

RESULTS AND DISCUSSION

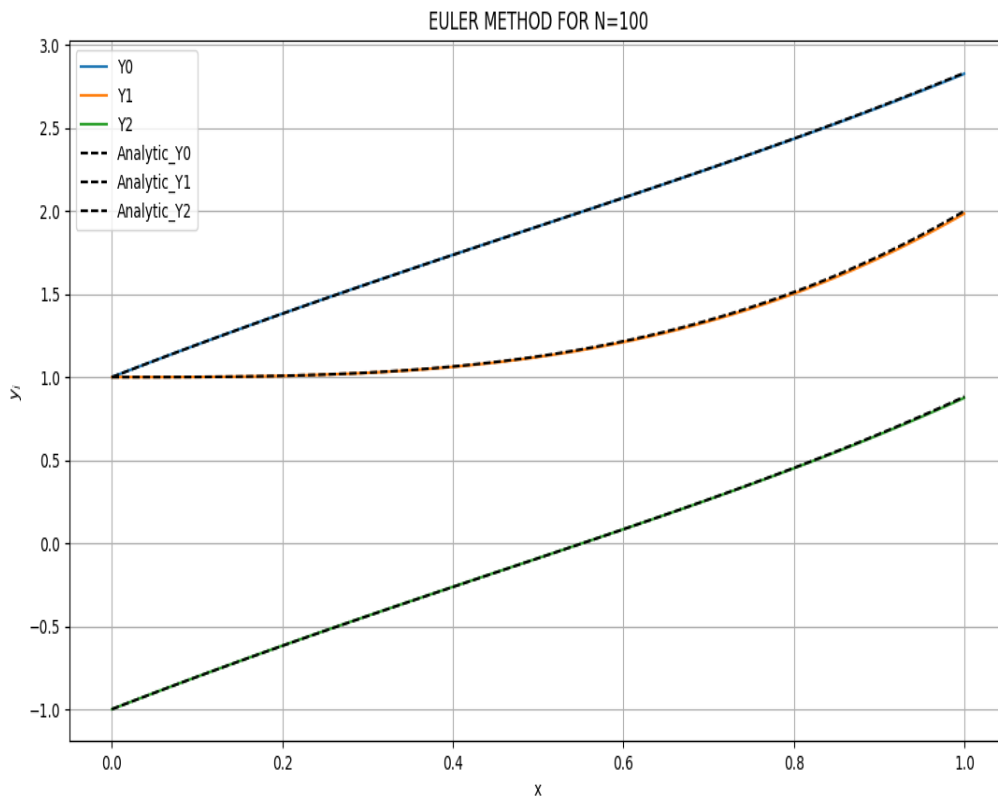


Figure 1: y_i vs x plot using Euler Method.

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

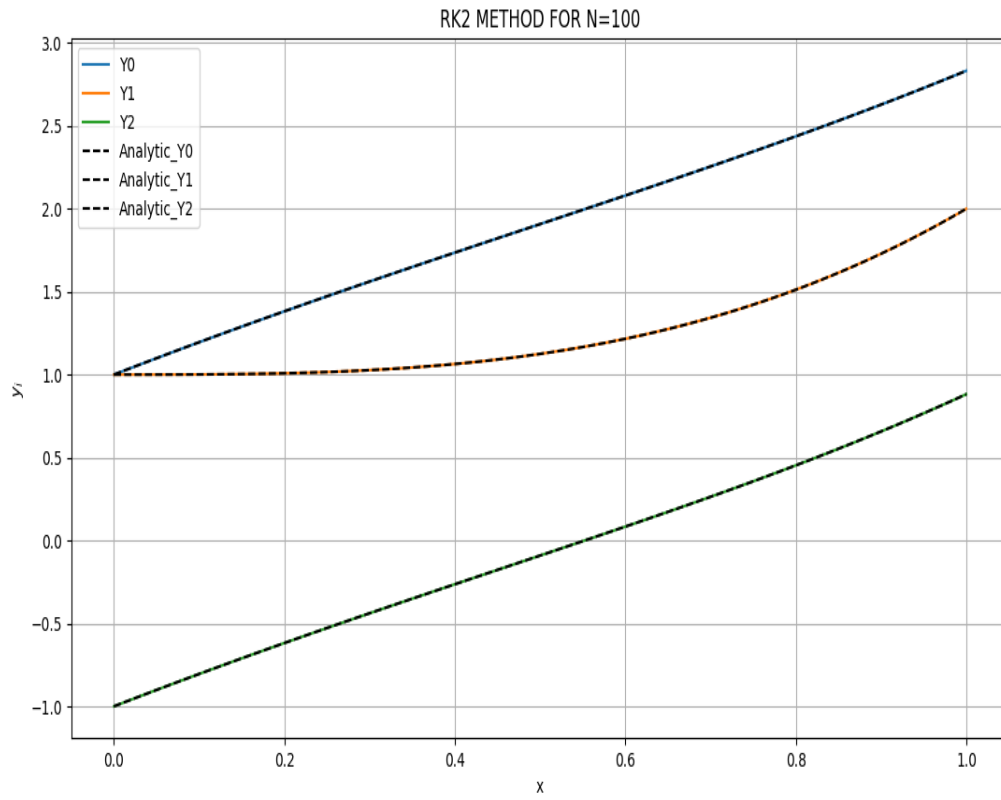


Figure 2: y_i vs x plot using RK2 Method.

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

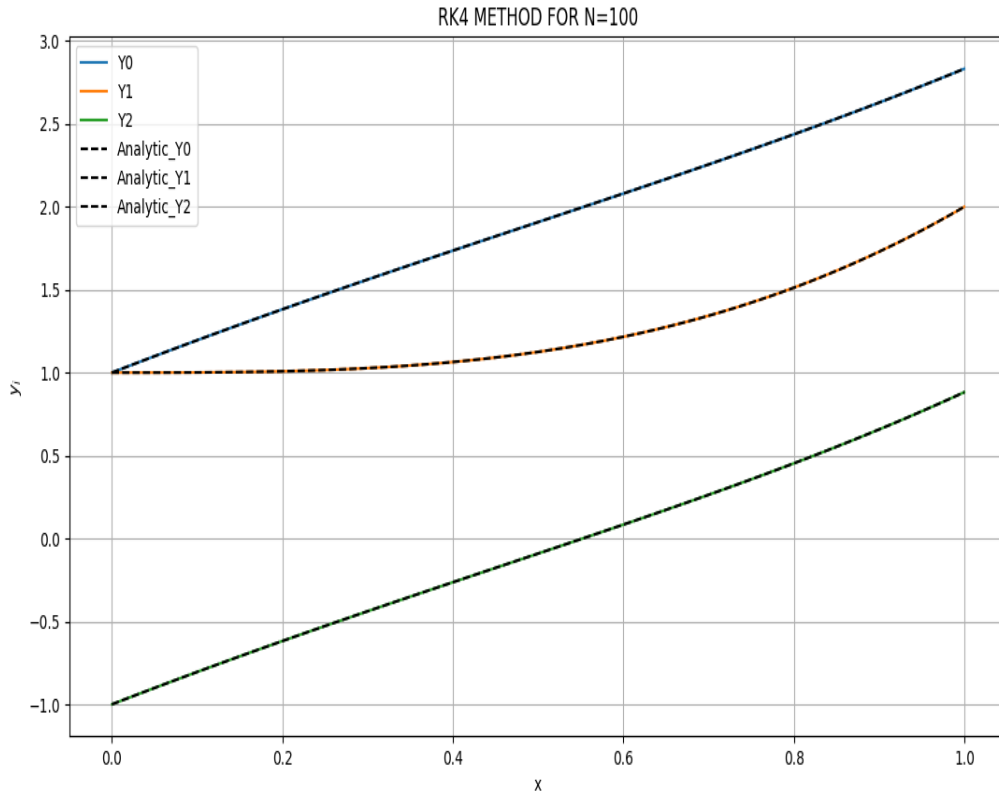


Figure 3: y_i vs x plot using *RK4 Method*.

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

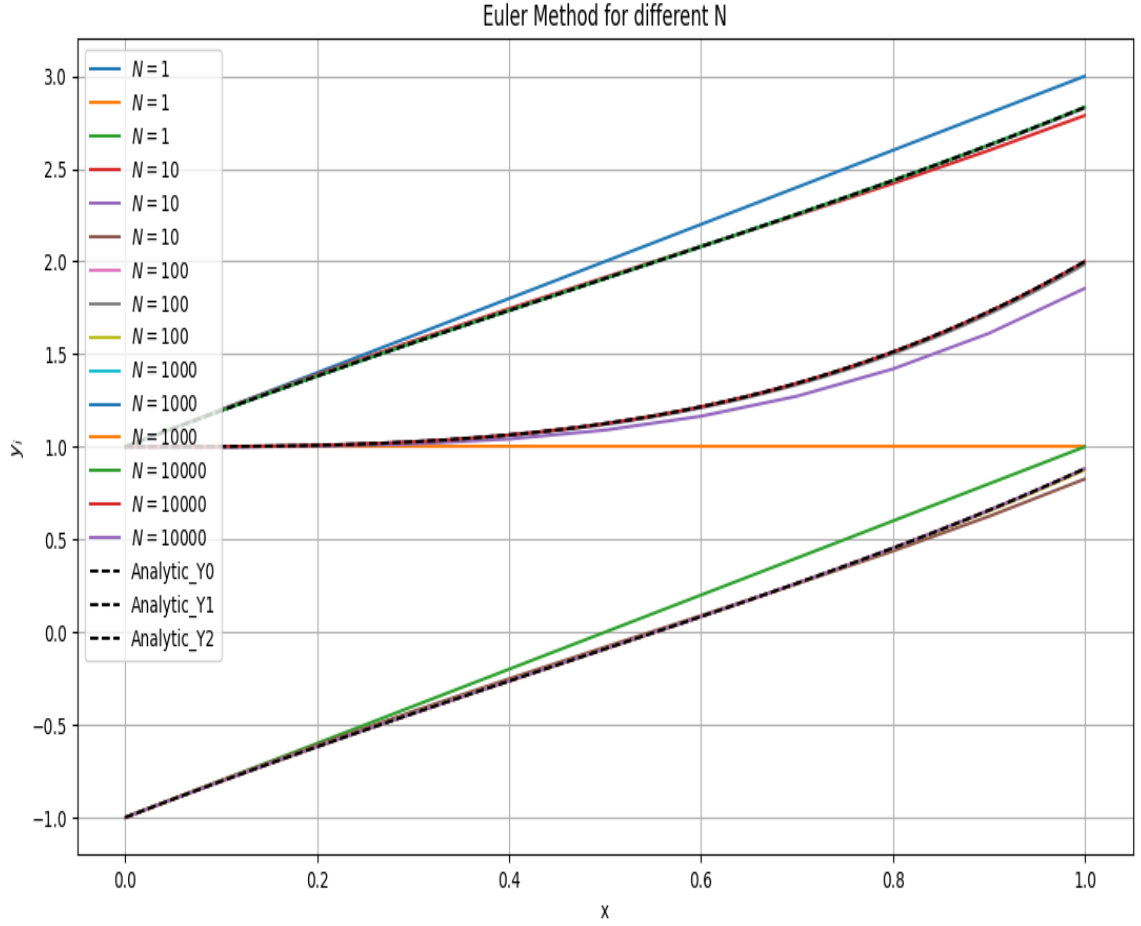


Figure 4: y_i vs x plot using Euler Method for different N .

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.

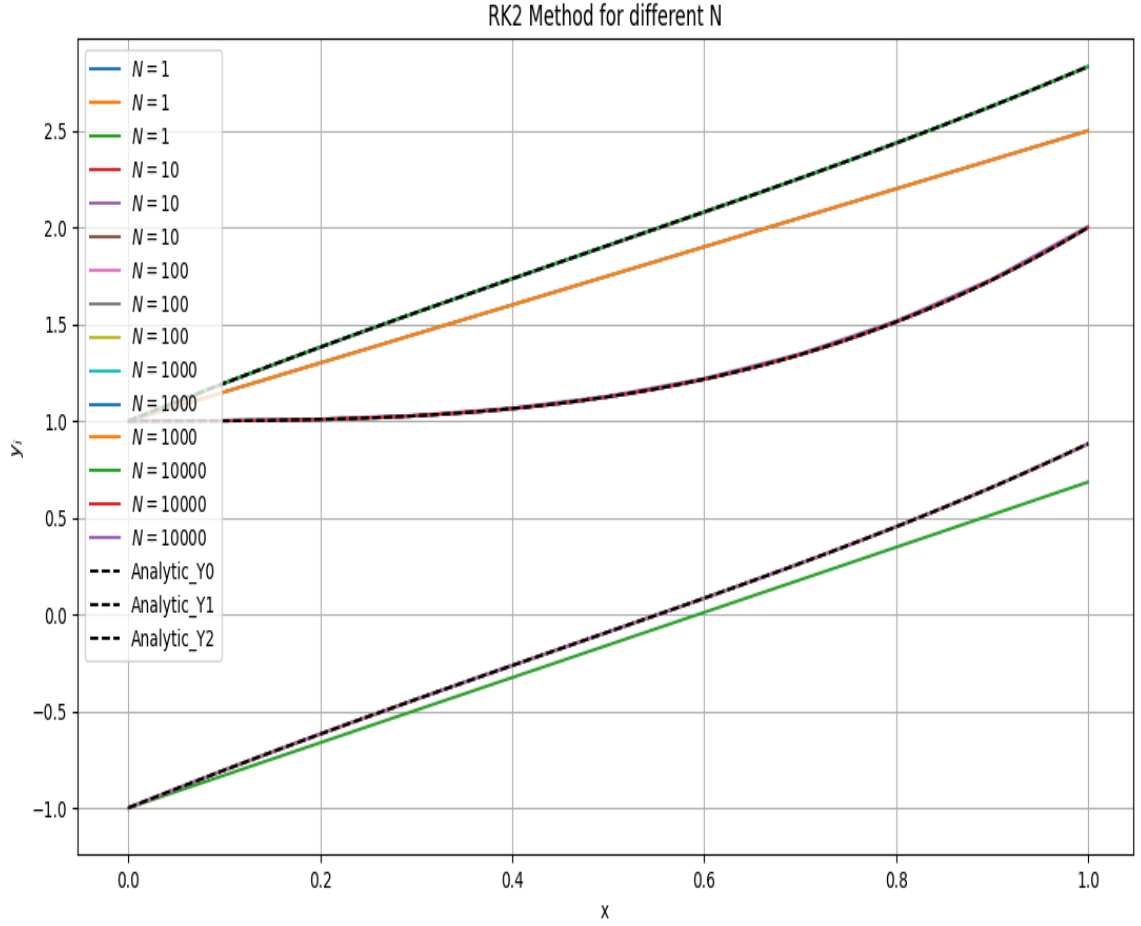


Figure 5: y_i vs x plot using RK2 Method for different N .

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.

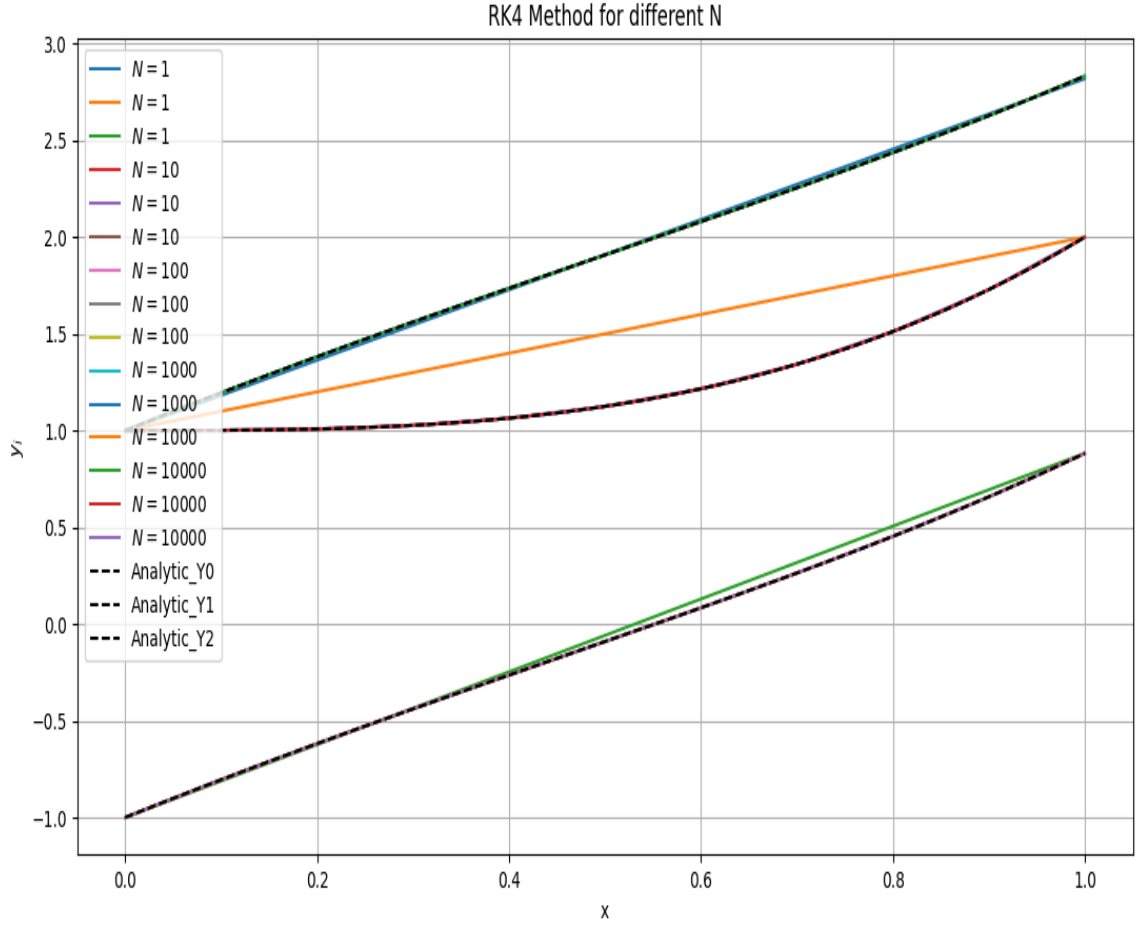


Figure 6: y_i vs x plot using *RK4 Method* for different N .

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.

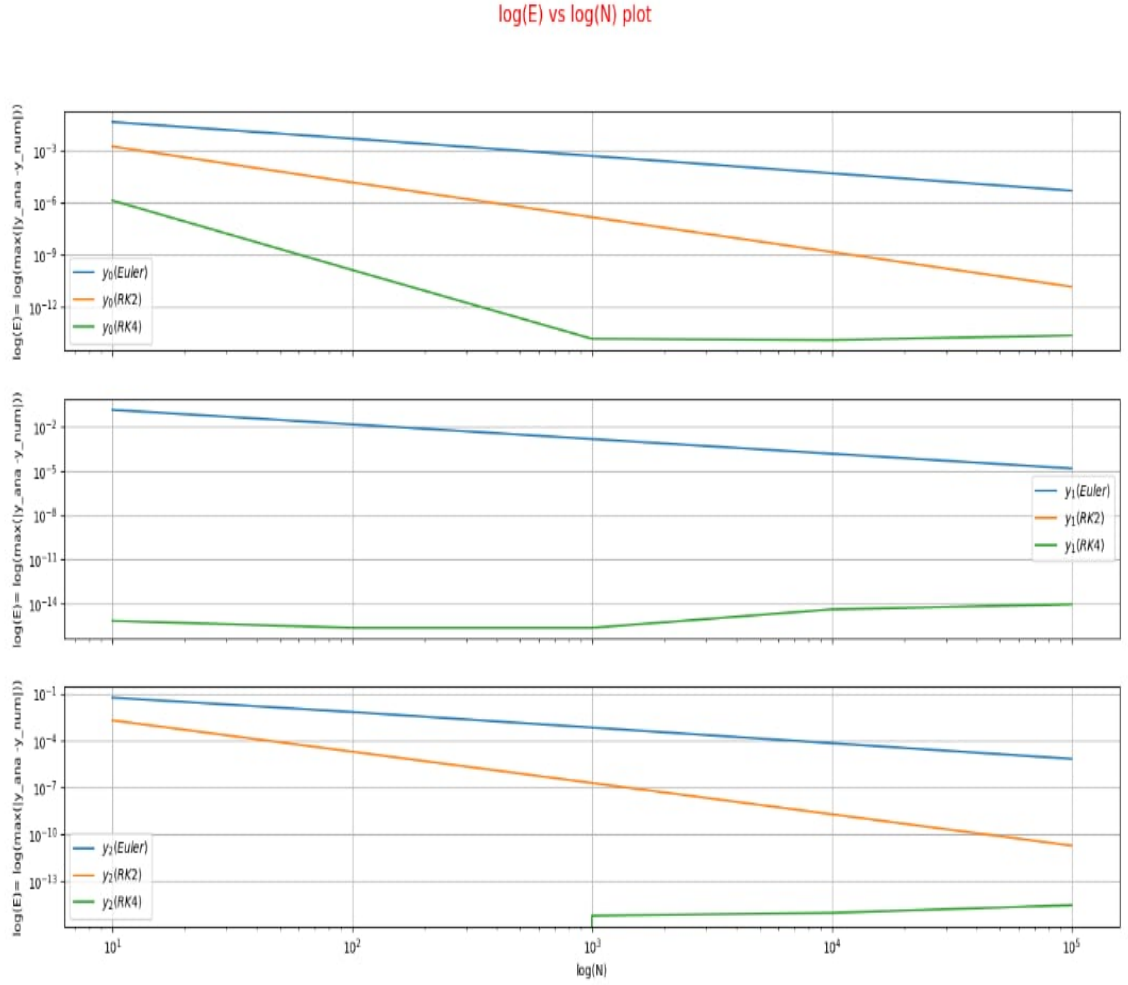


Figure 8: $\log(E)$ vs $\log(N)$ plot for all three methods

From the above plot, it can be understood that as the value of N increases, the error starts decreasing. Also, the error in 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.

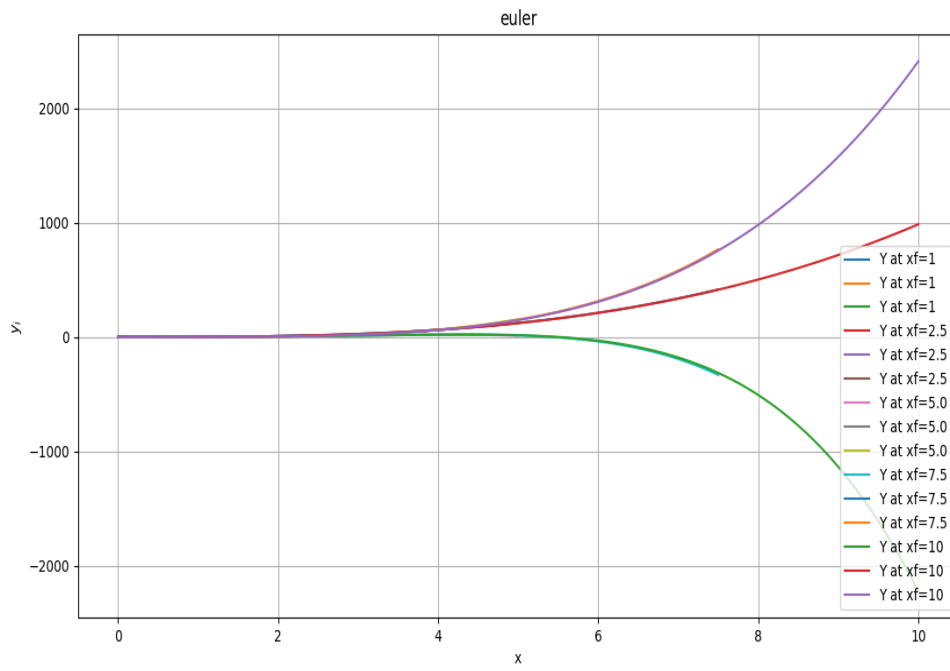


Figure 9: y_i at different x_f

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

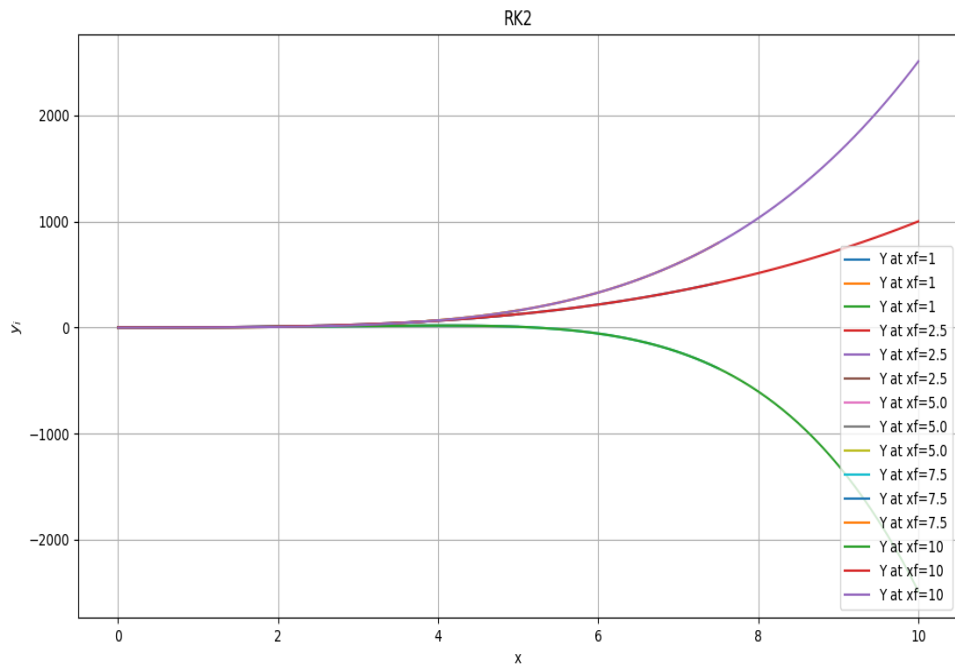


Figure 10: y_i at different x_f

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.

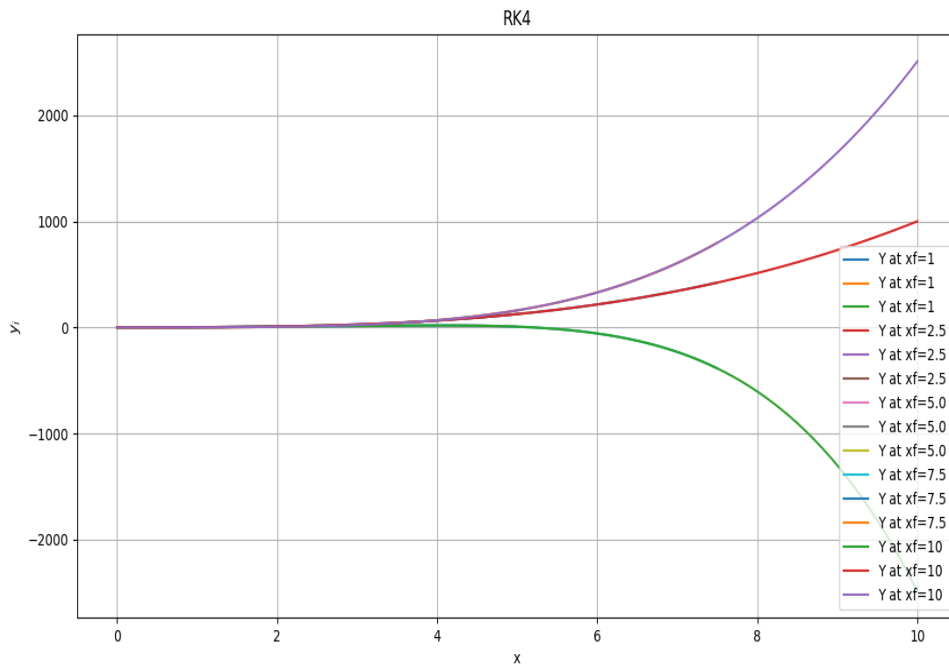


Figure 11: y_i at different x_f

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)

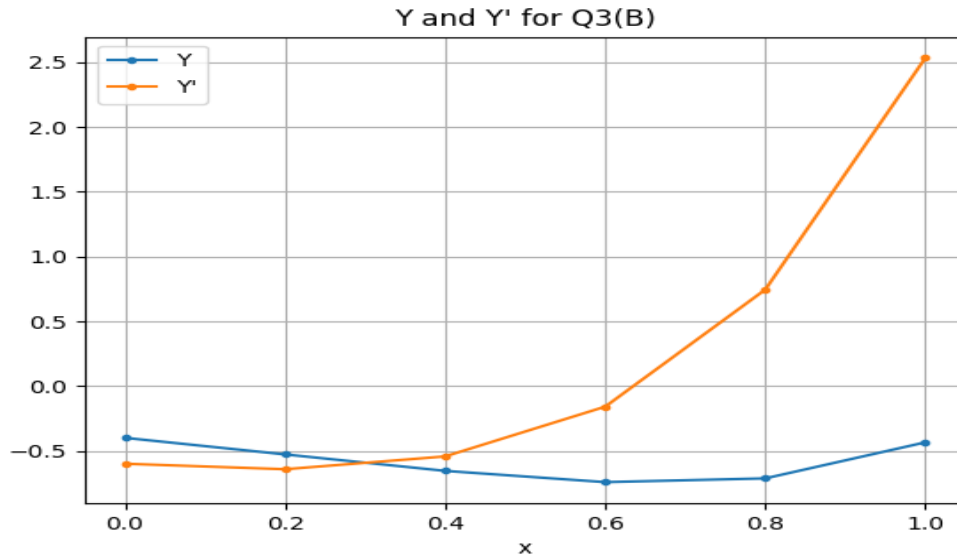


Figure 13: y and y' values vs x plot

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

First 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 $((N+1) \times (\text{No.of initial conditions given}))$ 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 $((N+1) \times (\text{No.of initial conditions given}))$ 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 $((N+1) \times (\text{No.of initial conditions given}))$ 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

```
1 import numpy as np
2
3 def euler(Func, IC, a, b, N):
4     h = ((b - a) / N)
5     t = np.linspace(a, b, N + 1)
6     X = np.zeros([N + 1, len(IC)])
7     X[0] = IC
8     for i in range(N):
9         X[i + 1] = X[i] + h * Func(t[i], X[i])
10    return X, t
11
12 def RK2(Func, IC, a, b, N):
13     h = ((b - a) / N)
14     t = np.linspace(a, b, N + 1)
15     X = np.zeros([N + 1, len(IC)])
16     X[0] = IC
17     for i in range(N):
18         k1 = h * Func(t[i], X[i])
19         k2 = h * Func(t[i] + h, X[i] + k1)
20         X[i + 1] = X[i] + (k1 + k2) / 2
21    return X, t
22
23 def RK2_M(Func, IC, a, b, N):
24     h = ((b - a) / N)
25     t = np.linspace(a, b, N + 1)
26     X = np.zeros([N + 1, len(IC)])
27     X[0] = IC
28     for i in range(N):
29         k1 = h * Func(t[i], X[i])
30         k2 = h * Func(t[i] + h / 2, X[i] + k1 / 2)
31         X[i + 1] = X[i] + (k1 + k2) / 2
32    return X, t
33
34 def RK4(Func, IC, a, b, N):
35     h = ((b - a) / N)
36     t = np.linspace(a, b, N + 1)
37     X = np.zeros([N + 1, len(IC)])
38     X[0] = IC
39     for i in range(N):
40         k1 = h * Func(t[i], X[i])
41         k2 = h * Func(t[i] + h / 2, X[i] + k1 / 2)
42         k3 = h * Func(t[i] + h / 2, X[i] + k2 / 2)
43         k4 = h * Func(t[i] + h, X[i] + k3)
44         X[i + 1] = X[i] + (k1 + 2 * k2 + 2 * k3 + k4) / 6
45    return X, t
```

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from MyIVP import euler
4 from MyIVP import RK2
5 from MyIVP import RK4
6
7 def Func(x,Var):
8     y1,y2,y3=Var
9     f1=y2-y3+x
10    f2=3*x**2
11    f3=y2 + np.exp(-x)
12    return np.array([f1,f2,f3])
13
14 def output(Func,IC,a,b,N):
15     X=euler(Func,IC,a,b,N)
16     l1=X[0].T
17     X1=RK2(Func,IC,a,b,N)
18     l2=X1[0].T
19     X3=RK4(Func,IC,a,b,N)
20     l3=X3[0].T
21     t=X[1]
22
23     return l1,l2,l3,t
24
25 def table(data,head,Title):
26     print(Title)
27     line='_'*len(head)*12+'____'
28     for i in head:
29         print("{0:^12}".format(i),end=" ")
30     print("\n",line)
31     for row in data:
32         for val in row:
33             print("{0:^12.2}".format(val), end=" ")
34     print("\n")
35
36 IC=[1,1,-1]
37 a=0
38 b=1
39 N=100
40
41 Ana_Sol_1= lambda x : -0.05*x**5 + 0.25*x**4 + x +2 - np.exp(-x)
42 Ana_Sol_2= lambda x : x**3 +1
43 Ana_Sol_3= lambda x : 0.25*x**4 + x - np.exp(-x)
44
45 X,Y,Z,t=output(Func,IC,a,b,N)
46
47 EUL_data=np.column_stack((t,X[0],X[1],X[2]))
48 RK2_data=np.column_stack((t,Y[0],Y[1],Y[2]))
49 RK4_data=np.column_stack((t,Z[0],Z[1],Z[2]))
50 heading=["x","Y1","Y2","Y3"]
51
52 table(EUL_data,heading,"\nEULER METHOD FOR N=100")
53 table(RK2_data,heading,"\nRK2 METHOD FOR N=100")
54 table(RK4_data,heading,"\nRK4 METHOD FOR N=100")
55

```

```

56 def graph(t,X,title):
57     for i in range(len(X)):
58         plt.plot(t,X[i],label="Y"+str(i))
59     plt.title(title)
60     plt.xlabel("x")
61     plt.ylabel("$y_i$")
62     plt.legend()
63     plt.grid()
64
65 def graph_ana(t,Analytic):
66     for g,i in zip(Analytic,range(len(Analytic))):
67         plt.plot(t,g(t),label="Analytic_Y"+str(i),ls="--",c="black")
68     plt.legend()
69
70 Analytic=[Ana_Sol_1,Ana_Sol_2,Ana_Sol_3]
71
72 graph(t,X,"\nEULER METHOD FOR N=100")
73 graph_ana(t,Analytic)
74 plt.show()
75
76 graph(t,Y,"\nRK2 METHOD FOR N=100")
77 graph_ana(t,Analytic)
78 plt.show()
79
80 graph(t,Z,"\nRK4 METHOD FOR N=100")
81 graph_ana(t,Analytic)
82 plt.show()
83
84 N_arr=[]
85 E_err_1=[];E_err_2=[];E_err_3=[]
86 RK2_err_1=[];RK2_err_2=[];RK2_err_3=[]
87 RK4_err_1=[];RK4_err_2=[];RK4_err_3=[]
88
89 def plot3(X,t,N):
90     for i in range(len(X)):
91         plt.plot(t,X[i],label="$N={0}$".format(N))
92
93 for i in range(0,6,1):
94     N=10**(i)
95     N_arr.append(N)
96     X,Y,Z,t=output(Func,IC,a,b,N)
97     plot3(X,t,N)
98     f1=Ana_Sol_1(t)
99     f2=Ana_Sol_2(t)
100    f3=Ana_Sol_3(t)
101    E_err_1.append(max(np.array(f1)-np.array(X[0])))
102    E_err_2.append(max(np.array(f2)-np.array(X[1])))
103    E_err_3.append(max(np.array(f3)-np.array(X[2])))
104
105 graph_ana(t,Analytic)
106 plt.title("Euler Method for different N")
107 plt.xlabel("x")
108 plt.ylabel("$y_i$")
109 plt.legend()
110 plt.grid()

```



```

111 plt.show()
112
113 for i in range(0,6,1):
114     N=10**(i)
115     X,Y,Z,t=output(Func,IC,a,b,N)
116     plot3(Y,t,N)
117     f1=Ana_Sol_1(t)
118     f2=Ana_Sol_2(t)
119     f3=Ana_Sol_3(t)
120     RK2_err_1.append(max(np.array(f1)-np.array(Y[0])))
121     RK2_err_2.append(max(np.array(f2)-np.array(Y[1])))
122     RK2_err_3.append(max(np.array(f3)-np.array(Y[2])))
123
124 graph_ana(t,Analytic)
125 plt.title("RK2 Method for different N")
126 plt.xlabel("x")
127 plt.ylabel("$y_i$")
128 plt.legend()
129 plt.grid()
130 plt.show()
131
132 for i in range(0,6,1):
133     N=10**(i)
134     X,Y,Z,t=output(Func,IC,a,b,N)
135     plot3(Z,t,N)
136     f1=Ana_Sol_1(t)
137     f2=Ana_Sol_2(t)
138     f3=Ana_Sol_3(t)
139     RK4_err_1.append(max(np.array(f1)-np.array(Z[0])))
140     RK4_err_2.append(max(np.array(f2)-np.array(Z[1])))
141     RK4_err_3.append(max(np.array(f3)-np.array(Z[2])))
142
143 graph_ana(t,Analytic)
144 plt.title("RK4 Method for different N")
145 plt.xlabel("x")
146 plt.ylabel("$y_i$")
147 plt.legend()
148 plt.grid()
149 plt.show()
150
151 print("\n")
152 data=np.column_stack((N_arr,E_err_1,RK2_err_1,RK4_err_1,E_err_2,
153     RK2_err_2,RK4_err_2,E_err_3,RK2_err_3,RK4_err_3))
154 head=["N","E_y1(Euler)","E_y1(RK2)","E_y1(RK4)","E_y2(Euler)","E_y2(RK2)",
155     "E_y2(RK4)","E_y3(Euler)","E_y3(RK2)","E_y3(RK4)"]
156 table(data,head,"\nTable for N and E= max(|y_ana -y_num|) values for y0
157     ,y1 and y2 for all three methods ")
158
159 fig, (ax1,ax2,ax3) = plt.subplots(3,sharex=True)
160 fig.suptitle("log(E) vs log(N) plot",fontsize=15,c="r")
161 ax1.plot(N_arr,E_err_1,label="$y_{0}(Euler)$")
162 ax1.plot(N_arr,RK2_err_1,label="$y_{0}(RK2)$")
163 ax1.plot(N_arr,RK4_err_1,label="$y_{0}(RK4)$")
164 ax1.set(ylabel="log(E)= log(max(|y_ana -y_num|))",yscale="log",xscale="
165     log")

```

```

162 ax1.grid()
163 ax1.legend()
164
165 ax2.plot(N_arr,E_err_2,label="$y_{1}(Euler)$")
166 ax2.plot(N_arr,RK2_err_2,label="$y_{1}(RK2)$")
167 ax2.plot(N_arr,RK4_err_2,label="$y_{1}(RK4)$")
168 ax2.set(ylabel="log(E)= log(max(|y_ana -y_num|))",xscale="log",yscale="
    log")
169 ax2.grid()
170 ax2.legend()
171
172 ax3.plot(N_arr,E_err_3,label="$y_{2}(Euler)$")
173 ax3.plot(N_arr,RK2_err_3,label="$y_{2}(RK2)$")
174 ax3.plot(N_arr,RK4_err_3,label="$y_{2}(RK4)$")
175 ax3.set(xlabel="log(N)",ylabel="log(E)= log(max(|y_ana -y_num|))",
    xscale="log",yscale="log")
176 ax3.grid()
177 ax3.legend()
178 plt.show()
179
180 x_f=[1,2.5,5.0,7.5,10]
181 for i in x_f:
182     l1,l2,l3,t=output(Func,IC,0,i,100)
183     for k in range(len(l1)):
184         plt.plot(t,l1[k],label="Y at xf="+str(i))
185     plt.title("euler")
186     plt.xlabel("x")
187     plt.ylabel("$y_i$")
188     plt.legend()
189     plt.grid()
190 plt.show()
191
192 for i in x_f:
193     l1,l2,l3,t=output(Func,IC,0,i,100)
194     for k in range(len(l1)):
195         plt.plot(t,l2[k],label="Y at xf="+str(i))
196     plt.title("RK2")
197     plt.xlabel("x")
198     plt.ylabel("$y_i$")
199     plt.legend()
200     plt.grid()
201 plt.show()
202
203 for i in x_f:
204     l1,l2,l3,t=output(Func,IC,0,i,100)
205     for k in range(len(l1)):
206         plt.plot(t,l3[k],label="Y at xf="+str(i))
207     plt.title("RK4")
208     plt.xlabel("x")
209     plt.ylabel("$y_i$")
210     plt.legend()
211     plt.grid()
212 plt.show()
213
214 def f3_b(x,var):

```

```

215     y,u=var
216     f1=u
217     f2=np.exp(2*x)*np.sin(x) -2*y +2*u
218     return np.array([f1,f2])
219
220 ic=[-0.4,-0.6]
221 z=RK2(f3_b,ic,0,1,5)
222 y=z[0].T
223
224 head=["x","y","y'"]
225 data_3b=np.column_stack((z[1],y[0],y[1]))
226 table(data_3b,head,"TABLE for output for q3(b)")
227
228 plt.plot(z[1],y[0],label="Y",marker=".")
229 plt.plot(z[1],y[1],label="Y'",marker=".")
230 plt.xlabel("x")
231 plt.title("Y and Y' for Q3(B)")
232 plt.legend()
233 plt.grid()
234 plt.show()

```

```

1 from MyIVP import euler
2 from MyIVP import RK2
3 from MyIVP import RK4
4 import numpy as np
5
6
7 def euler_tol(Func, IC, a, b, N, N_max, tol):
8     w=0
9     Val=[]
10    N_arr=[]
11    count=0
12    while N<=N_max:
13        g=euler(Func, IC, a, b, N)[0].T
14        t=euler(Func, IC, a, b, N)[1]
15        Val.append(g)
16        N_arr.append(N)
17        if count>=1:
18            J=[]
19            K=[]
20            for i in range(len(IC)):
21                J.append(Val[-1][i][-1])
22                K.append(Val[-2][i][-1])
23            J=np.array(J)
24            K=np.array(K)
25            ff=[]
26            for g1, g2 in zip(J, K):
27                if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
28                    err=abs(g1-g2)
29                else:
30                    err=abs((g2-g1)/g1)
31                ff.append(err)
32            if max(ff)<=tol:
33                w=1
34                break
35            else:
36                pass
37
38        N=2*N
39        count+=1
40    if w==0:
41        s=("N_max reached without achieving required tolerance")
42    elif w==1:
43        s="Given tolerance achieved with", N_arr[-1], "sub-intervals"
44
45    return Val, Val[-1], N_arr[-1], N_arr, s, g, t
46
47 def RK2_tol(Func, IC, a, b, N, N_max, tol):
48     w=0
49     Val=[]
50     N_arr=[]
51     count=0
52     while N<=N_max:
53         g=RK2(Func, IC, a, b, N)[0].T
54         t=RK2(Func, IC, a, b, N)[1]
55         Val.append(g)

```

```

56     N_arr.append(N)
57     if count>=1:
58         J=[]
59         K=[]
60         for i in range(len(IC)):
61             J.append(Val[-1][i][-1])
62             K.append(Val[-2][i][-1])
63         J=np.array(J)
64         K=np.array(K)
65         ff=[]
66         for g1,g2 in zip(J,K):
67             if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
68                 err=abs(g1-g2)
69             else:
70                 err=abs((g2-g1)/g1)
71             ff.append(err)
72         if max(ff)<=tol:
73             w=1
74             break
75         else:
76             pass
77
78     N=2*N
79     count+=1
80     if w==0:
81         s=("N_max reached without achieving required tolerance")
82     elif w==1:
83         s="Given tolerance achieved with",N_arr[-1],"sub-intervals"
84
85     return Val,Val[-1],N_arr[-1],N_arr,s,g,t
86
87 def RK4_tol(Func, IC ,a,b,N,N_max,tol):
88     w=0
89     Val=[]
90     N_arr=[]
91     count=0
92     while N<=N_max:
93         g=RK4(Func, IC ,a,b,N)[0].T
94         t=RK4(Func, IC ,a,b,N)[1]
95         Val.append(g)
96         N_arr.append(N)
97         if count>=1:
98             J=[]
99             K=[]
100             for i in range(len(IC)):
101                 J.append(Val[-1][i][-1])
102                 K.append(Val[-2][i][-1])
103             J=np.array(J)
104             K=np.array(K)
105             ff=[]
106             for g1,g2 in zip(J,K):
107                 if abs(g1)<=0.1e-5 or abs(g2)<=0.1e-5:
108                     err=abs(g1-g2)
109                 else:
110                     err=abs((g2-g1)/g1)

```

```

111         ff.append(err)
112         if max(ff)<=tol:
113             w=1
114             break
115         else:
116             pass
117
118         N=2*N
119         count+=1
120     if w==0:
121         s=("N_max reached without achieving required tolerance")
122     elif w==1:
123         s="Given tolerance achieved with",N_arr[-1],"sub-intervals"
124
125     return Val,Val[-1],N_arr[-1],N_arr,s,g,t

```