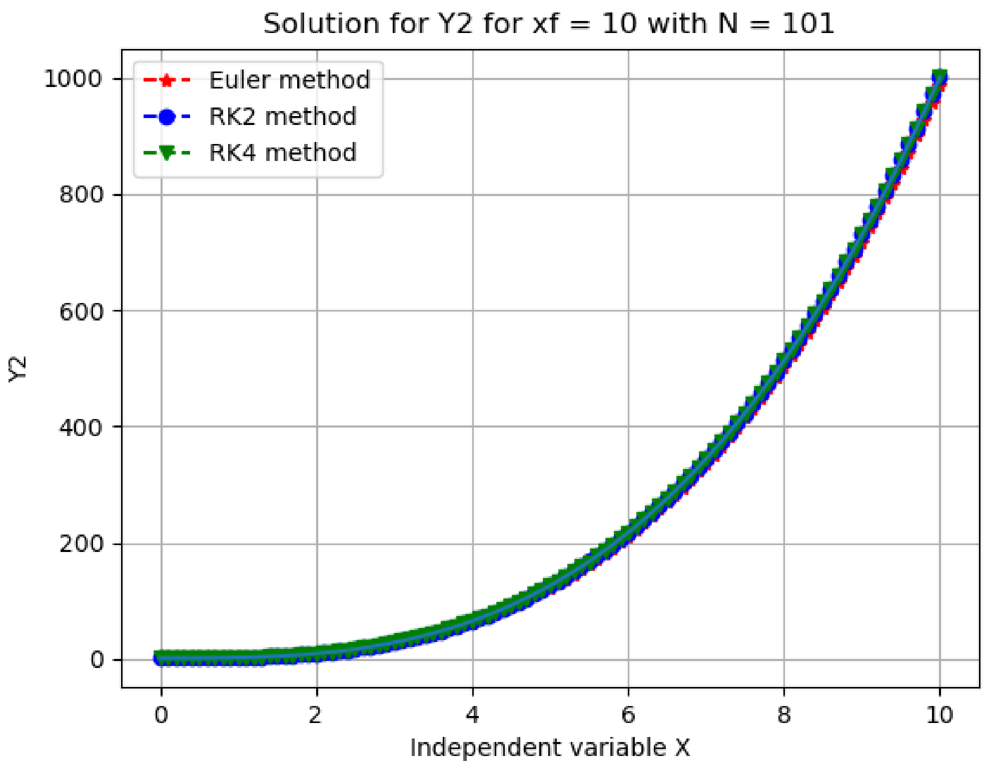

Figure 14: $\text{Y2} = x^3 + 1$

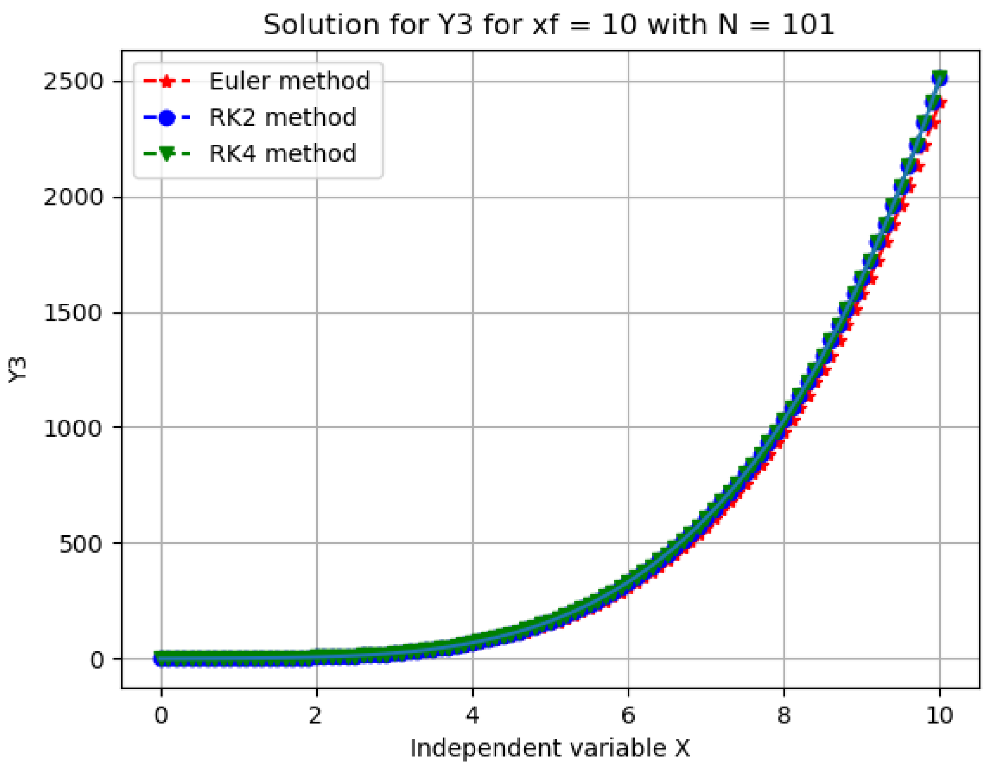

Figure 15: $\text{Y3} = 0.25x^4 + x - e^{-x}$

3.6 Error comparison of Euler,RK2,RK4 methods

Euler method

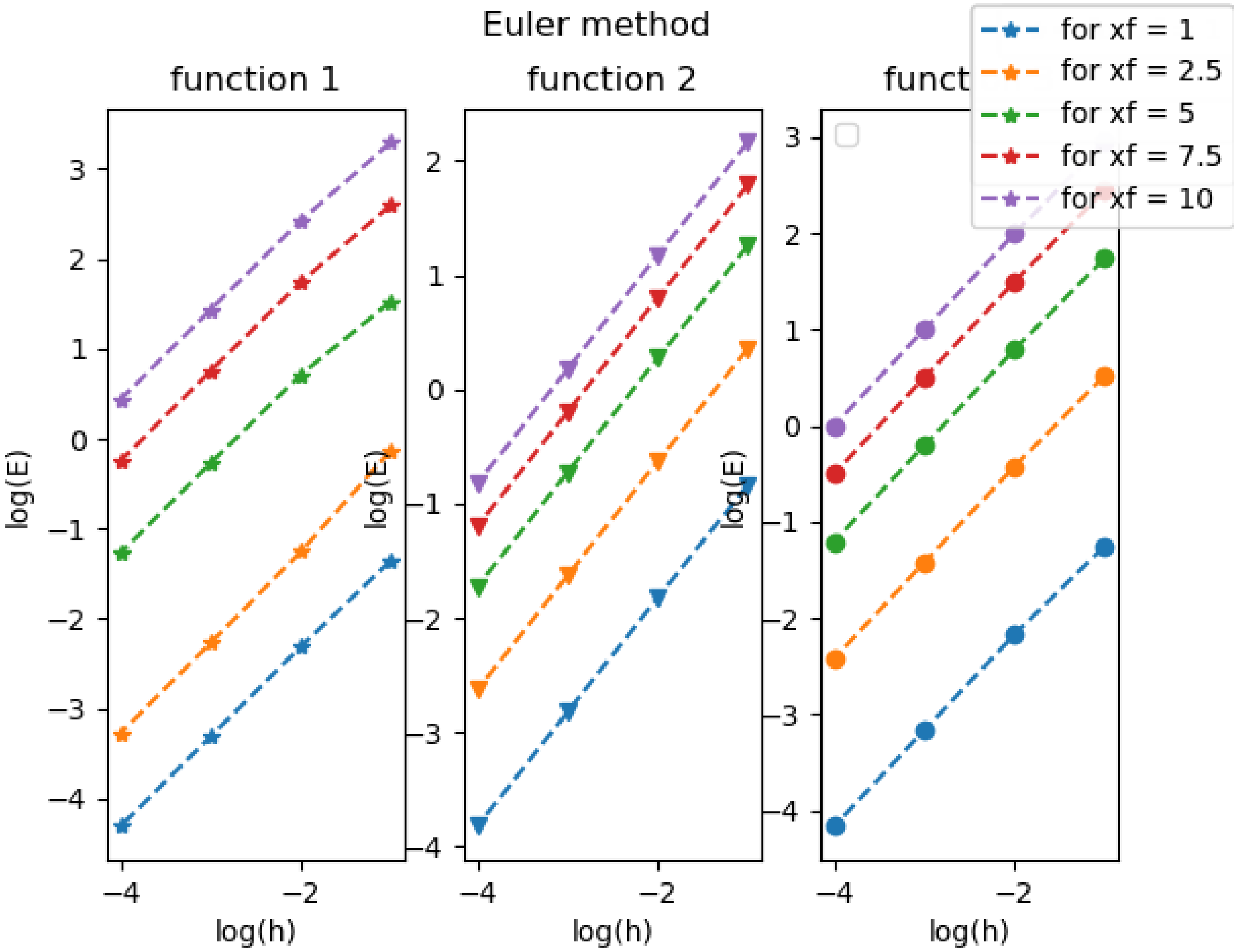

Figure 16: Log(E) vs Log(h) plot

	final x	Function 1	Function 2	function 3
0	1.0	0.9772877122137696	0.9927107660700054	0.9564837280727676
1	2.5	1.0686180902412898	0.9927107660697951	0.9713135148284173
2	5.0	0.8979844944162363	0.9927107660698116	0.972314745771142
3	7.5	0.9220174319143221	0.992710766069661	0.9724421057919376
4	10.0	0.9289809286839019	0.9927107660698116	0.9724780956803644

Table 1: Slopes of error line of Euler method

- We Know that **Global truncation error** in Euler method reduces proportional to the step size,Hence the error line has a **slope 1**
- Note that as we increase the final point of computation the error in the method increases for that step size that is because,We have kept the N constant for the all the limits that's why step size size changes for each computation with different step size,**Hence for larger upper limit we have,larger step size and for larger step size the error is more**
- that's the reason the error line shifts upwards for increase in upper limit

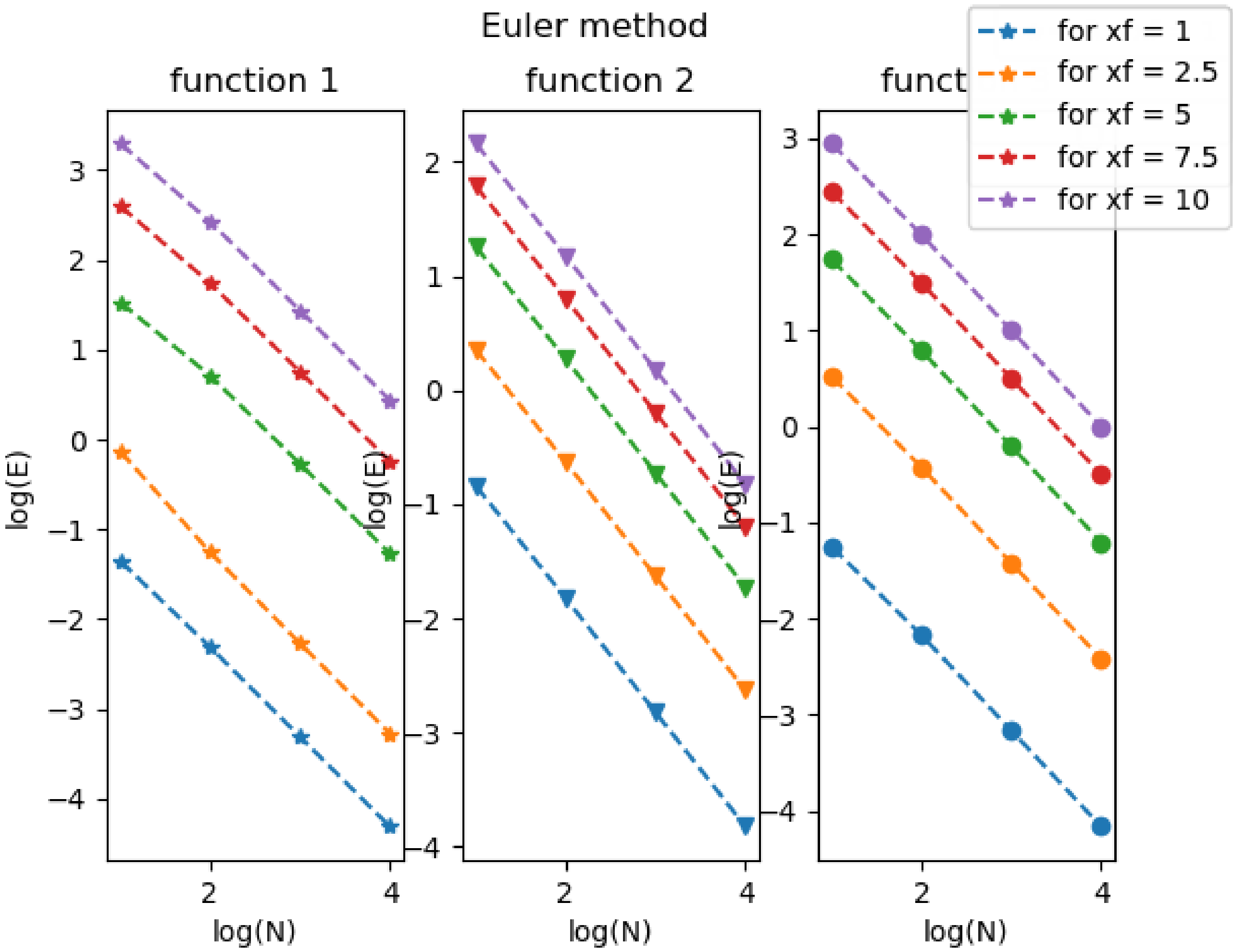

Figure 17: Log(E) vs Log(N) plot

The Error should reduces as we increase the N,hence the error line will have the negative slope that of the error line depending on h

RK2 method

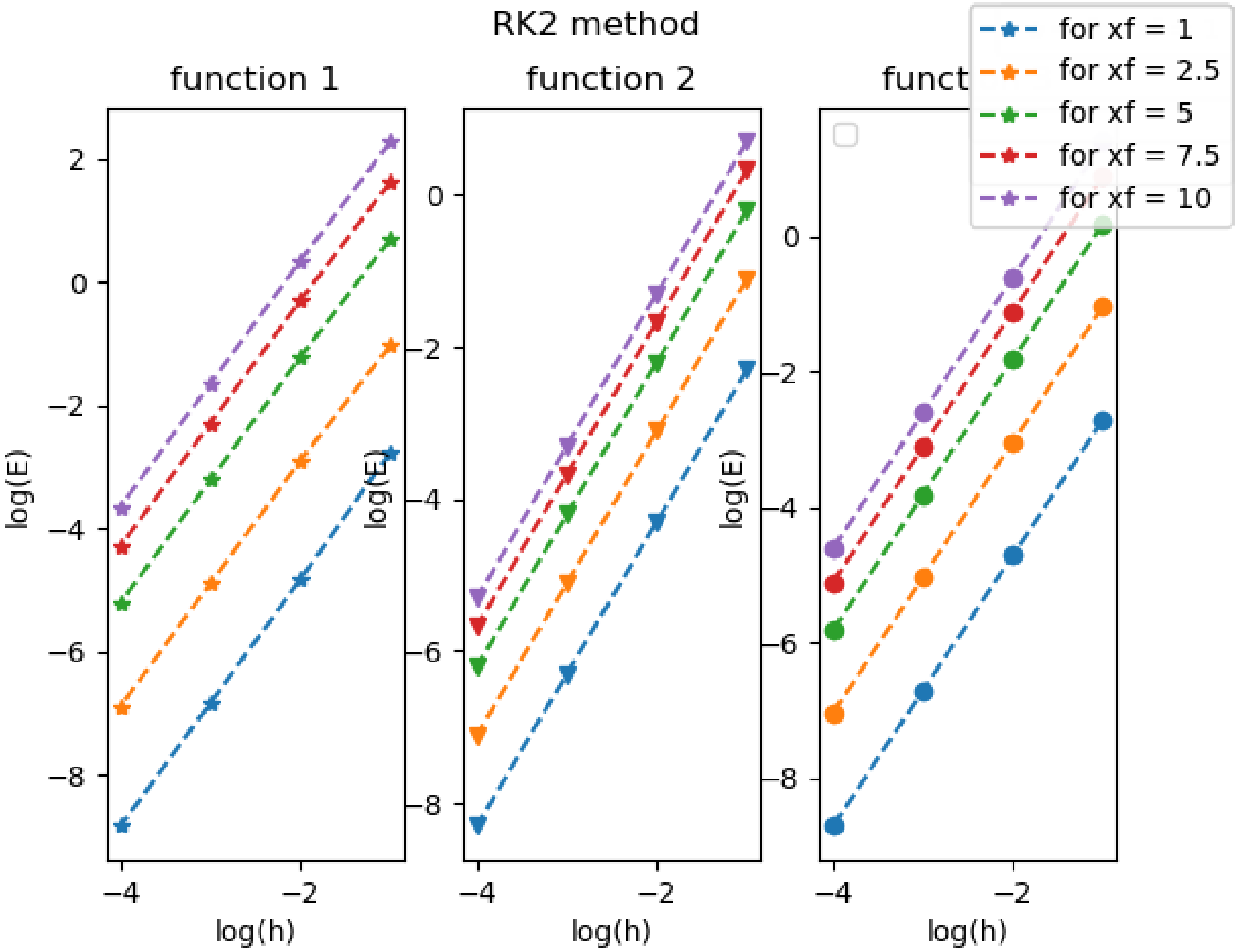

Figure 18: Log(E) vs Log(h) plot

	final x	Function 1	Function 2	function 3
0	1.0	2.037552215459998	1.9999999986196038	2.000009657430583
1	2.5	1.9319576828229992	1.9999999996595197	2.0000116248583937
2	5.0	1.956717557276991	1.99999999965952	2.0000120702388586
3	7.5	1.9609099882355239	2.000000000219087	2.0000119671271315
4	10.0	1.9626426051668409	1.99999999965952	2.0000118209888984

Table 2: Slopes of error line of Euler method

- As the theory suggests that global error must depend on h^2 we can see that the slope of error line is very close to 2 for all the functions and final condition

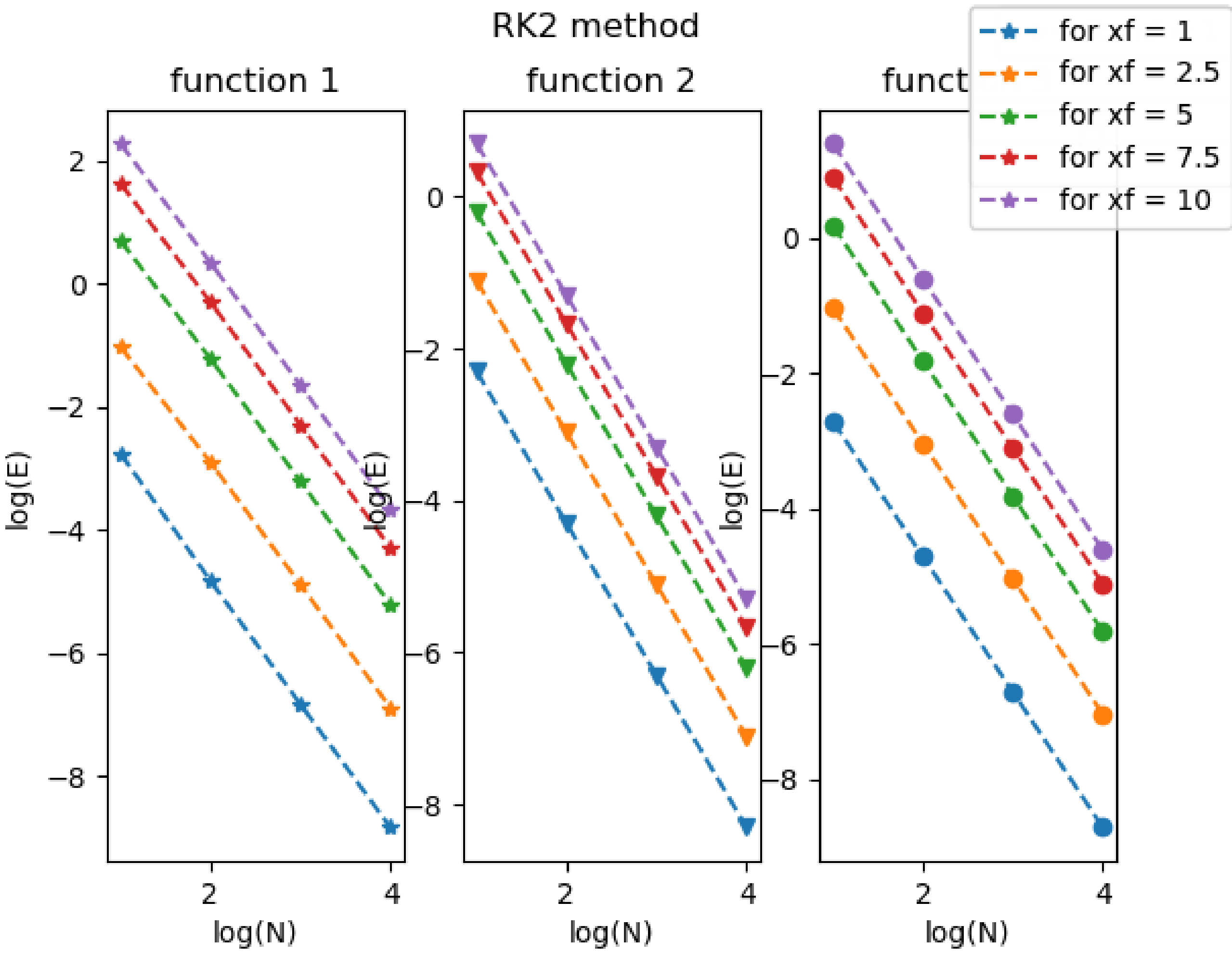

Figure 19: Log(E) vs Log(N) plot

RK4 method

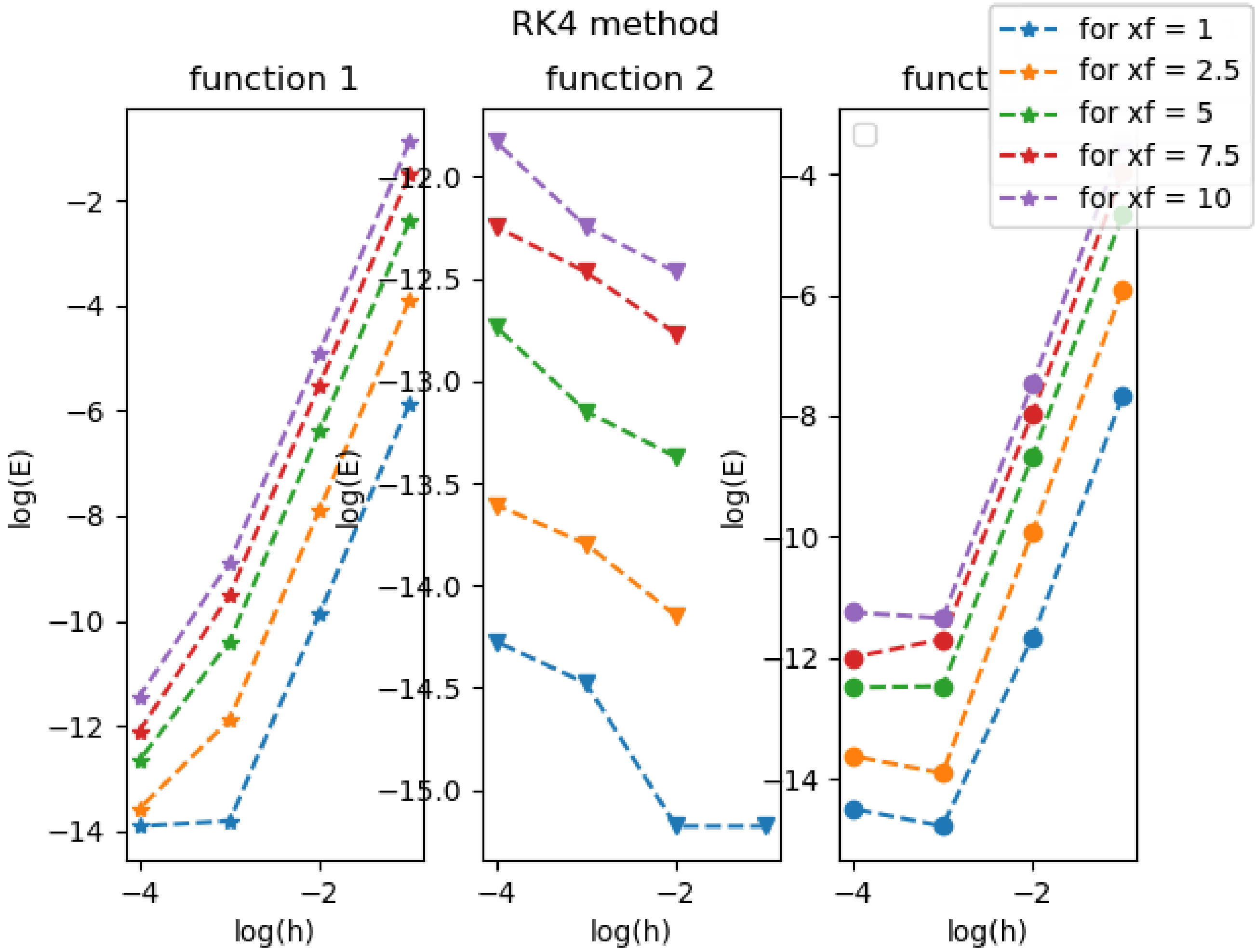

Figure 20: Log(E) vs Log(h) plot

	final x	Function 1	Function 2	function 3
0	1.0	3.969482202560141	-0.3494850021680094	3.5598883464288504
1	2.5	3.9982777064734694		3.999867247721645
2	5.0	3.9995947500768736		3.898748861137624
3	7.5	3.9994521403632115		3.861202563497847
4	10.0	3.9995980531247834		3.935018395904387

Table 3: Slopes of error line of Euler method

- Note that the error plot for second function is very different from the functions 1 and 3 that's beacause RK4 is a 4th order method i.e, it can exactly calculate the Y_{num} at desired x for a polynomial upto degree 4,as we have $Y2 = x^3 + 1$,Hence the error present is only due to the roundoff error of python
- further note that as we reduced the step size more and more for function 1 and 3 we see that truncation error diminishes and roundoff takes over
- further we see that as theory suggests the global error dependence as h^4,the slope of error lines is close to 4.
- note that we also could not find a slope of error regression line for function 2

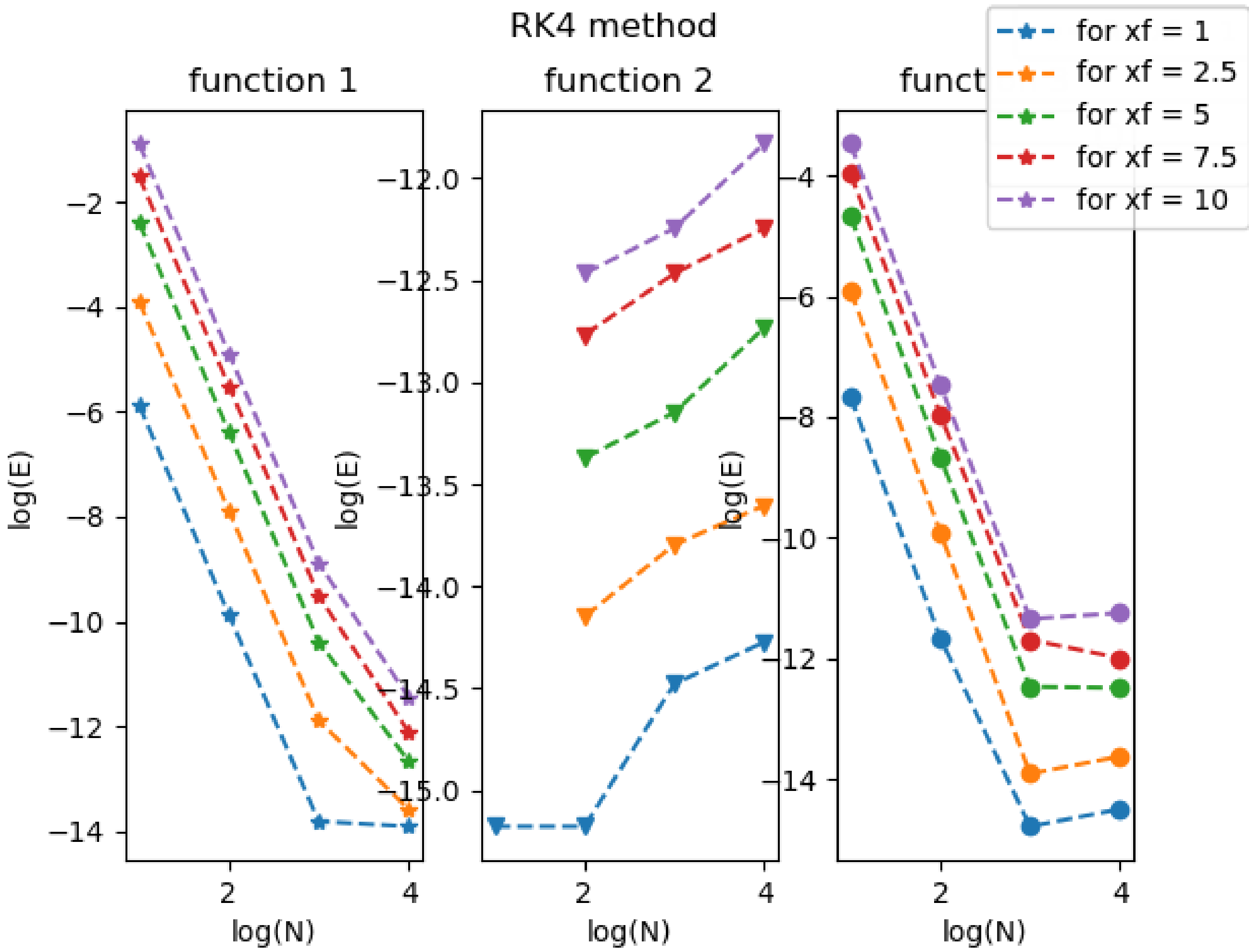

Figure 21: Log(E) vs Log(N) plot

3.7 Accuracy of numerical methods

We have checked the accuracy of all the three methods by implying a accuracy bound of 3 significant digits in the calculation of Y_{num}.

	final x	Euler method	RK2 method	RK4 method
0	1.0	512.0	32.0	8.0
1	2.5	2048.0	128.0	16.0
2	5.0	262144.0	1024.0	32.0

Table 4: different N required for achieving dersired accuracy

We can infer from here that how good a method RK4 is for solving IVP,for achieving a accuracy of 3 significant digits RK4 method only requires 32 calculations in between 0 and 10 whilst Euler method requires 262144 calculations,which is very computationally expensive.

3.8 Solution of second order DE

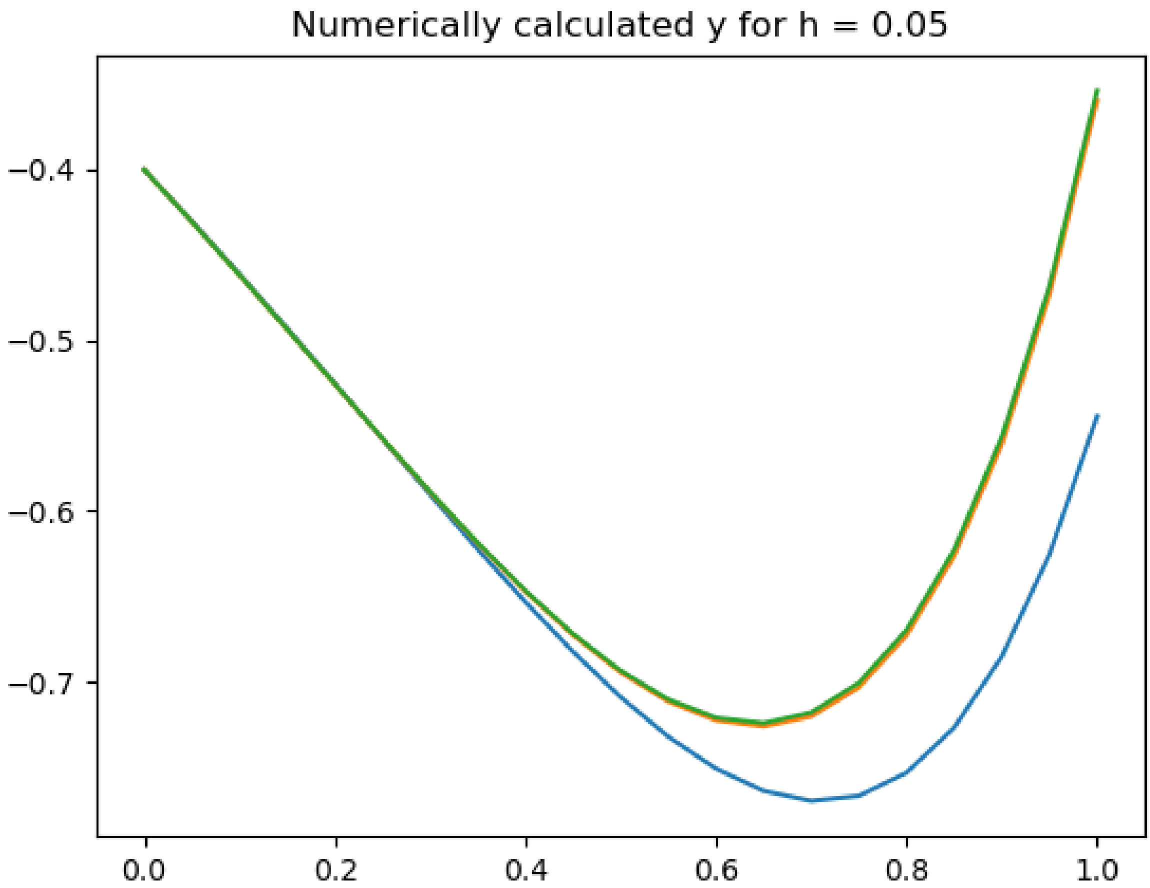

Figure 22: x vs y

	Xi	Yi
0	0.0	-0.4
1	0.2	-0.528
2	0.4	-0.6551192881688669
3	0.6000000000000001	-0.7419975442954315
4	0.8	-0.7130415839701495
5	1.0	-0.4347499488069328

Table 5: Y calculated using RK2 method