RESULTS AND DISCUSSION

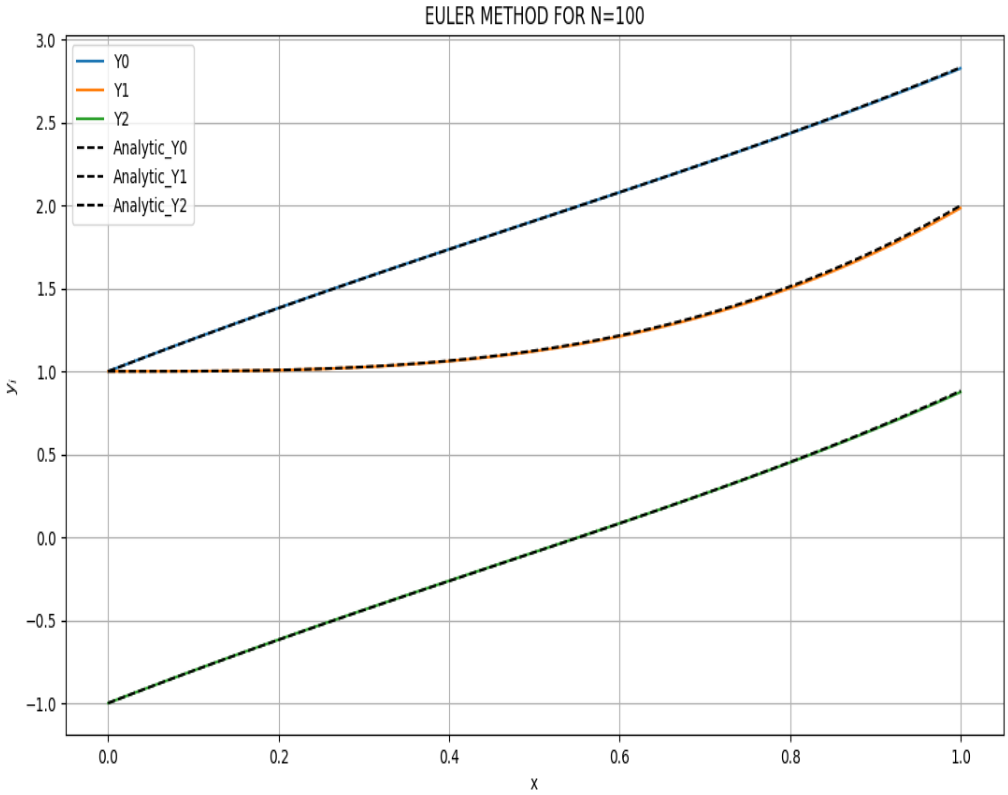

Figure 1: *y_i vs x plot using Euler Method.*

Above plots show y_i vs x values in range x $\epsilon[0, 1]$ for 100 steps by euler method.The analytic results are also plotted along with numerical ones as dashed black line.

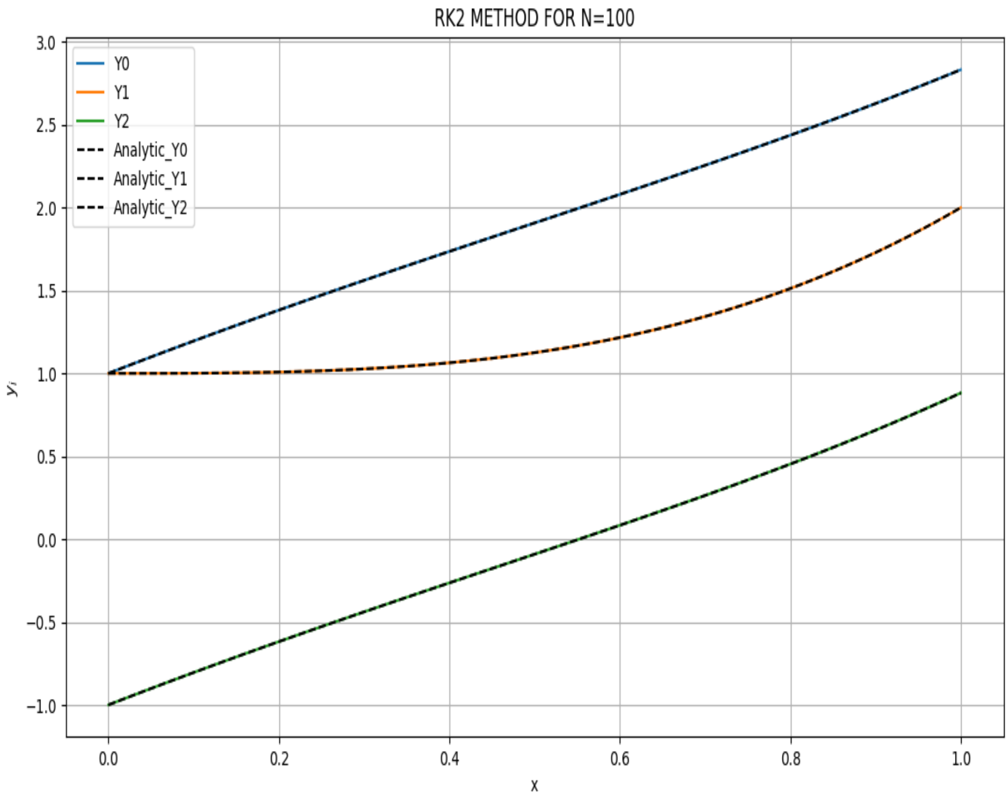

Figure 2: y_i *vs* x *plot using RK2 Method.*

Above plots show y_i vs x values in range $x\ \epsilon[0,1]$ for 100 steps by RK2 method.The analytic results are also plotted along with numerical ones as dashed black line.

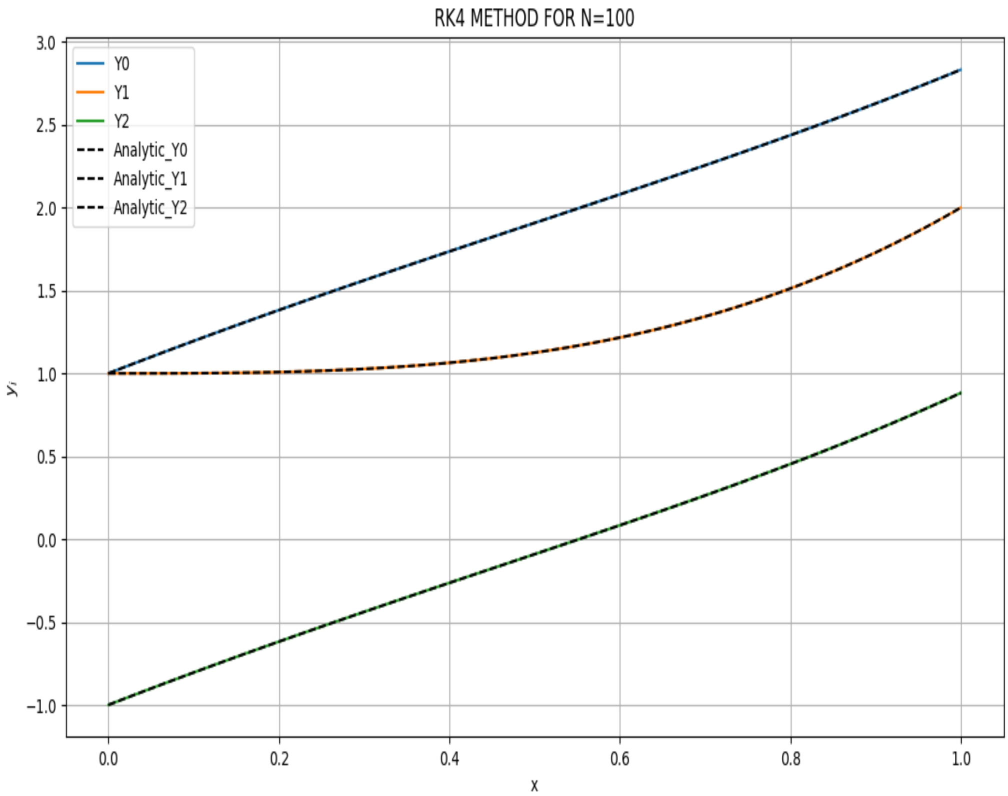

Figure 3: *y_i vs x plot using RK4 Method.*

Above plots show y_i vs x values in range $x\ \epsilon[0,1]$ for 100 steps by RK4 method.The analytic results are also plotted along with numerical ones as dashed black line.

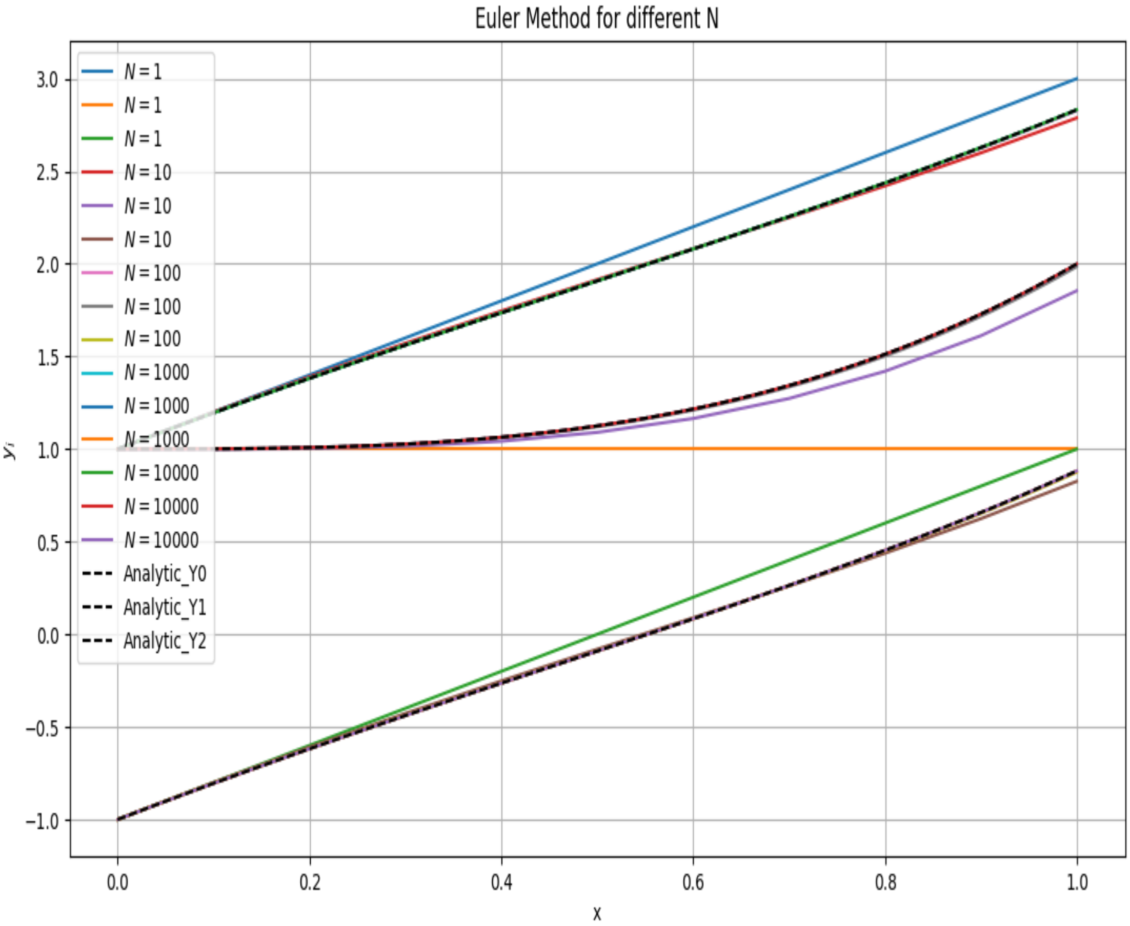

Figure 4: *y_i vs x plot using Euler Method for different N.*

The above plot is between y_i vs x for increasing N(Number of intervals) by Euler method , it can be understood that as the value of N increases , the plot starts overlapping with that of analytic one. In other words, the numerical results start converging with as the value of N is increased.

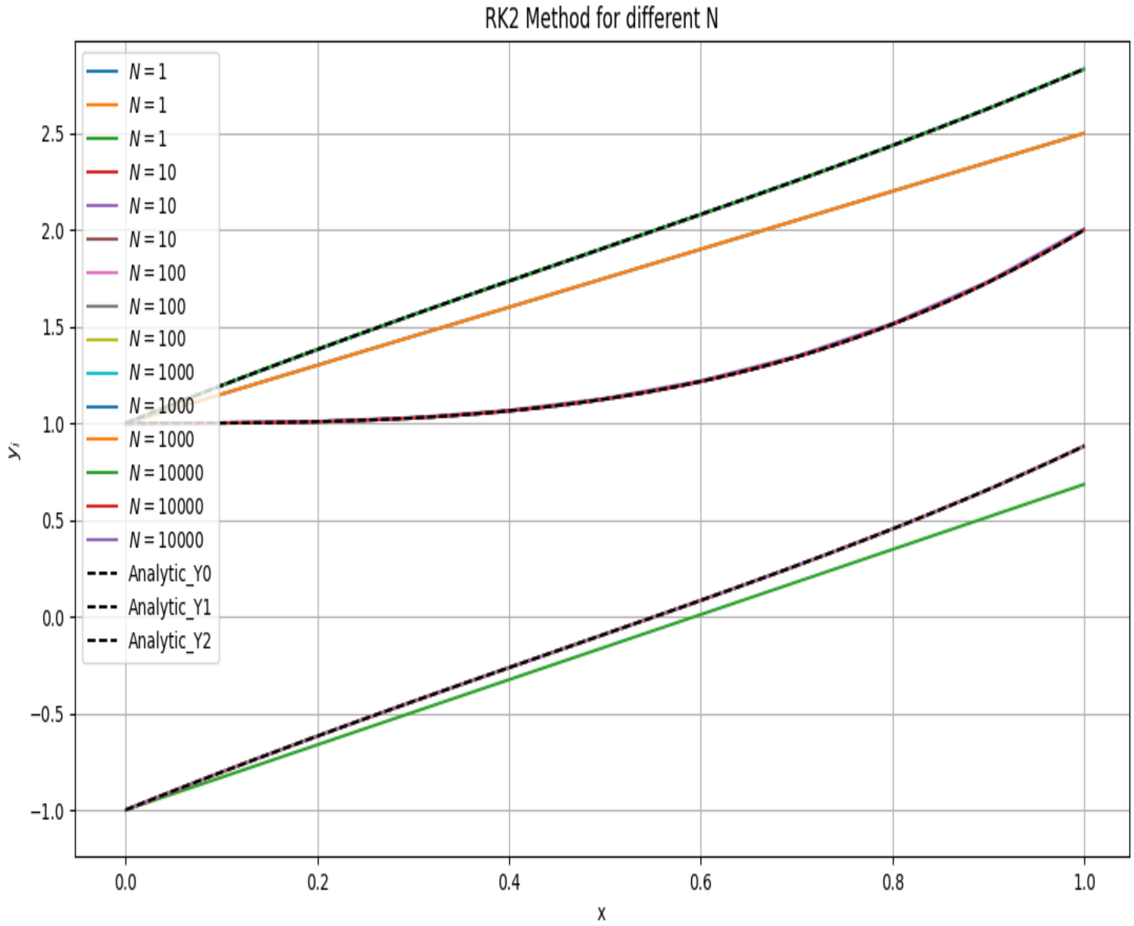

Figure 5: *y_i vs x plot using RK2 Method for different N.*

The above plot is between y_i vs x for increasing N(Number of intervals) by RK2 method , it can be understood that as the value of N increases , the plot starts overlapping with that of analytic one. In other words, the numerical results start converging with as the value of N is increased.

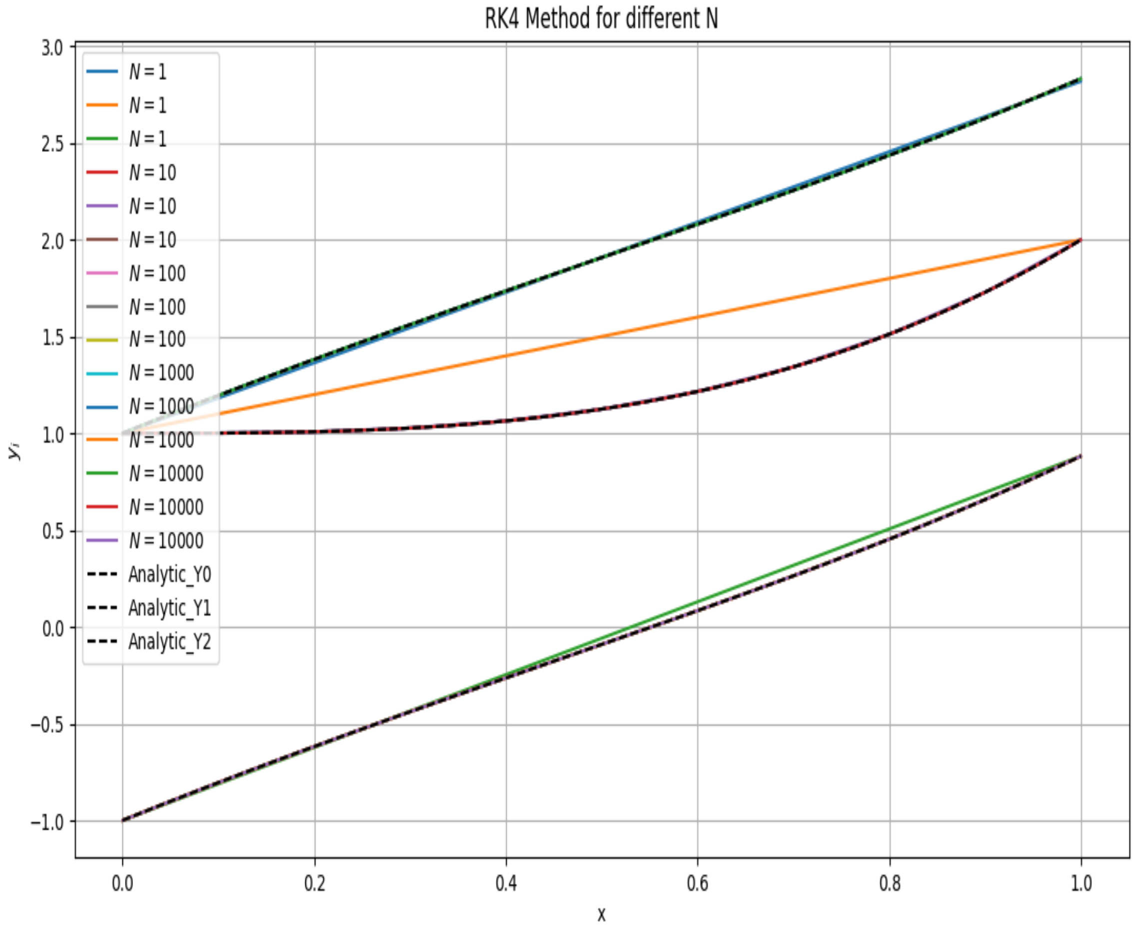

Figure 6: *y_i vs x plot using RK4 Method for different N.*

The above plot is between y_i vs x for increasing N(Number of intervals) by RK4 method , it can be understood that as the value of N increases , the plot starts overlapping with that of analytic one. In other words, the numerical results start converging with as the value of N is increased.

Table for N and E= max(|y_ana -y_num|) values for y0,y1 and y2 for all three methods

N	E_y1(Euler)	E_y1(RK2)	E_y1(RK4)	E_y2(Euler)	E_y2(RK2)	E_y2(RK4)	E_y3(Euler)	E_y3(RK2)	E_y3(RK4)
1.0	0.0	0.33	0.013	1.0	0.0	0.0	0.0	0.2	0.0
1e+01	0.044	0.0017	1.4e-06	0.14	0.0	6.7e-16	0.056	0.002	0.0
1e+02	0.0049	1.5e-05	1.4e-10	0.015	0.0	4.4e-16	0.0067	2e-05	0.0
1e+03	0.00049	1.5e-07	1.6e-14	0.0015	0.0	2.2e-16	0.00068	2e-07	6.7e-16
1e+04	4.9e-05	1.5e-09	1.2e-14	0.00015	0.0	4e-15	6.8e-05	2e-09	1e-15
1e+05	4.9e-06	1.5e-11	2.3e-14	1.5e-05	0.0	8.9e-15	6.8e-06	2e-11	3.1e-15

Figure 7: E_{y_i} *for different N using Euler, RK2 and RK4 Method*

From the above table , it can be understood that as the value of N increases , the error starts decreasing. Also, the error inn Euler Method is more than Runge Kutta Method. Out of RK2 and RK4 Method , error in RK4 is comparatively less.

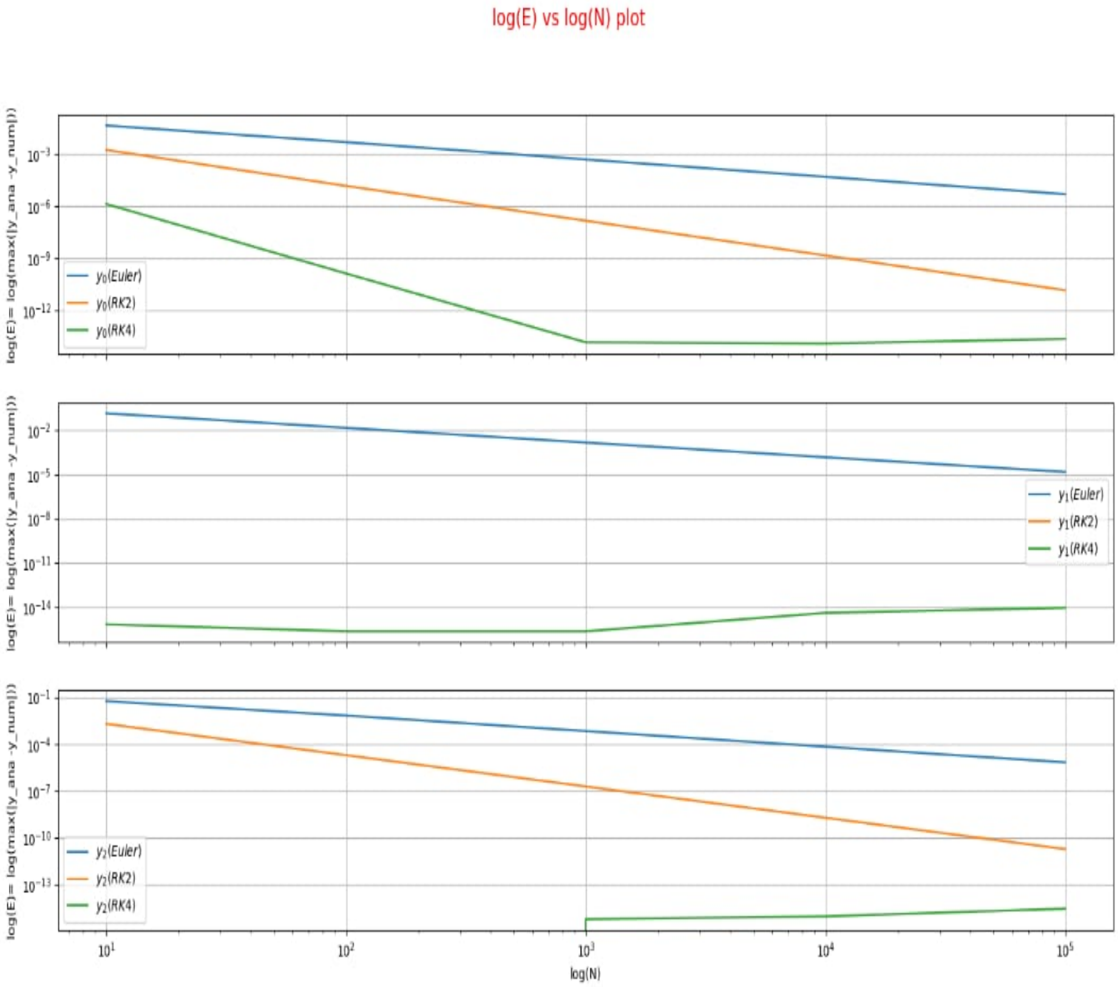

Figure 8: *log(E) vs log(N) plot for all three methods*

From the above plot , it can be understood that as the value of N increases , the error starts decreasing. Also, the error inn Euler Method is more than Runge Kutta Method. Out of RK2 and RK4 Method , error in RK4 is comparatively less because in RK4 method, slope calculation is done at 4 points whereas in RK2 , slope is calculated at two points only. Slope calculation at more points gives higher accuracy.

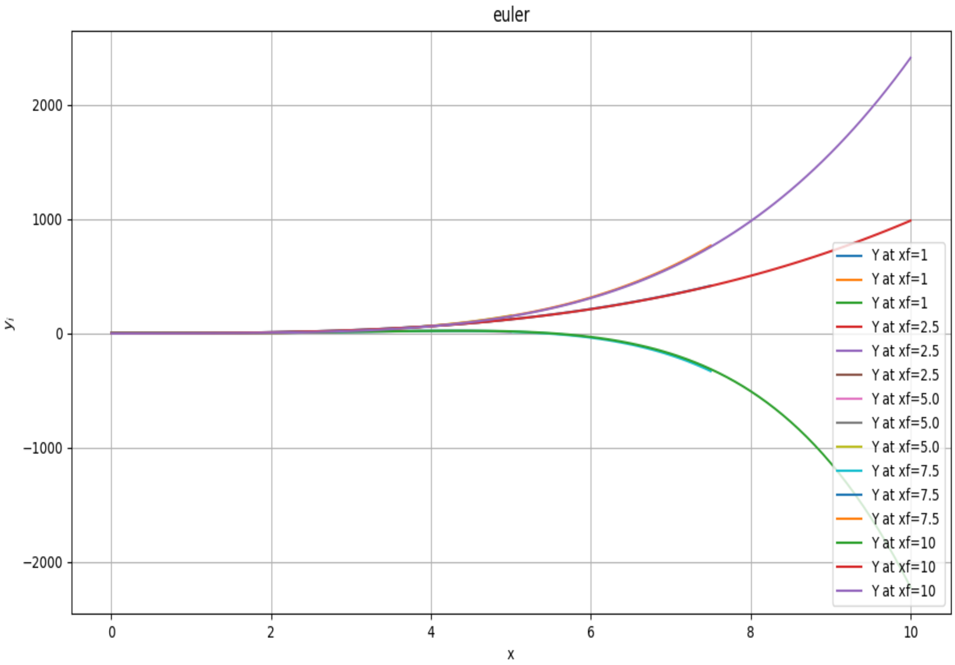

Figure 9: y_i *at different* x_f

The above plot is between y_i vs x for different x_f by euler method.

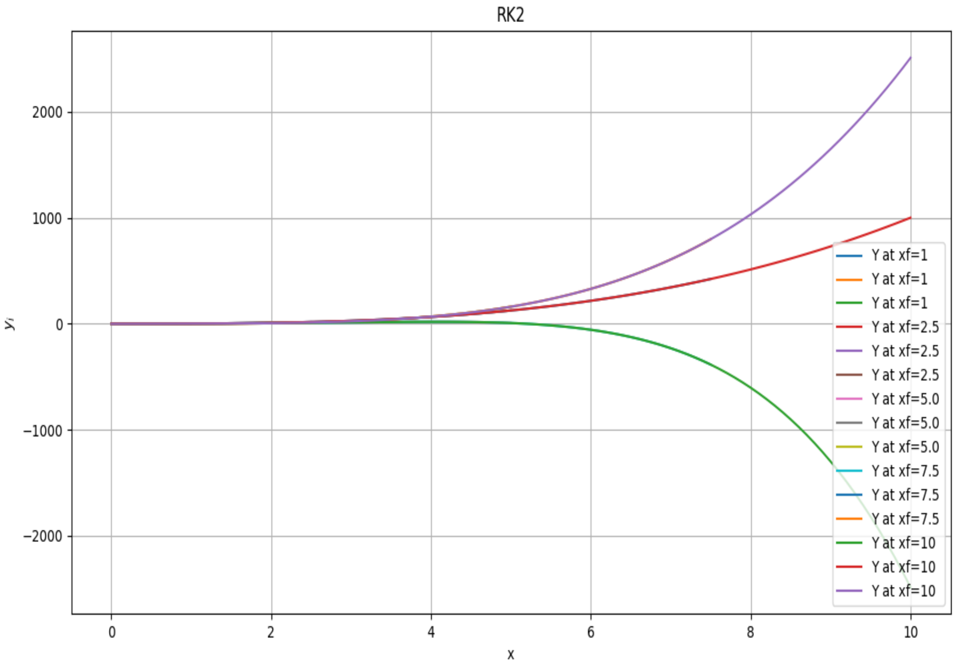

Figure 10: y_i *at different* x_f

The above plot is between y_i vs x for different x_f by RK2 method. The above plot is between y_i vs x for different x_f by RK4 method.

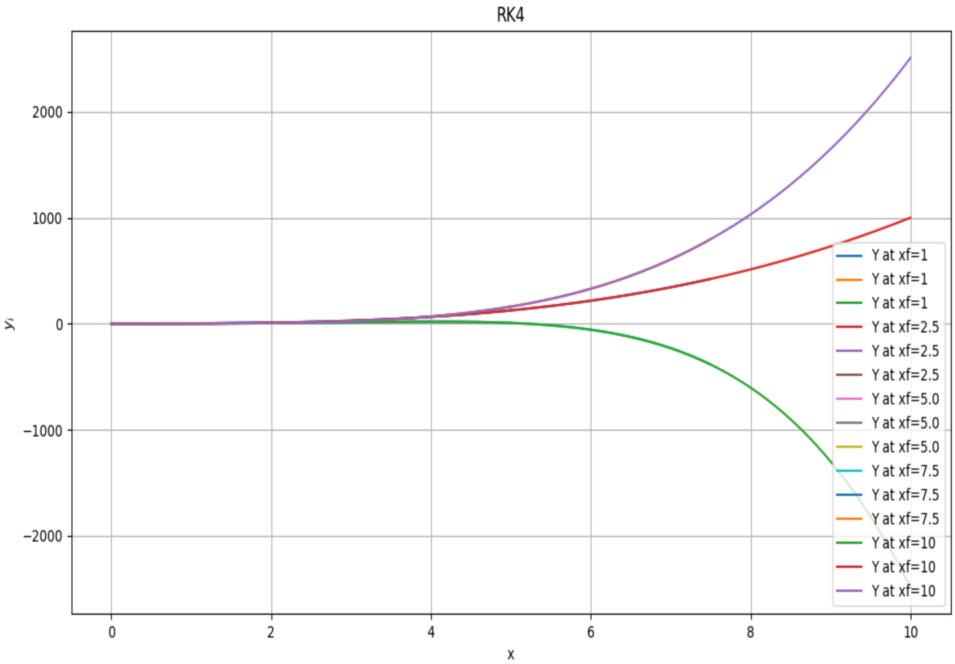

Figure 11: *y_i at different x_f*

TABLE for output for q3(b)

x	y	y'
0.0	-0.4	-0.6
0.2	-0.53	-0.64
0.4	-0.66	-0.54
0.6	-0.74	-0.16
0.8	-0.71	0.74
1.0	-0.43	2.5

Figure 12: *y and y' values for N=5.*

y(1)=-0.4375 (both from code and analytically)

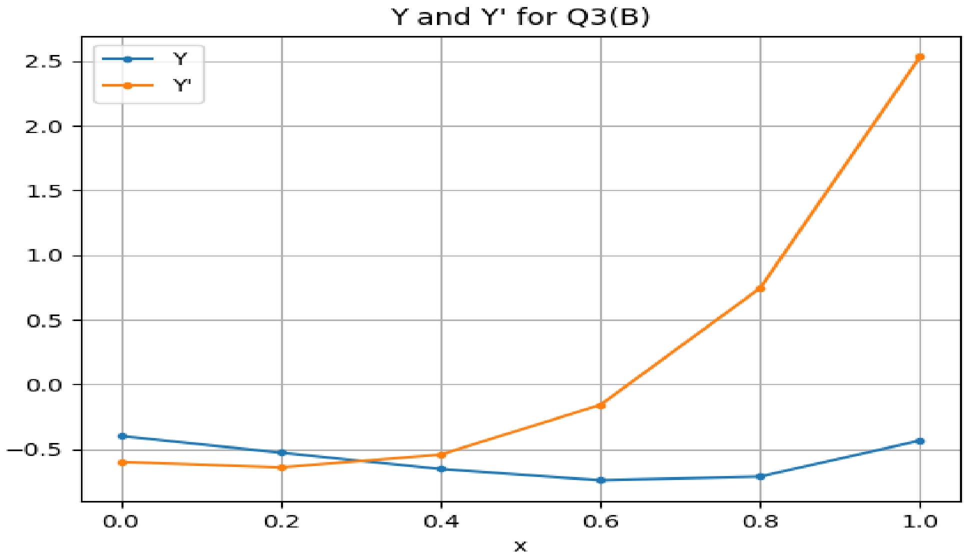

Figure 13: *y and y′ values vs x plot*

PSEUDOCODE

0.1 For Euler

Euler function takes the parameters :
Func: array of functions
IC: array of initial conditions
a: starting point
b: ending point
N: no.of steps between the starting and ending point

Firts it calculate the step size using a,b and N i.e, h=b-a/N

Creating a array of N+1 points from a to b and creating a matrix ((N+1) x (No.of initial conditions given)) which contains initial conditions in first row and rest are zeroes.

By using for loop we calculate the next term by just adding the initial term with h times the value of the function.

Euler Returns the value of functions as a matrix and points as a array at which functions are calculated .

0.2 For RK2

Rk2 function takes the parameters :
Func: array of functions

IC: array of initial conditions
a: starting point
b: ending point
N: no.of steps between the starting and ending point

Firts it calculate the step size using a,b and N i.e, h=b-a/N

Creating a array of N+1 points from a to b and creating a matrix ((N+1) x (No.of initial conditions given)) which contains initial conditions in first row and rest are zeroes.

By using for loop we calculate the next term by Calculating the slope k1 and using k1 evaluate k2 slope and then just adding the previous term with algebric mean of k1 and k2.

Rk2 Returns the value of functions as a matrix and points as a array at which functions are calculated .

0.3 For RK4

Rk4 function takes the parameters :
Func: array of functions
IC: array of initial conditions
a: starting point
b: ending point
N: no.of steps between the starting and ending point

Firts it calculate the step size using a,b and N i.e, h=b-a/N

Creating a array of N+1 points from a to b and creating a matrix ((N+1) x (No.of initial conditions given)) which contains initial conditions in first row and rest are zeroes.

By using for loop we calculate the next term by Calculating the slope k1 and using k1 evaluate k2 slope , using k2 to evaluate k3 , using k3 to find k4 and then just adding the previous term with one sixth of (k1 +2k2 +2k3 +k4)

Rk4 Returns the value of functions as a matrix and points as a array at which functions are calculated .

1 Program

```python
import numpy as np

def euler(Func,IC,a,b,N):
    h=((b-a)/N)
    t=np.linspace(a,b,N+1)
    X=np.zeros([N+1,len(IC)])
    X[0]=IC
    for i in range(N):
        X[i+1] = X[i] + h*Func(t[i],X[i])
    return X,t

def RK2(Func,IC,a,b,N):
    h=((b-a)/N)
    t = np.linspace(a,b,N+1)
    X  = np.zeros([N+1, len(IC)])
    X[0] =IC
    for i in range(N):
        k1 =h* Func(t[i],X[i])
        k2 = h*Func( t[i] + h,X[i] +  k1)
        X[i+1] = X[i] +  (k1 +k2 )/2
    return X,t

def RK2_M(Func,IC,a,b,N):
    h=((b-a)/N)
    t = np.linspace(a,b,N+1)
    X  = np.zeros([N+1, len(IC)])
    X[0] =IC
    for i in range(N):
        k1 =h* Func(t[i],X[i])
        k2 = h*Func( t[i] + h/2,X[i] +  k1/2)
        X[i+1] = X[i] +  (k1 +k2 )/2
    return X,t

def RK4(Func,IC,a,b,N):
    h=((b-a)/N)
    t = np.linspace(a,b,N+1)
    X  = np.zeros([N+1, len(IC)])
    X[0] =IC
    for i in range(N):
        k1 =h* Func(t[i],X[i])
        k2 =h*Func(t[i] + h/2,X[i] +  k1/2)
        k3 =h*Func(t[i]+h/2,X[i] + k2/2)
        k4 =h*Func(t[i]+h,X[i]+k3)
        X[i+1] = X[i] +  (k1+2*k2+2*k3+k4)/6
    return X,t
```

```python
import numpy as np
import matplotlib.pyplot as plt
from MyIVP import euler
from MyIVP import RK2
from MyIVP import RK4

def Func(x,Var):
    y1,y2,y3=Var
    f1=y2-y3+x
    f2=3*x**2
    f3=y2 + np.exp(-x)
    return np.array([f1,f2,f3])

def output(Func,IC,a,b,N):
    X=euler(Func,IC,a,b,N)
    l1=X[0].T
    X1=RK2(Func,IC,a,b,N)
    l2=X1[0].T
    X3=RK4(Func,IC,a,b,N)
    l3=X3[0].T
    t=X[1]

    return l1,l2,l3,t

def table(data,head,Title):
    print(Title)
    line='_'*len(head)*12+'____'
    for i in head:
        print("{0:^12}".format(i),end=" ")
    print("\n",line)
    for row in data:
        for val in row:
            print("{0:^12.2}".format(val), end=" ")
        print("\n")

IC=[1,1,-1]
a=0
b=1
N=100

Ana_Sol_1= lambda x : -0.05*x**5 + 0.25*x**4 + x +2 - np.exp(-x)
Ana_Sol_2= lambda x :  x**3 +1
Ana_Sol_3= lambda x :  0.25*x**4 + x - np.exp(-x)

X,Y,Z,t=output(Func,IC,a,b,N)

EUL_data=np.column_stack((t,X[0],X[1],X[2]))
RK2_data=np.column_stack((t,Y[0],Y[1],Y[2]))
RK4_data=np.column_stack((t,Z[0],Z[1],Z[2]))
heading=["x","Y1","Y2","Y3"]

table(EUL_data,heading,"\nEULER METHOD FOR N=100")
table(RK2_data,heading,"\nRK2 METHOD FOR N=100")
table(RK4_data,heading,"\nRK4 METHOD FOR N=100")

```

```python
def graph(t,X,title):
    for i in range(len(X)):
        plt.plot(t,X[i],label="Y"+str(i))
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("$y_i$")
    plt.legend()
    plt.grid()

def graph_ana(t,Analytic):
    for g,i in zip(Analytic,range(len(Analytic))):
        plt.plot(t,g(t),label="Analytic_Y"+str(i),ls="--",c="black")
    plt.legend()

Analytic=[Ana_Sol_1,Ana_Sol_2,Ana_Sol_3]

graph(t,X,"\nEULER METHOD FOR N=100")
graph_ana(t,Analytic)
plt.show()

graph(t,Y,"\nRK2 METHOD FOR N=100")
graph_ana(t,Analytic)
plt.show()

graph(t,Z,"\nRK4 METHOD FOR N=100")
graph_ana(t,Analytic)
plt.show()

N_arr=[]
E_err_1=[];E_err_2=[];E_err_3=[]
RK2_err_1=[];RK2_err_2=[];RK2_err_3=[]
RK4_err_1=[];RK4_err_2=[];RK4_err_3=[]

def plot3(X,t,N):
    for i in range(len(X)):
        plt.plot(t,X[i],label="$N={0}$".format(N))

for i in range(0,6,1):
    N=10**(i)
    N_arr.append(N)
    X,Y,Z,t=output(Func,IC,a,b,N)
    plot3(X,t,N)
    f1=Ana_Sol_1(t)
    f2=Ana_Sol_2(t)
    f3=Ana_Sol_3(t)
    E_err_1.append(max(np.array(f1)-np.array(X[0])))
    E_err_2.append(max(np.array(f2)-np.array(X[1])))
    E_err_3.append(max(np.array(f3)-np.array(X[2])))

graph_ana(t,Analytic)
plt.title("Euler Method for different N")
plt.xlabel("x")
plt.ylabel("$y_i$")
plt.legend()
plt.grid()
```

```
plt.show()

for i in range(0,6,1):
    N=10**(i)
    X,Y,Z,t=output(Func,IC,a,b,N)
    plot3(Y,t,N)
    f1=Ana_Sol_1(t)
    f2=Ana_Sol_2(t)
    f3=Ana_Sol_3(t)
    RK2_err_1.append(max(np.array(f1)-np.array(Y[0])))
    RK2_err_2.append(max(np.array(f2)-np.array(Y[1])))
    RK2_err_3.append(max(np.array(f3)-np.array(Y[2])))

graph_ana(t,Analytic)
plt.title("RK2 Method for different N")
plt.xlabel("x")
plt.ylabel("$y_i$")
plt.legend()
plt.grid()
plt.show()

for i in range(0,6,1):
    N=10**(i)
    X,Y,Z,t=output(Func,IC,a,b,N)
    plot3(Z,t,N)
    f1=Ana_Sol_1(t)
    f2=Ana_Sol_2(t)
    f3=Ana_Sol_3(t)
    RK4_err_1.append(max(np.array(f1)-np.array(Z[0])))
    RK4_err_2.append(max(np.array(f2)-np.array(Z[1])))
    RK4_err_3.append(max(np.array(f3)-np.array(Z[2])))

graph_ana(t,Analytic)
plt.title("RK4 Method for different N")
plt.xlabel("x")
plt.ylabel("$y_i$")
plt.legend()
plt.grid()
plt.show()

print("\n")
data=np.column_stack((N_arr,E_err_1,RK2_err_1,RK4_err_1,E_err_2,
    RK2_err_2,RK4_err_2,E_err_3,RK2_err_3,RK4_err_3))
head=["N","E_y1(Euler)","E_y1(RK2)","E_y1(RK4)","E_y2(Euler)","E_y2(RK2
    )","E_y2(RK4)","E_y3(Euler)","E_y3(RK2)","E_y3(RK4)"]
table(data,head,"\nTable for N and E= max(|y_ana -y_num|) values for y0
    ,y1 and y2 for all three methods  ")

fig, (ax1,ax2,ax3) = plt.subplots(3,sharex=True)
fig.suptitle("log(E) vs log(N) plot",fontsize=15,c="r")
ax1.plot(N_arr,E_err_1,label="$y_{0}(Euler)$")
ax1.plot(N_arr,RK2_err_1,label="$y_{0}(RK2)$")
ax1.plot(N_arr,RK4_err_1,label="$y_{0}(RK4)$")
ax1.set(ylabel="log(E)= log(max(|y_ana -y_num|))",yscale="log",xscale="
    log")
```

```
ax1.grid()
ax1.legend()

ax2.plot(N_arr,E_err_2,label="$y_{1}(Euler)$")
ax2.plot(N_arr,RK2_err_2,label="$y_{1}(RK2)$")
ax2.plot(N_arr,RK4_err_2,label="$y_{1}(RK4)$")
ax2.set(ylabel="log(E)= log(max(|y_ana -y_num|))",xscale="log",yscale="log")
ax2.grid()
ax2.legend()

ax3.plot(N_arr,E_err_3,label="$y_{2}(Euler)$")
ax3.plot(N_arr,RK2_err_3,label="$y_{2}(RK2)$")
ax3.plot(N_arr,RK4_err_3,label="$y_{2}(RK4)$")
ax3.set(xlabel="log(N)",ylabel="log(E)= log(max(|y_ana -y_num|))",xscale="log",yscale="log")
ax3.grid()
ax3.legend()
plt.show()

x_f=[1,2.5,5.0,7.5,10]
for i in x_f:
    l1,l2,l3,t=output(Func,IC,0,i,100)
    for k in range(len(l1)):
        plt.plot(t,l1[k],label="Y at xf="+str(i))
    plt.title("euler")
    plt.xlabel("x")
    plt.ylabel("$y_i$")
    plt.legend()
    plt.grid()
plt.show()

for i in x_f:
    l1,l2,l3,t=output(Func,IC,0,i,100)
    for k in range(len(l1)):
        plt.plot(t,l2[k],label="Y at xf="+str(i))
    plt.title("RK2")
    plt.xlabel("x")
    plt.ylabel("$y_i$")
    plt.legend()
    plt.grid()
plt.show()

for i in x_f:
    l1,l2,l3,t=output(Func,IC,0,i,100)
    for k in range(len(l1)):
        plt.plot(t,l3[k],label="Y at xf="+str(i))
    plt.title("RK4")
    plt.xlabel("x")
    plt.ylabel("$y_i$")
    plt.legend()
    plt.grid()
plt.show()

def f3_b(x,var):
```

```
    y,u=var
    f1=u
    f2=np.exp(2*x)*np.sin(x) -2*y +2*u
    return np.array([f1,f2])

ic=[-0.4,-0.6]
z=RK2(f3_b,ic,0,1,5)
y=z[0].T

head=["x","y","y'"]
data_3b=np.column_stack((z[1],y[0],y[1]))
table(data_3b,head,"TABLE for output for q3(b)")

plt.plot(z[1],y[0],label="Y",marker=".")
plt.plot(z[1],y[1],label="Y'",marker=".")
plt.xlabel("x")
plt.title("Y and Y' for Q3(B)")
plt.legend()
plt.grid()
plt.show()
```

```python
from MyIVP import euler
from MyIVP import RK2
from MyIVP import RK4
import numpy as np


def euler_tol(Func, IC ,a,b,N,N_max,tol):
    w=0
    Val=[]
    N_arr=[]
    count=0
    while N<=N_max:
        g=euler(Func, IC ,a,b,N)[0].T
        t=euler(Func, IC ,a,b,N)[1]
        Val.append(g)
        N_arr.append(N)
        if count>=1:
            J=[]
            K=[]
            for i in range(len(IC)):
                J.append(Val[-1][i][-1])
                K.append(Val[-2][i][-1])
            J=np.array(J)
            K=np.array(K)
            ff=[]
            for g1,g2 in zip(J,K):
                if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
                    err=abs(g1-g2)
                else:
                    err=abs((g2-g1)/g1)
                ff.append(err)
            if max(ff)<=tol:
                w=1
                break
            else:
                pass

        N=2*N
        count+=1
    if w==0:
        s=("N_max reached without achieving required tolerance")
    elif w==1:
      s="Given tolerance achieved with",N_arr[-1],"sub-intervals"

    return Val,Val[-1],N_arr[-1],N_arr,s,g,t

def RK2_tol(Func, IC ,a,b,N,N_max,tol):
    w=0
    Val=[]
    N_arr=[]
    count=0
    while N<=N_max:
        g=RK2(Func, IC ,a,b,N)[0].T
        t=RK2(Func, IC ,a,b,N)[1]
        Val.append(g)
```

```
        N_arr.append(N)
        if count>=1:
            J=[]
            K=[]
            for i in range(len(IC)):
                J.append(Val[-1][i][-1])
                K.append(Val[-2][i][-1])
            J=np.array(J)
            K=np.array(K)
            ff=[]
            for g1,g2 in zip(J,K):
                if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
                    err=abs(g1-g2)
                else:
                    err=abs((g2-g1)/g1)
                ff.append(err)
            if max(ff)<=tol:
                w=1
                break
            else:
                pass

        N=2*N
        count+=1
    if w==0:
        s=("N_max reached without achieving required tolerance")
    elif w==1:
      s="Given tolerance achieved with",N_arr[-1],"sub-intervals"

    return Val,Val[-1],N_arr[-1],N_arr,s,g,t

def RK4_tol(Func, IC ,a,b,N,N_max,tol):
    w=0
    Val=[]
    N_arr=[]
    count=0
    while N<=N_max:
        g=RK4(Func, IC ,a,b,N)[0].T
        t=RK4(Func, IC ,a,b,N)[1]
        Val.append(g)
        N_arr.append(N)
        if count>=1:
            J=[]
            K=[]
            for i in range(len(IC)):
                J.append(Val[-1][i][-1])
                K.append(Val[-2][i][-1])
            J=np.array(J)
            K=np.array(K)
            ff=[]
            for g1,g2 in zip(J,K):
                if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
                    err=abs(g1-g2)
                else:
                    err=abs((g2-g1)/g1)
```

```
                ff.append(err)
            if max(ff)<=tol:
                w=1
                break
            else:
                pass

        N=2*N
        count+=1
    if w==0:
        s=("N_max reached without achieving required tolerance")
    elif w==1:
      s="Given tolerance achieved with",N_arr[-1],"sub-intervals"

    return Val,Val[-1],N_arr[-1],N_arr,s,g,t
```