Shooting Method:Linear Boundary value problem
Lab Report for Assignment No. 9

SHASHVAT JAIN
(2020PHY1114)

HARSH SAXENA
(2020PHY1162)

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

May 1, 2022

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

Contents

1 Theory

1.1 IVP vs BVP

The solution $x(t)$ and/or its derivatives are required to have specific values at a single point, for example,$x(0) = 1$ and $x'(0) = 2$. Such problems are traditionally called **'initial value problems (IVPs)'** because the system is assumed to start evolving from the fixed initial point (in this case, 0).
The solution $x(t)$ is required to have specific values at a pair of points, for example,$x(0) = 1$ and $x(1) = 3$. These problems are known as **'boundary value problems (BVPs)'** because the points 0 and 1 are regarded as boundary points (or edges) of the domain of interest in the application.
Suppose that a given n^{th} order differential equation is written

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

A set of n constraints or boundary conditions must be given in order to completely determine the solutions. Generally these take the form of values of the function $y(x)$ and its derivative at the initial and final points of an interval over which solutions are required. If all n boundary conditions are given for $x = a$, then the problem becomes an **initial value problem**.
But if n_1 conditions are given at the initial point where $n_1 < n$,$y(a), y_1(a), ..., y_{n_1}(a)$ and that $n_2(n - n_1)$ are given at the final point $y(b), y_1(b)$... then it is a **'Two point boundary value'** problem.

1.2 Overview of Boundary value problems

General Form:

$$y'' = f\left(x, y, \frac{dy}{dx}\right)$$

$$f\left(x, y, \frac{dy}{dx}\right) = p(x)y + q(x)\frac{dy}{dx} + r(x)$$

Where second one is a Linear differential equation,Hence it becomes a Linear BVP

- **The Dirichlet (or first-type) boundary condition** is a type of boundary condition, When imposed on an ordinary or a partial differential equation, it specifies the values that a solution needs to take along the boundary of the domain. For an ordinary differential equation, for instance,

  $$y'' + y = 0$$

  the Dirichlet boundary conditions on the interval $[a, b]$ take the form

  $$y(a) = \alpha, y(b) = \beta$$

  where α and β are given numbers.

- **The Neumann boundary condition** is a type of boundary condition, . When imposed on an ordinary (ODE) or a partial differential equation (PDE), it specifies the values that the derivative of a solution is going to take on the boundary of the domain.
  For an ordinary differential equation, for instance,

  $$y'' + y = 0$$

  the Neumann boundary conditions on the interval [a,b] take the form

  $$y'(a) = \alpha, y'(b) = \beta$$

  where α and β are given numbers.

- **Robin boundary conditions** are a weighted combination of Dirichlet boundary conditions and Neumann boundary conditions.
  If Ω is the domain on which the given equation is to be solved and $\partial\Omega$ denotes its boundary, the Robin boundary condition is:
  $$au + b\frac{\partial u}{\partial n} = g \quad on \;\; \partial\Omega$$
  for some non-zero constants a and b and a given function g defined on $\partial\Omega$.
  Here u is the unknown solution defined on Ω and $\frac{\partial u}{\partial n}$ denotes the normal derivative at the boundary.
  In one dimension for example $\Omega = [0, 1]$ Robin conditions become:
  $$au(0) - bu'(0) = g(0)$$
  $$au(1) + bu'(1) = g(1)$$
  Notice the change of sign in front of the term involving a derivative: that is because the normal to $[0, 1]$ at 0 points in the negative direction, while at 1 it points in the positive direction.

Suppose we have second order differential equation :

$$y'' + p(x)y' + q(x)y = g(x)$$

A BVP is **homogenous** if in addition to g(x)=0 , we have $y_0 = 0$ and $y_1 = k$ where $k \in \mathcal{R}$ were y_0 and y_1 are the values of the function and its derivative at the boundary points.
If a B.V.P is not homogenous it is known as **non-homogenous** B.V.P .

1.3 The Shooting Method

we have previously studied various methods of solving differential equation which were not bounded i.e, they had an initial value of function and its derivatives at the first boundary and we had solved them using Euler,RK2 or RK4 methods becasue of their varying degree of accuracy.In **shooting method** we try to convert the given BVP in corresponding IVP which satisfies the Boundary conditions
name of the shooting method is derived from analogy with the target shooting ,we shoot the target and observe where it hits the target, based on the errors, we can adjust our aim and shoot again in the hope that it will hit close to the target. We can see from the analogy that the shooting method is an iterative method.
This method deals with guessing the Initial value of function or its derivative at the first boundary,this approach requires selecting the proper conditions or the "Trajectory" so that the solution hits the "target" i.e, satisfies the boundary condition at the other end
Let's see how the shooting methods works using the second-order ODE given $f(a) = f_a$ and $f(b) = f_b$,

$$F\left(x, f(x), \frac{df(x)}{dx}\right) = \frac{d^2f(x)}{dx^2}$$

- We start the whole process by guessing $f'(a) = \alpha$, together with $f(a) = f_a$, we turn the above problem into an initial value problem with two conditions all on value $x = a$. This is the aim step.
- Now we use RK4 method, to interpolate to the other boundary b to find $f(b) = f_\beta$. This is the shooting step.
- Now we compare the value of f_β with f_b, usually our initial guess is not good, and $f_\beta \neq f_b$, but what we want is $f_\beta - f_b = 0$, therefore, we adjust our initial guesses and repeat. Until the error is acceptable, we can stop. This is the iterative step.

We can see that the ideas behind the shooting methods is very simple. But the comparing and finding the best guesses are not easy, this procedure is very tedious. But essentially, finding the best guess to get $f_\beta - f_b = 0$ is a root-finding problem, once we realize this, we have a systematic way to search for the best guess. Since f_β is a function of α, therefore, the problem becomes finding the root of $g(\alpha) - f_b = 0$.

1.4 Linear Shooting method

$$\begin{aligned} y'' + p(x)y' + q(x)y + r(x) &= 0 \\ \alpha_1 y(a) + \alpha_2 y'(a) &= \alpha_3 \\ \beta_1 y(b) + \beta_2 y'(b) &= \beta_3 \end{aligned}$$

We solve this Linear BVP by taking the use of Homogenity of the Differential equation i.e,
for Linear boundary value problems like these,We know that solution to every Linear Non-homogeneous differential equation can be written as a particular solution plus a constant times a solution to the corresponding Homogeneous Problem,This suggests working with not one but two or more initial conditions.
In our general case we know neither the Initial value of function nor the initial value of first derivative so both will have to be guessed

$$IVP1 \begin{cases} y" = p(x)y' + q(x)y + r(x) \\ y(a) = 0, y'(a) = 0 \end{cases}$$

$$IVP2 \begin{cases} y" = p(x)y' + q(x)y \\ y(a) = 1, y'(a) = 0 \end{cases}$$

$$IVP3 \begin{cases} y" = p(x)y' + q(x)y \\ y(a) = 0, y'(a) = 1 \end{cases}$$

Now our final solution will be

$$y"(x) = y_1(x) + c_1 y_2(x) + c_2 y_3(x)$$

where $y_1(x), y_2(x), y_3(x)$ are the solution to IVP1,IVP2,IVP3,to determine the value of constants c_1 and c_2 we need to satisfy both the boundary conditions.
From boundary condition at x = a

$$[\alpha_1 y_2(a) + \alpha_2 y_2'(a)]c_1 + [\alpha_1 y_3(a) + \alpha_2 y_3'(a)]c_2 = \alpha_3 - \alpha_1 y_1(a) - \alpha_2 y_1'(a) \quad (1)$$

$$\boxed{\alpha_1 c_1 + \alpha_2 c_2 = \alpha_3} \quad (2)$$

similarly at x = b

$$\boxed{[\beta_1 y_2(b) + \beta_2 y_2'(b)]c_1 + [\beta_1 y_3(b) + \beta_2 y_3'(b)]c_2 = \beta_3 - \beta_1 y_1(b) - \beta_2 y_1'(b)} \quad (3)$$

We solve for c_1 and c_2 using equation 2 and 3

Case 1: If $\alpha_2 = \beta_2 = 0$ i:e $y(a) = \alpha$ and $y(b) = \beta$

It becomes Dirichlet Boundary Conditions.
here we don't have the problem of not having value of function or its derivative at first boundary now we can formulate two IVP's one non-homogeneous with given initial value of y(a) and one Homogeneous with y(a) = 0 and any arbitrary value y'(a)

then let $y_1(x)$ be the solution of IVP1 and $y_2(x)$ of IVP2,then due to linearity of differential equation

$$y(x) = y_1(x) + cy_2(x)$$

at x = b

$$y(b) = y_1(b) + cy_2(b) \tag{4}$$

$$c = \frac{\beta - y_1(b)}{y_1(b)} \tag{5}$$

Case 2: If $\alpha_1 = \beta_1 = 0$ i:e $y'(a) = \alpha$ and $y'(b) = \beta$
It becomes Neumann Boundary Conditions.
here we will solve the following two IVP's

$$IVP1 \begin{cases} y" = p(x)y' + q(x)y + r(x) \\ y(a) = 0, y'(a) = \alpha \end{cases}$$

$$IVP2 \begin{cases} y" = p(x)y' + q(x)y \\ y(a) = 1, y'(a) = 0 \end{cases}$$

Similarly,

$$y(x) = y_1(x) + cy_2(x) \tag{6}$$

$$y'(x) = y_1'(x) + cy_2'(x) \tag{7}$$

Then at x=b in equation 7

$$c = \frac{\beta - y_1'(b)}{y_1'(b)} \tag{8}$$

2 Programming

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

def slope(x,S):

    return [*S[1:],-S[0] + np.sin(3*x)]
```

```
10
11 #Dirichlet or neumann bc
12
13 def linearshooting(a,b,z_a,z_b,M,f,s0,s1,tol,key = 'd',cf = None):
14     dict1 = {'d':0,'n':1,'r1':0,'r2':1}
15     def phi(s):
16         if key == 'd':
17            Ic = [z_a,s]
18         elif key == 'n':
19             Ic = [s,z_a]
20         else :
21             Ic = [s, (cf*s + z_a)]
22
23         res = RK4_vec(Ic,a,b,M,2,f)
24         y_c = res[1][:,dict1[key]]
25
26         return abs(z_b - y_c[-1]),res
27     S = np.zeros(1000)
28
29     S[0] = s0 ; S[1] = s1
30     res_list = [phi(S[0])[1],phi(S[1])[1]]
31     for i in range(2,len(S)):
32         S[i] = S[i-1] - (((S[i-1]-S[i-2])*(phi(S[i-1])[0]))/(phi(S[i-1])[0] - phi(S[i
   -2])[0]))
33         res_list.append(phi(S[i])[1])
34
35         if phi(S[i])[0] <= tol:
36            return phi(S[i])[1],S[:i+1],res_list
37     return phi(S[-1])[1]
38
39
40 def Y_anay(x):
41     return (3/8)*np.sin(x) - np.cos(x) - (1/8)*np.sin(3*x)
42
43 Y_anay = np.vectorize(Y_anay)
44
45
46 def plotting(f,arr,N,title,k_p,sp1_title,sp2_title):
47     sig_l = ['o','1','v','*','2','x','o','2','x','*','v','d']
48     fig,(ax1,ax2) = plt.subplots(1,2)
49     plt.suptitle(title)
50
51     for i in range(len(N)):
52         ax1.plot(arr[i][0],arr[i][1][:,0],f'--{sig_l[i]}',label = f'for {k_p} = {N[i
   ]}')
53         ax2.plot(arr[i][0],arr[i][1][:,1],f'--{sig_l[i]}',label = f'for {k_p} = {N[i
   ]}')
54     ax1.plot(arr[2][0],f(arr[2][0]),'b')
55     ax1.legend()
56     ax2.legend()
57     ax1.set_title(sp1_title)
58     ax2.set_title(sp2_title)
59     ax1.set_xlabel('X')
60     ax1.set_ylabel('Y')
61     ax2.set_xlabel('X')
62     ax2.set_ylabel("Y'")
63
64
65
66 def plot_s(a,b,z_a,z_b,key,tol,condition,cf = None):
67     S_arr = linearshooting(a,b,z_a,z_b,32,slope,0,1,tol,key,cf)[1:]
68     itr = S_arr[0]
```

```
    plt_ar = S_arr[1]

    plotting(Y_anay,plt_ar,itr,condition,'s',f'Y vs X for N = 32 and tol = {tol}', f"Y' vs X for N = 32 and tol = {tol}")


N = np.logspace(1,6,base=2,num = 6)

def solve_cnt(a,b,z_a,z_b,key,tol,condition,cf = None):

    df = []
    for i in range(len(N)):
        df1 = linearshooting(a,b,z_a,z_b,N[i],slope,0,1,tol,key,cf)[0]
        df.append(df1)

    plotting(Y_anay,df,N,condition,'N','Y vs X',"Y' vs X")




#Robin 2
solve_cnt(0,np.pi/2,-1,1,'r2',10**(-12),'Robin-2 BC',-1)
plot_s(0,np.pi/2,-1,1,'r2',10**(-12),'Robin-2 BC',-1)



def Error(a,b,f,N,tol):
    mat = linearshooting(a,b,-1,1,N,slope,0,1,tol,'r2',-1)[0]
    x = mat[0]
    h = x[1] - x[0]
    ynum = mat[1][:,0]
    E_l = abs(ynum - f(x))
    E_n = max(E_l)
    E_r = np.sqrt(np.sum(np.power(E_l,2))/len(E_l))
    data = {'x':x,'y_num':ynum,'y_analytic':f(x),'Error':E_l}
    df  = pd.DataFrame(data)
    return np.log(N),np.log(h),np.log(E_n),np.log(E_r),df


Error = np.vectorize(Error,otypes = [float,float,float,float,pd.DataFrame])

rev_t = Error(0,np.pi/2,Y_anay,N,10**(-8))

data1 = {'N':N,'h':np.exp(rev_t[1]),'E_max':np.exp(rev_t[2]),'E_rms':np.exp(rev_t[3])}
df2 = pd.DataFrame(data1)
df2.to_csv('er_n.csv')
print(df2)

cnv_max = []
cnv_rms = []
for i in range(len(rev_t[2])-1):
    cnv_max.append(rev_t[2][i+1]/rev_t[2][i])
    cnv_rms.append(rev_t[3][i+1]/rev_t[3][i])

data5 = {'N':N[1:],'Ratio of Max err':cnv_max,'Ratio of rms error':cnv_rms}
df5 = pd.DataFrame(data5)
df5.to_csv('er_ratio.csv')
print(df5)


rev_t[4][1].to_csv('y_er_4.csv')
```

```
rev_t[4][2].to_csv('y_er_8.csv')

def err_line(mat,key=0):
    dict2 = {0:'N',1:'h'}

    slope_max, intercept_max,r_value, p_value, std_err = stats.linregress(mat[key], mat[2])
    slope_rms, intercept_rms,r_value, p_value, std_err = stats.linregress(mat[key], mat[3])

    fig,ax = plt.subplots(1,2)
    plt.suptitle(f'Log Error plots vs {dict2[key]}')
    ax[0].plot(mat[key],mat[2],'o',label = 'error points')
    ax[0].plot(mat[key],mat[key]*slope_max + intercept_max,label = f'Slope = {slope_max}')
    ax[1].plot(mat[key],mat[3],'x',label = 'error points')
    ax[1].plot(mat[key],mat[key]*slope_rms + intercept_rms,label = f'Slope = {slope_rms}')
    ax[0].set_title(f'E_max vs {dict2[key]}')
    ax[1].set_title(f'E_rms vs {dict2[key]}')
    ax[0].set_xlabel(f'log{dict2[key]}')
    ax[1].set_xlabel(f'log{dict2[key]}')
    ax[0].set_ylabel('log(E_max)')
    ax[1].set_ylabel('log(E_rms)')
    ax[0].legend()
    ax[1].legend()
    plt.show()

    return slope_max,slope_rms


tu = err_line(rev_t,0)

print(tu)
```

3 Discussion

3.1 solving BVP

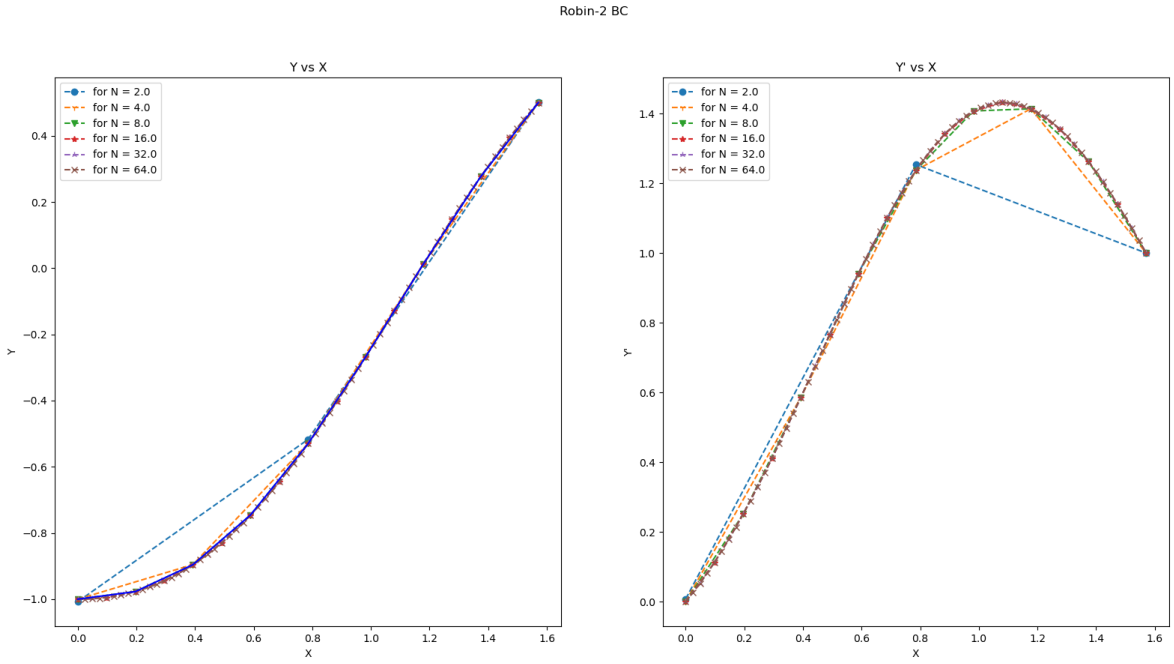

Figure 1: Solution of BVP for varying N

We have determined the function y for given differential equation and -

$$y'' + y = \sin(3x) \qquad\qquad 0 \leq x \leq \frac{\pi}{2}$$

with Mixed boundary condition as -

$$y(0) + y'(0) = -1, y'(\frac{\pi}{2}) = 1$$

1. We have determined the solution using shooting method,starting with guessing y'(0) for the tolerence 10^{-12}.
2. Note that as we increase The no. of terms **N** our shooted solution will match the analytic solution
3. as for the y'(x) for the given tolerence on s our y'(0) $\approx$ -0.1
4. since it is a Linear BVP hence we get our desired s in 3rd iteration

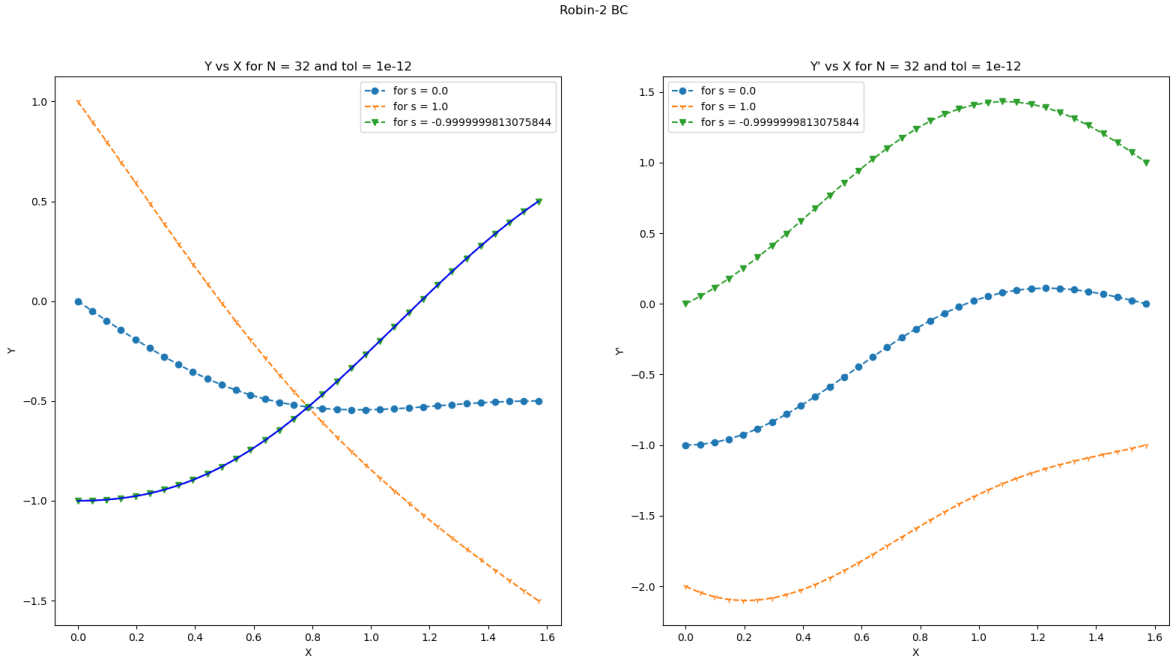

Figure 2: shooting

3.2 Errors

	N	h	E$_max$	E$_rms$
0	2.0	0.7853981633974483	0.012845522640442145	0.008554221577912384
1	4.0	0.39269908169872414	0.0004979072670148188	0.00035654436690789717
2	8.0	0.19634954084936204	3.7630094437635125e-05	2.370216140963086e-05
3	16.0	0.09817477042468102	2.6830999487748026e-06	1.5856616169181738e-06
4	32.0	0.049087385212340524	1.7721823297733876e-07	1.0269448895966886e-07
5	64.0	0.024543692606170262	1.1359973461910763e-08	6.531689347158815e-09

Table 1: Data for E_{max} and E_{rms}

We observe that E_{rms} is less than E_{max}

	N	Ratio of Max err	Ratio of rms error
0	4.0	1.7463871205732582	1.667402020881979
1	8.0	1.3395893350764143	1.3414629962895006
2	16.0	1.2592174505084708	1.253951045985287
3	32.0	1.211820414047516	1.204949400037405
4	64.0	1.1767211093718197	1.1712140928138026

Table 2: Data for E_N/E_{2N}

Note that as we increase N the ratio of errors converges to 1 i.e, the truncation error in RK4 reduces as we decrease step size.

	x	y_num	$y_analytic$	Error
0	0.0	-1.0001992587389088	-1.0	0.00019925873890880297
1	0.39269908169872414	-0.8954338290586165	-0.895858186938289	0.00042435787967254335
2	0.7853981633974483	-0.5298321786228959	-0.5303300858899107	0.0004979072670148188
3	1.1780972450961724	0.01161800530271484	0.01160682137227887	1.118393043596927e-05
4	1.5707963267948966	0.4995903671815504	0.49999999999999994	0.0004096328184495235

Table 3: Data for Error in Y values at N = 4

	x	y_num	$y_analytic$	Error
0	0.0	-1.0000027409787062	-1.0	2.7409787062282476e-06
1	0.19634954084936207	-0.9770570513702602	-0.9770726887746326	1.5637404372381702e-05
2	0.39269908169872414	-0.8958297693115691	-0.895858186938289	2.8417626719901e-05
3	0.5890486225480862	-0.7456974361800149	-0.7457289349705982	3.14987905832842e-05
4	0.7853981633974483	-0.5303062838348476	-0.5303300858899107	2.3802055063137928e-05
5	0.9817477042468103	-0.268147745269775	-0.2681554186581639	7.673388388917957e-06
6	1.1780972450961724	0.011594969547522704	0.01160682137227887	1.1851824756166363e-05
7	1.3744467859455345	0.2766091813373326	0.2766378596729012	2.867833556863264e-05
8	1.5707963267948966	0.4999623699055623	0.49999999999999994	3.76300944376351e-05

Table 4: Data for Error in Y values at N = 4

As We have increased number of terms the error in each y is reduced.

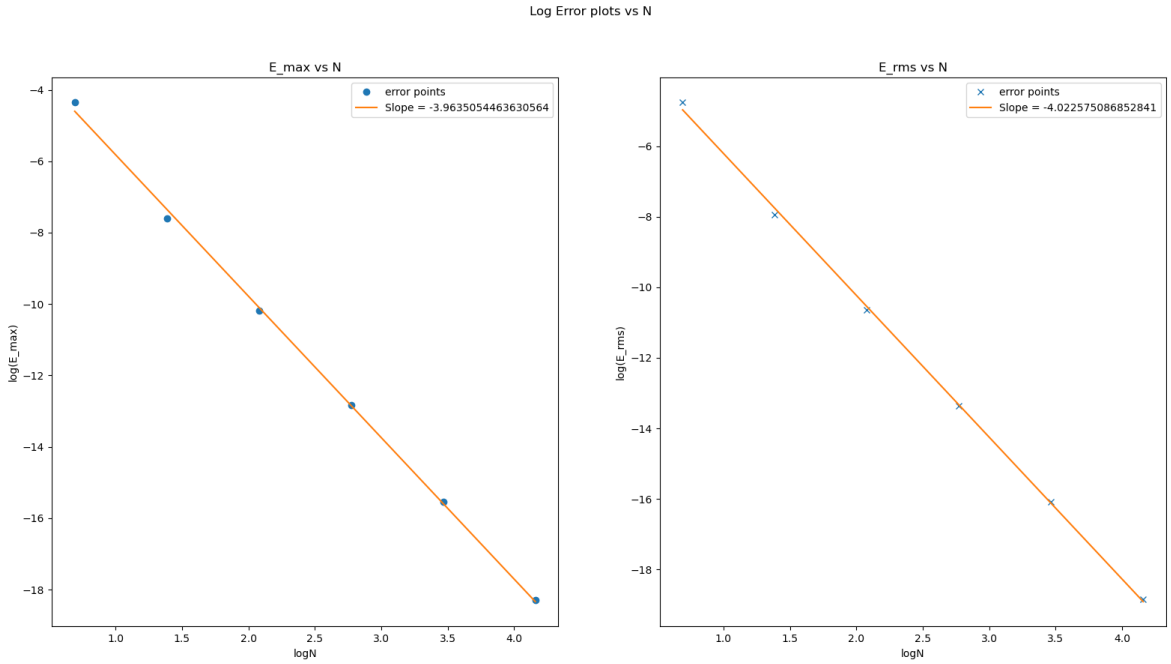

Figure 3: log(E) vs log(N)

$Slope_{abs}$ = -3.9635054463630564 $Slope_{rms}$ = -4.022575086852841

As we have used RK4 method to numerically determine the solution the global error in y is reducing as expected i.e, $E \propto N^{-4}$