Shooting Method:Non-linear Boundary value problem
Lab Report for Assignment No. 10

SHASHVAT JAIN
(2020PHY1114)

HARSH SAXENA
(2020PHY1162)

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

April 29, 2022

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

Contents

1 Theory

1.1 Shooting Method for Non Linear two Point Boundary Value Problem

The general second order differential equation is :

$$y'' = f(x, y, y'); \quad \text{a < x < b}$$

with robin boundary conditions:

$$\alpha_1 y(a) + \alpha_2 y'(a) = \alpha_3 \tag{1}$$
$$\beta_1 y(b) + \beta_2 y'(b) = \beta_3 \tag{2}$$

1. **case 1** if $\alpha_2 = \beta_2 = 0$ i.e y(a)= α and y(b) = β
   It becomes Dirichlet Boundary condition

2. **case 2** if $\alpha_1 = \beta_1 = 0$
   i.e y'(a)= α and y'(b) = β
   It becomes Neumann Boundary condition.

1. **In the case of Dirichlit Boundary conditions:**
   The solution to the boundary value is approximated by using the solution to a sequence of initial value problem involving a parameter S having the form,

   $$y'' = f(x, y, y')\text{for } a \leq x \leq b$$

   with y(a)=α and y'(a)= S (guess) This converts the problem into an IVP.
   Let, y(x,s) be the solution of this IVP.
   The solution of this problem satisfy:

   $$y(b, s) = \beta$$

   $$\text{If } \phi(s) = y(b, s) - \beta$$

   Thus the problem reduces to finding s=S^* such that $\phi(s^* = 0)$.

   Thus, we solve that BVP by using the solution of a sequence of IVP's involving parameter 'S'.
   we choose $s = s_k$ such that:

   $$\lim_{n\to\infty} y(b, s_k) = y(b) = \beta$$

   where y(x,s_k) denotes the solution of the IVP with $s = s_k$.
   while y(x) denotes the solution of the BVP.
   we choose the values of the x_k untill y(b,s_k) is sufficiently close to β.

   $$y(b, s_k) - \beta = 0$$

   This is an non linear equation which we can solve using Newton Raphson or Secant Method.

2. **In the case of Neumann Boundary conditions:**

   $$y'(a) = \alpha \text{ and } y'(b) = \beta$$

   Now y(a) is approximated and then improved in each iteration.

   we use the initial conditions:

   $$y(a) = s\text{and } y'(a) = \alpha$$

s is chosen such that

$$\begin{aligned}\phi(s) &= y'(b,s) - y'(b)\\ &= y'(b,s) - \beta\\ &= 0\end{aligned}$$

where, y(x) is the solution of BVP and y(x,s) is the solution of IVP with y(a)=s.

3. **In the case of Robin Boundary conditions:**
   Let us take an example where:

$$y(a) = \alpha$$
$$\beta_1 y(b) + \beta_2 y'(b) = \beta \text{ is given}$$

   we have to guess $y'(a) = s$ for this to convert it into a IVP.
   Therefore, the objective function whose roots are to be determined becomes:

$$\phi(s) = \beta - \beta_1 y(b,s) - \beta_2 y'(b,s)$$

   we will find the value of s iteratively such that $\phi(s)$ approaches 0.

   similarly, it can be done for the different cases .

1.2 Secant Method

Here we need two initial approximations, s_0 and s,
then the remaining terms of the sequence are generated by:

$$s_k = s_{k-1} - \phi(s_{k-1}) \left[\frac{s_{k-1} - s_{k-2}}{\phi(s_{k-1}) - \phi(s_{k-2})}\right] \tag{3}$$

for k=2,3,4,......

The initial value problem is solved initially for two values s_0 and s_1.
The iteration is stopped when $|\phi(s_k)|<$ tolerance

2 Algorithm

Algorithm 1 Objective

Data: INPUT -: all α's and β's are the values of boundary conditions, x_0 and x_f are the starting and ending point for x values,fucntion is the slope given for solving IVP, n is the order of Equation, N is the number of x points, guess value s

Result: OUTPUT -: returns the value of objective(psi) function,solution(y values), the x values

x_{val} = array of x values from x_0 to x_f ; `/* forms an array of y values using function from x values */`
if *alpha2 = 0* **then**
    initial conditions are α_1 and $s(guess\quad value)$
end
`/* if value of f(a) is given */`
else
    initial value $= \frac{\beta_1 - s \times \alpha_1}{\alpha_2}$
end
`/* if f(a) is not given */`
y = rk4(all required parameters) `/* calculate the y values */`
objective = β_2 - calculated β_2 value `/* calculating the value of psi */`

Algorithm 2 bvp$_{s}olution$

Data: INPUT -: all α's and β's are the values of boundary conditions, x_0 and x_f are the starting and ending point for x values,function is the slope given for solving IVP, n is the order of Equation, N is the number of x points, guess values, max no of iterations, $s_0, s_1, tolerance of solution$

Result: OUTPUT -: returns the value of y separately as y1 and y2, array of guess values , and x values

`/* initialise the y arrays , s arrays */`
`/* now get the output of objective function at s = 0 */`
objective(ψ) = objective(required parameters) **if** $\psi < tolerance$ **then**
    stop the function and give output
end
else
end
*now get the output of objective function at s = 1
objective(ψ) = objective(required parameters)
if $\psi < tolerance$ **then**
    stop the function and return the output
end
else
    calculate s_k = secant method(s_0, s_1, $\psi(s_0).\psi(s_1)$
        `/* now get the output of objective function at s =` s_k `*/`
    objective(ψ) = objective(required parameters)
end
`/* if this objective is less than tolerance than stop the function otherwise continue the above process till you reach k =` max$_{i}terations$ `*/`

3 Programming

```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):

    return [*S[1:],2*S[0]**3]

#Dirichlet or neumann bc

def linearshooting(a,b,z_a,z_b,M,f,s0,s1,tol,key = 'd',cf = None):
    dict1 = {'d':0,'n':1,'r':0}
    def phi(s):
        if key == 'd':
           Ic = [z_a,s]
        elif key == 'n':
            Ic = [s,z_a]
        elif key =='r':
            Ic = [s, (cf*s + z_a)]

        res = RK4_vec(Ic,a,b,M,2,f)
        y_c = res[1][:,dict1[key]]

        return abs(z_b - y_c[-1]),res
    S = np.zeros(1000)

    S[0] = s0 ; S[1] = s1
    res_list = [phi(S[0])[1],phi(S[1])[1]]
    for i in range(2,len(S)):
        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]))
        res_list.append(phi(S[i])[1])

        if phi(S[i])[0] <= tol:
           return phi(S[i])[1],S[:i],res_list
    return phi(S[-1])[1]


def Y_anay(x):
    return 1/(x+3)

Y_anay = np.vectorize(Y_anay)


def plotting(f,arr,N,title,k_p,sp1_title,sp2_title):
    sig_l = ['o','1','v','*','2','x','o','2','x','*','v','d']
    fig,(ax1,ax2) = plt.subplots(1,2)
    plt.suptitle(title)

    for i in range(len(N)):
        ax1.plot(arr[i][0],arr[i][1][:,0],f'--{sig_l[i]}',label = f'for {k_p} = {N[i
    ]}')
        ax2.plot(arr[i][0],arr[i][1][:,1],f'--{sig_l[i]}',label = f'for {k_p} = {N[i
    ]}')
    ax1.plot(arr[4][0],f(arr[4][0]),'b')
    ax1.legend()
    ax2.legend()
    ax1.set_title(sp1_title)
```

```
    ax2.set_title(sp2_title)
    ax1.set_xlabel('X')
    ax1.set_ylabel('Y')
    ax2.set_xlabel('X')
    ax2.set_ylabel("Y'")



def plot_s(a,b,z_a,z_b,key,tol,condition,cf = None):
    S_arr = linearshooting(a,b,z_a,z_b,32,slope,0,1,tol,key,cf)[1:]
    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")



#dirichlet bc
#solve_cnt(0,1,1/3,1/4,'d',10**(-8),'Dirichlet BC')
plot_s(0, 1,1/3,1/4,'d',10**(-8),'Dirichlet BC')
#neumann Bc
#solve_cnt(0,1,-1/9,-1/16,'n',10**(-3),'Neumann BC')

#Robin 1
#solve_cnt(0,1,-2/9,1/4,'r',10**(-3),'Robin-1 BC',1/3)
plot_s(0, 1,-2/9,1/4,'r',10**(-6),'Robin-1 BC',1/3)
#Robin 2
#solve_cnt(1,0,3/16,1/3,'r',10**(-3),'Robin-2 BC',-1)
plot_s(1,0,3/16,1/3,'r',10**(-3),'Robin-2 BC',-1)

'''
def Error(a,b,f,N,tol):
    mat = linearshooting(a,b,1/3,1/4,N,slope,0,1,tol)[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(0,1,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

Yu = err_line(rev_t,1)

tu = err_line(rev_t,0)

data2 = {'err':['E_max','E_rms'],'h':Yu,'N':tu}

df3 = pd.DataFrame(data2)
df3.to_csv('slope.csv')
print(df3)
'''
```

4 Discussion

4.1 Dirichlet condition

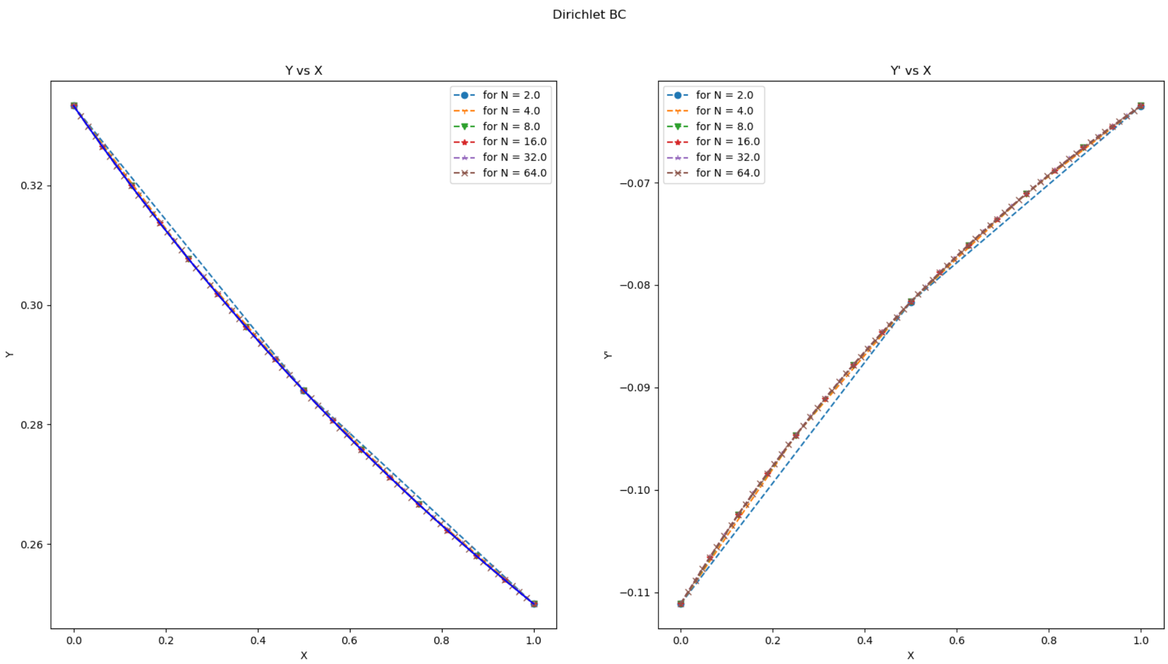

Figure 1: Dirichlet condition

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

$$y" = 2y^3 \qquad\qquad 0 \leq x \leq 1$$

with Dirichlet boundary condition as -

$$y(0) = \frac{1}{3}, y(1) = \frac{1}{4}$$

1. We have determined the solution using shooting method,starting with guessing y'(0) for the tolerence 10^{-8}.
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) $\approx$ -0.11

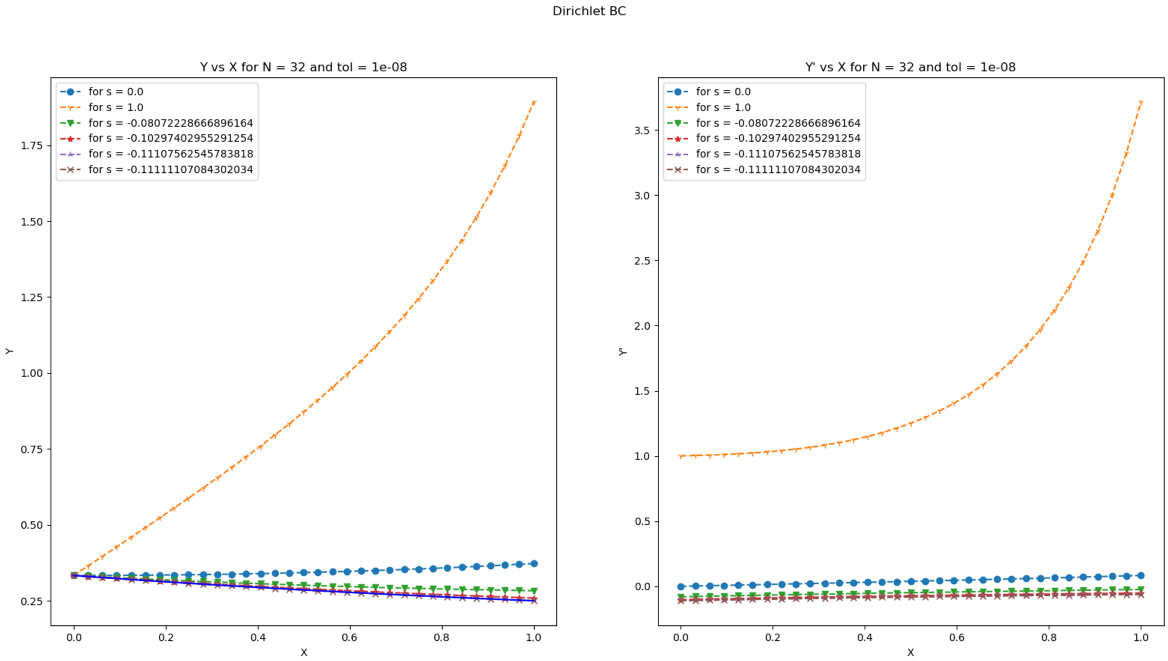

Figure 2: shooting for Dirichlet condition

4.2 Neumann condition

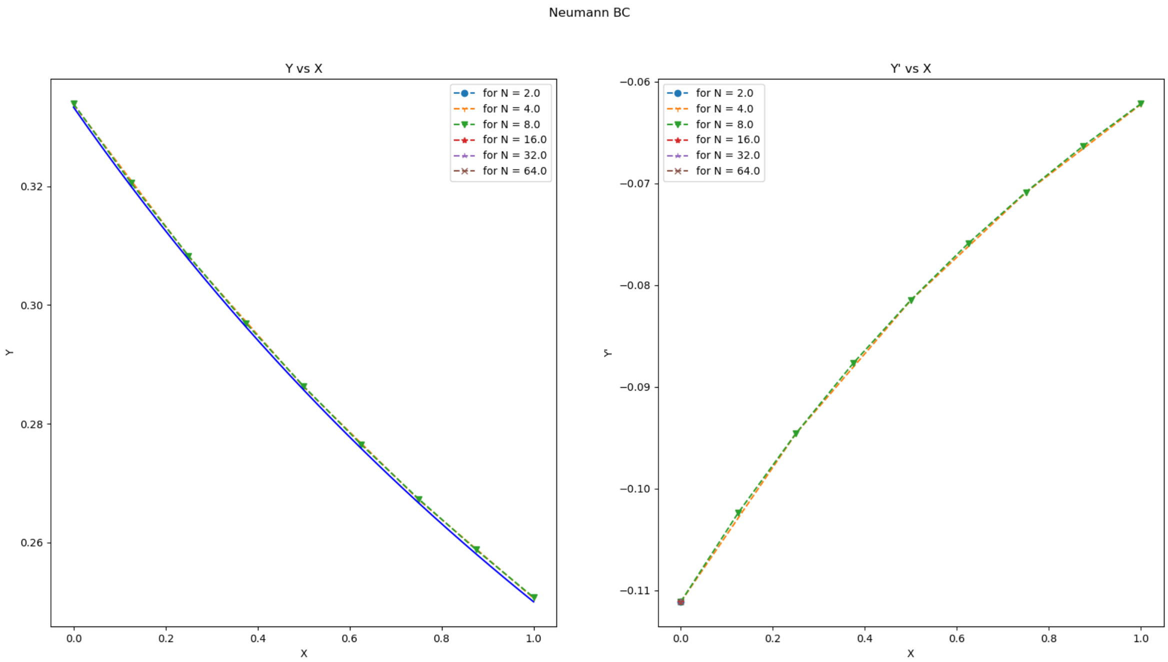

Figure 3: Neumann condition

$$y'(0) = -\frac{1}{9}, y(1) = -\frac{1}{16}$$

1. We have determined the solution using shooting method,starting with guessing y(0) for the tolerence 10^{-3}.
2. Note that as we increase The no. of terms **N** our shooted solution will match the analytic solution of y'(x) and y(x)

3. as for the y(x) for the given tolerence s our y(0) ≈ 0.325

4.3 Robin Condition

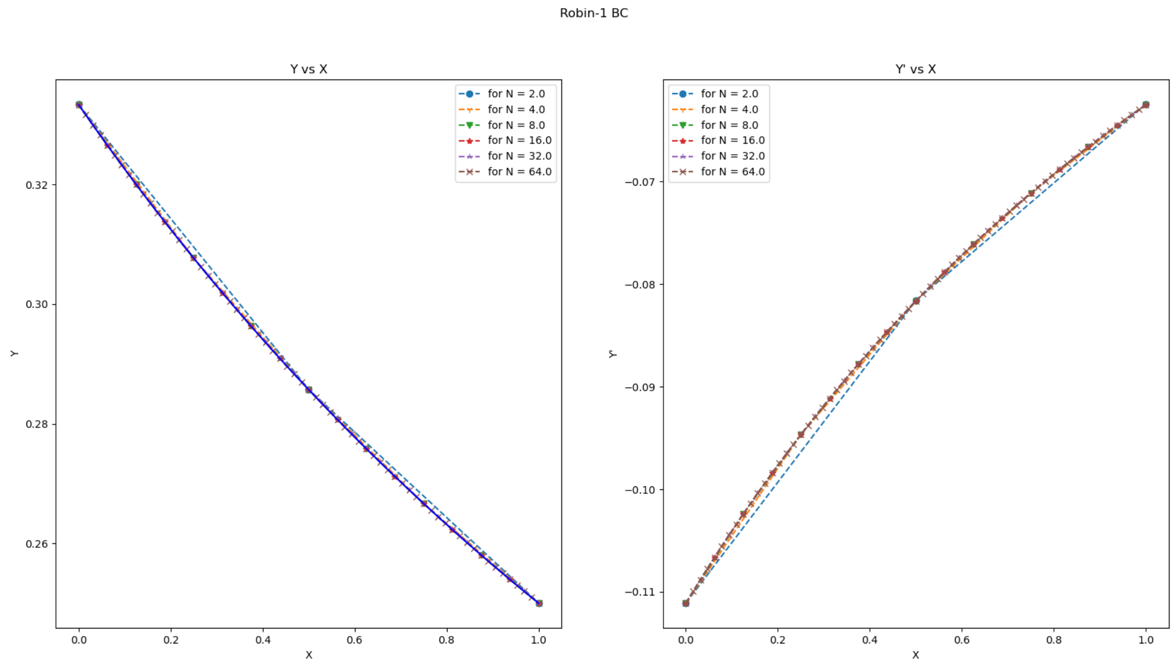

Figure 4: Robin - 1 condition

condition: $3y(0) - 9y'(0) = 2, y(1) = \frac{1}{4}$

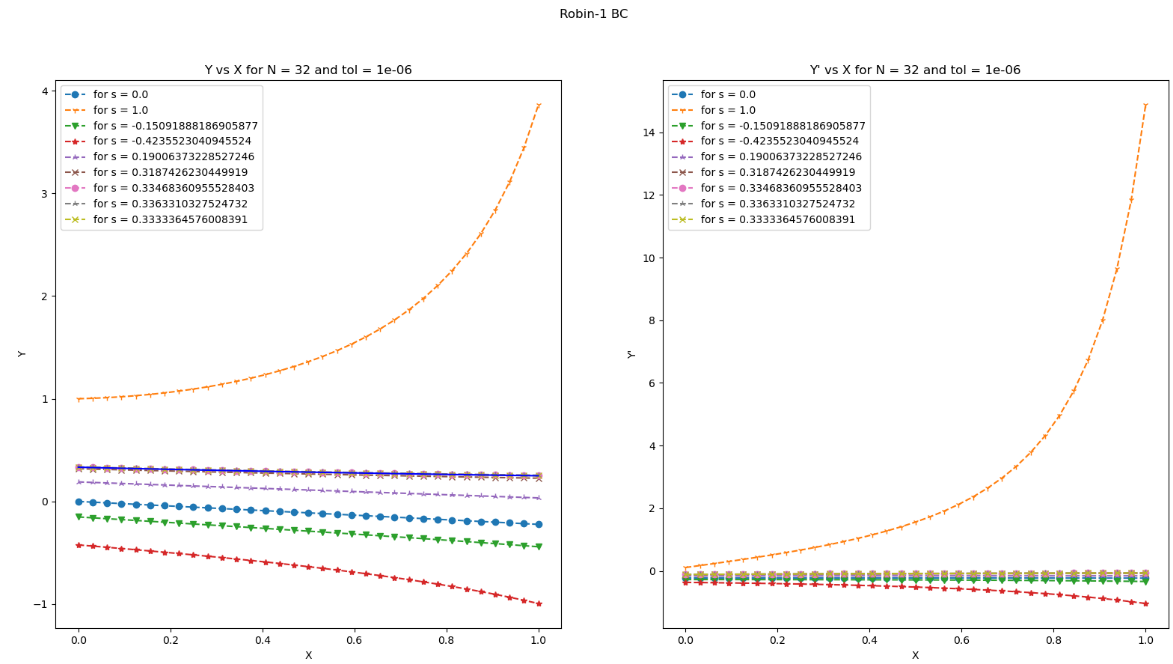

Figure 5: shooting for Dirichlet condition

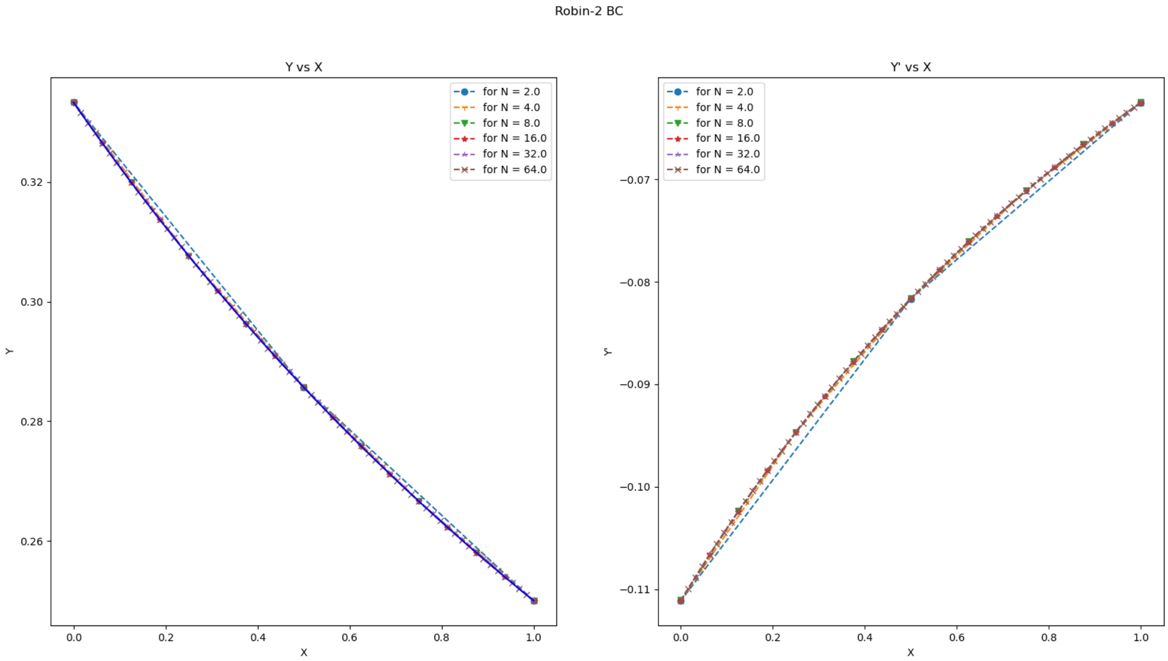

Figure 6: Robin - 2 condition

condition: $y(0) = \frac{1}{3}, 2y(1) + 2y'(1) = \frac{3}{8}$

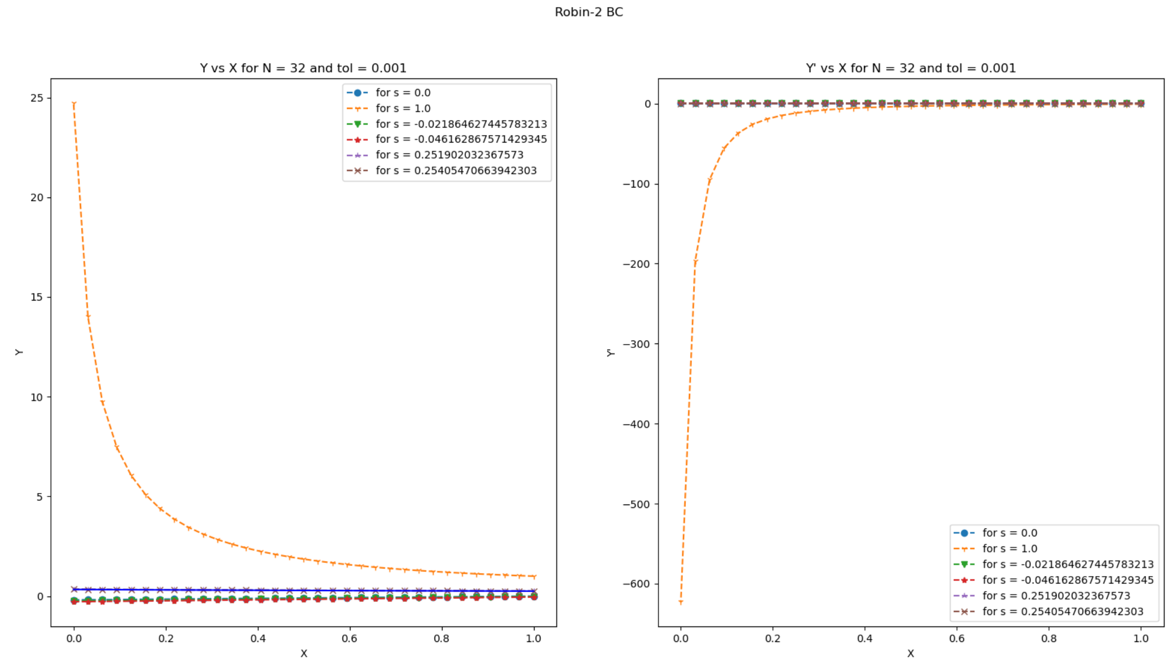

Figure 7: shooting for Dirichlet condition

We observe that we get approximately same solution for all the boundary condition.

4.4 Errors

	N	h	E_{max}	E_{rms}
0	2	0.5	2.88E-06	1.66E-06
1	4	0.25	1.72E-07	1.14E-07
2	8	0.125	1.06E-08	7.30E-09
3	16	0.0625	6.52E-10	4.60E-10
4	32	0.03125	4.04E-11	2.90E-11
5	64	0.015625	2.59E-12	1.90E-12

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	1.2210233244540545	1.2011221336016977
1	8.0	1.1787828603493995	1.1721425508872567
2	16.0	1.1518522312705337	1.1475281335959933
3	32.0	1.1314164681263084	1.128640424367299
4	64.0	1.1147601612022222	1.1123097852123878

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	0.333333333	0.333333333	0
1	0.25	0.307692462	0.307692308	1.54E-07
2	0.5	0.285714457	0.285714286	1.72E-07
3	0.75	0.266666777	0.266666667	1.10E-07
4	1	0.25	0.25	2.07E-13

Table 3: Data for Error in Y values using Dir-Condition

	x	y_{num}	$y_{analytic}$	Error
0	0	0.333333333	0.333333333	0
1	0.125	0.320000006	0.32	6.01E-09
2	0.25	0.307692317	0.307692308	9.31E-09
3	0.375	0.296296307	0.296296296	1.06E-08
4	0.5	0.285714296	0.285714286	1.04E-08
5	0.625	0.275862078	0.275862069	8.95E-09
6	0.75	0.266666673	0.266666667	6.64E-09
7	0.875	0.25806452	0.258064516	3.61E-09
8	1	0.25	0.25	2.17E-13

Table 4: Data for Error in Y values using Dir-Condition

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

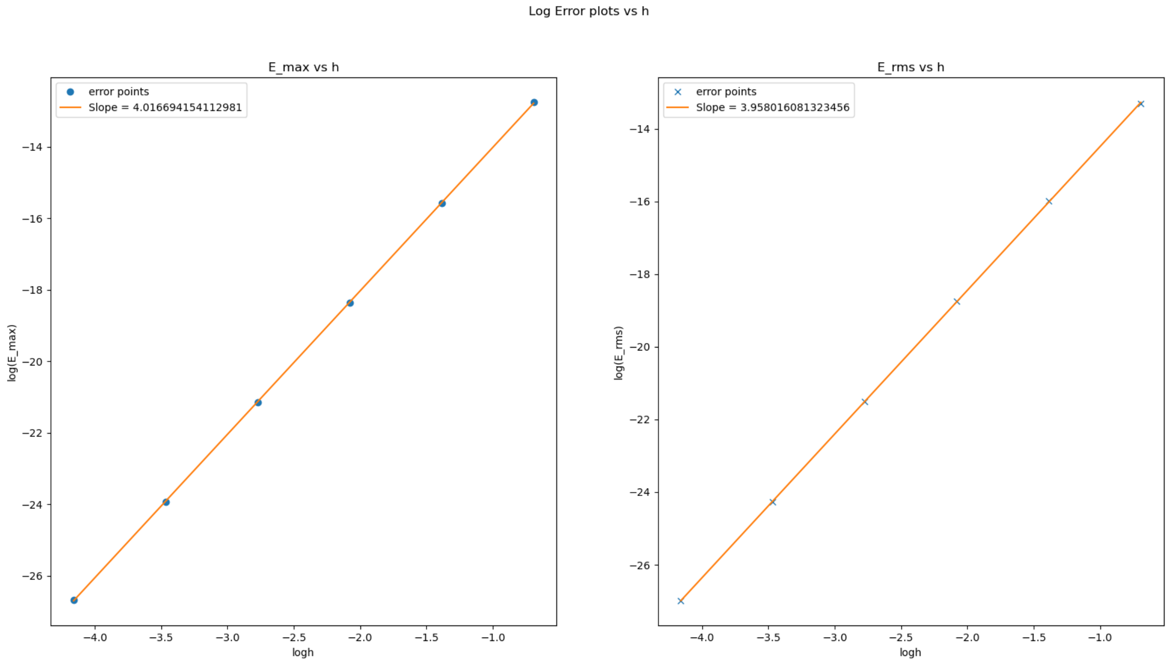

Figure 8: log(E) vs log(h)

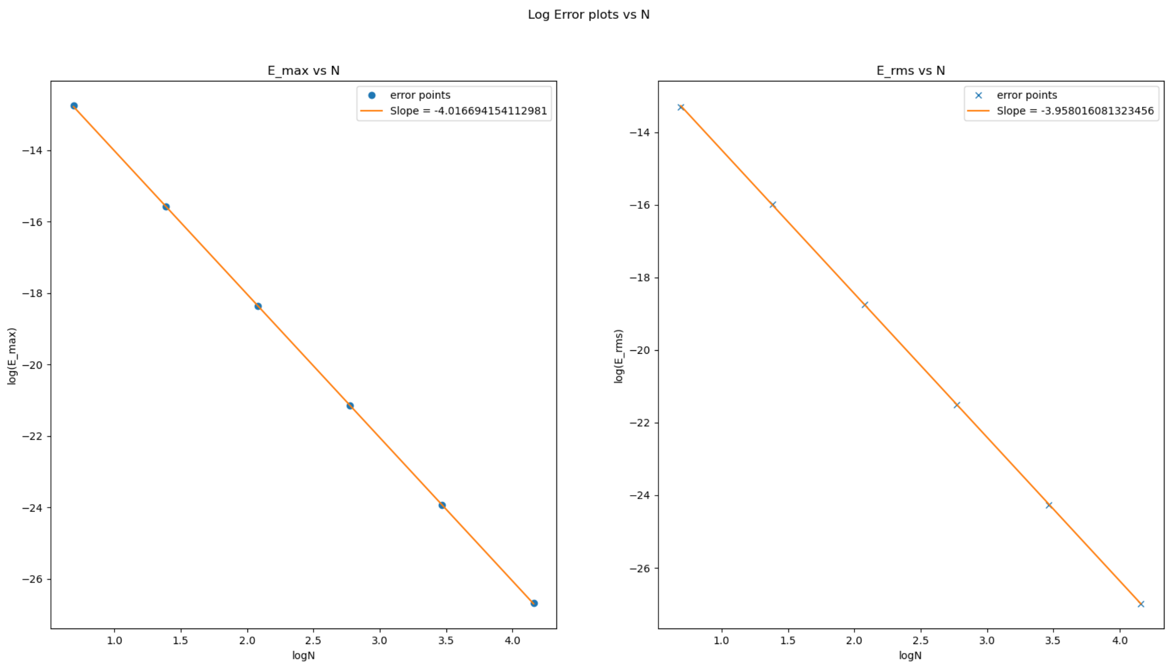

Figure 9: log(E) vs log(N)

	err	h	N
0	E$_max$	4.016694154112981	-4.016694154112981
1	E$_rms$	3.958016081323456	-3.958016081323456

Table 5: tabulated slopes for error line

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