Assignment 9 - Linear Shooting

SGTB Khalsa College, University of Delhi

Preetpal Singh(2020PHY1140)(20068567043)

Ankur Kumar(2020PHY1113)(20068567010)

Unique Paper Code: 32221401

Paper Title: Mathematical Physics III

Submitted on: April 18, 2022

B.Sc(H) Physics Sem IV

Submitted to: Dr. Mamta

Theory

1 Initial Value Problem

The general solution of any differential equation gives information about the structure of the complete solution space for the problem.The problems in which we are given the value of function y(x) and its derivative $y^{'(x)}$ at the same point say at x=0 i.e. y(0)=x_1 and $y^{'(0)=x_2}$.Such problems are traditionally called initial value problems (IVP's) because the system is assumed to start evolving from the fixed initial point.

2 Boundary Value Problem

.The problems in which we are given the value of function on more than one points say at x=0 , x=1 i.e $y(0) = 1$ and $y(1) = 5$.These problems are known as boundary value problems (BVP's) because the points 0 and 1 are regarded as boundary points (or edges) of the domain of interest in the application.

2.1 Difference between Boundary value and Initial value

An initial value problem is how to aim my gun. A boundary value problem is how to aim my gun so that the bullet hits the target.

In short the difference is given in the table below:

IVP	BVP
The value of the function and its derivative is given at the same point	The value of the function is given at more than one points
This usually apply for dynamic system that is changing over time	This is very useful for a system that has space boundary

3 Two- Point Boundary Value Problem

Here we will discuss about the Two- Point Boundary Value problem these are the differential equations whose values are given at two points of the defined domain. There are three types of two point boundary conditions :

1. Dirichlet Boundary Condition
2. Neumann Boundary Condition
3. Robin Boundary Condition

We further illustrate these condition in coming sections.

3.0.1 General form of the second order boundary value problem (BVP)

The general form is given by:

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

3.0.2 Types of boundary conditions

1. **Dirichlet Boundary Condition:-** If the value of the unknown function is specified at two points i.e $y(0) = 1$ and $y(1) = 5$, then it is Dirichlet boundary condition.

2. **Neumann Boundary Condition:-** If the value of the first derivative of the unknown function is specified at two points i.e $y^{'(0)=1}$ and $y^{'(1)=3}$, then it is Neumann boundary condition.

3. **Robin Boundary Condition:-** If the value of unknown function and the first derivative of the unknown function is specified at two points i.e $y(0) = 1$ and $y^{'(1)=3}$, then it is Robin boundary condition.It is also known as mixed boundary condition.

3.0.3 Homogeneous and Non-homogeneous BVP

If we have a boundary value problem in form of :

$$y^{''} + p(x)y' + q(x)y = g(x)$$

Then we can say that the given boundary value problem is convergence if g(x)=0 along with $y_0 = 0$ and $y_1 = 0$ (regardless of the boundary conditions we use). If any of these are not zero we will call the BVP non-homogeneous.

4 Shooting Method

In numerical analysis, the shooting method is a method for solving a boundary value problem by reducing it to an initial value problem. It involves finding solutions to the initial value problem for different initial conditions until one finds the solution that also satisfies the boundary conditions of the boundary value problem.

5 Linear Shooting Method

since the given equation is :

$$y^{''} + p(x)y' + q(x)y + r(x) = 0$$

with robin boundary conditions:

$$\alpha_1 y(a) + \alpha_2 y^{'(a)} = \alpha_3$$

$$\beta_1 y(b) + \beta_2 y^{'(b)} = \beta_3$$

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.

For the computational grid, let N be a positive integer, and the partition the interval [a,b] into

$$a = x_0 < x_1 < x_2 .... < x_{n-1} < x_n = b$$

where $x_i = a + ih$ and $h = \frac{(b-a)}{N}$. Further, let ω_i denotes the approximation to the exact solution, y(x) , at $x = X_i$.
Evaluate the differential equations at each interior grid point $x = x_i(1 \leq i \leq N - 1$, replace the

derivatives by second order central difference approximations and collecting the like terms. The resulting computational template is :

$$(-1+\frac{h}{2}p_i)\omega_i+(2+h^2q_i)w_i+(-1+\frac{h}{2}p_i)\omega_{i+1}=-h^2r_i$$

Let's focus on the boundary condition at $x_0 = a$:

$$\alpha_1 y(a)+\alpha_2 y'(a)=\alpha_3$$

To maintain the second order accuracy of the other equations, we could replace the derivative in the boundary conditions by O(h^2) forward difference approximation.

$$y'1 \approx \frac{-3y_i+4yi+1-y_{i+2}}{2h}$$

6 Condition

suppose the Linear Boundary value problem :

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

with y(a)= α and y(b)= β.
if this satisfies:

1. p(x) , q(x) , r(x) are continuous on [a,b]
2. p(x) $>$ 0 on [a,b]

7 Initial Value Problem

The general solution of any differential equation gives information about the structure of the complete solution space for the problem.The problems in which we are given the value of function y(x) and its derivative $y^{'(x)}$ at the same point say at x=0 i.e. y(0)=x_1 and $y^{'(0)=x_2}$.Such problems are traditionally called initial value problems (IVP's) because the system is assumed to start evolving from the fixed initial point.

8 Boundary Value Problem

.The problems in which we are given the value of function on more than one points say at x=0 , x=1 i.e $y(0) = 1$ and $y(1) = 5$.These problems are known as boundary value problems (BVP's) because the points 0 and 1 are regarded as boundary points (or edges) of the do... [8:47 PM, 4/18/2022] Akarsh Shukla SGTB:

9 Linear Shooting Method

since the given equation is :

$$y^{''}+p(x)y'+q(x)y+r(x)=0$$

with robin boundary conditions:

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

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.

Explain Linear shooting method to solve the BVP

$$y''(x) + p(x)y'(x) + q(x)y(x) + r(x) = 0 \quad ; \quad a < x < b$$

with the Robin boundary conditions

$$\alpha_1 y(a) + \alpha_2 y'(a) = \alpha_3$$
$$\beta_1 y(b) + \beta_2 y'(b) = \beta_3$$

using RK4 for solving the corresponding IVP. Discuss the Neumann and Dirichlet conditions as a special case of this.

Consider BVP with Dirichlet Boundary Condition:

$$y'' = f(x, y, y')a \leq x \leq b$$
$$\text{with } \ y(a) = \alpha \ \ and \ \ y(b) = \beta$$

If we wish to solve this by converting it into initial value problem,we need to know value of y' at x= a so make an assumption for it .
Let y'(a)=s . so, the 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.

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.

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 .

10 Condition of Having Unique solution

suppose we have a boundary value problem as:

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

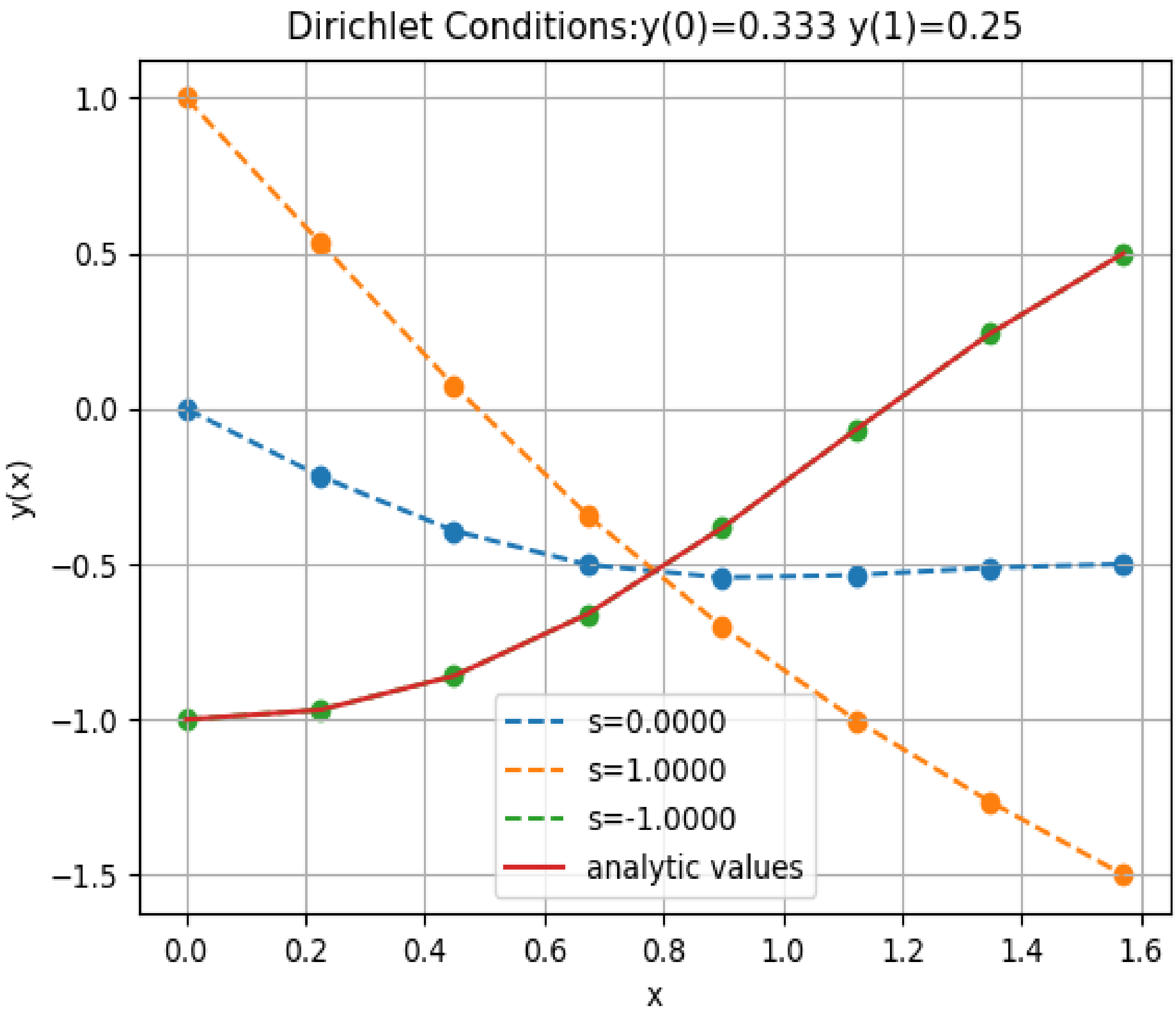
$$\text{with } \ y(a) = \alpha \ \ and \ \ y(b) = \beta$$

is continuous on set $D = (x, y, y'), a \leq x \leq b$
and that the partial derivatives f(y) and f'(y) are also continuous on D. If :

1. f(x,y,y') $>$ 0 , for all (x,y,y') $\in D$.
2. A constant M exists, with

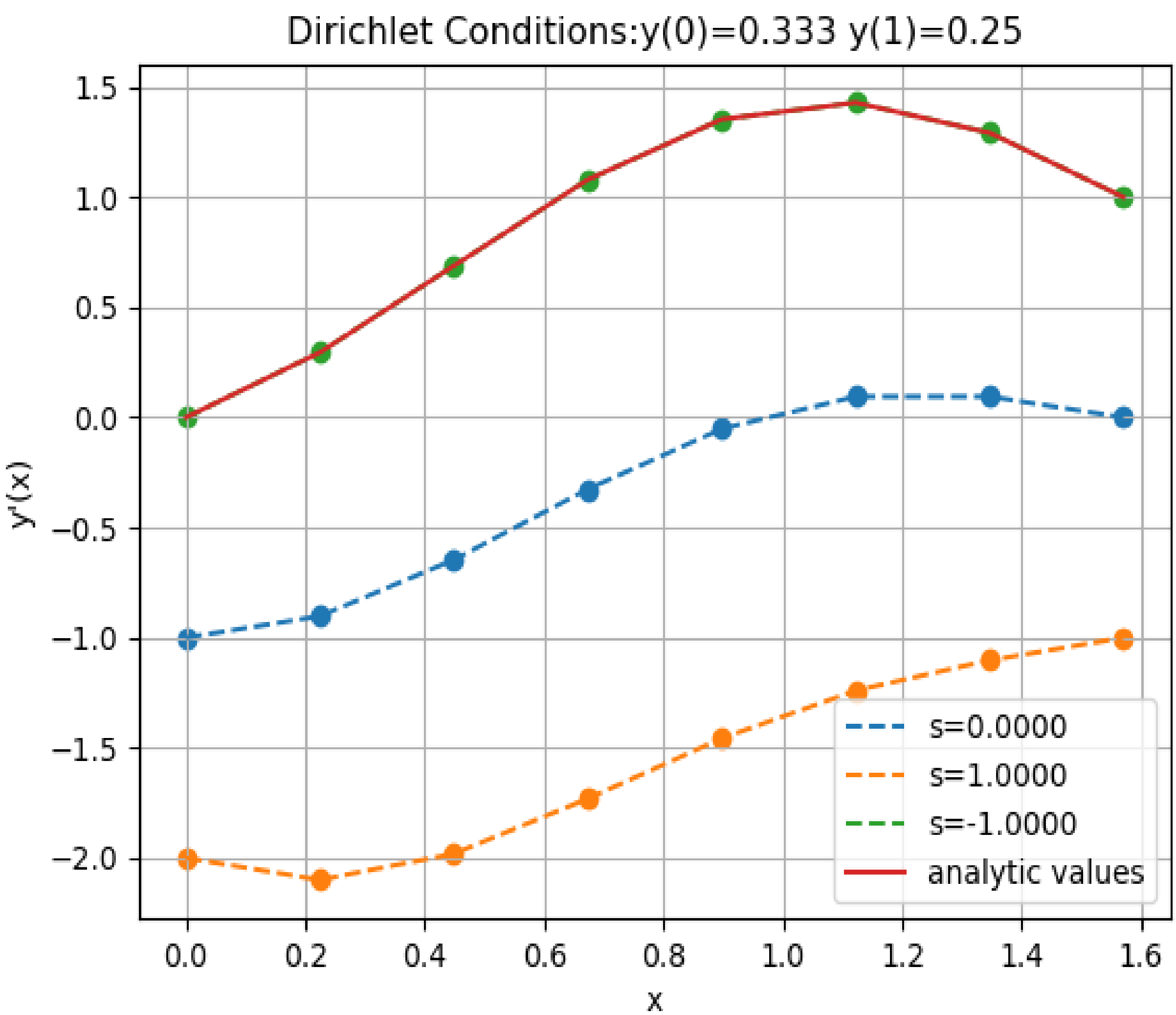
$$|f(x, y, y')| < M \ \text{ for all (x,y,y')} \in D$$

then the boundary-value problem has a unique solution.



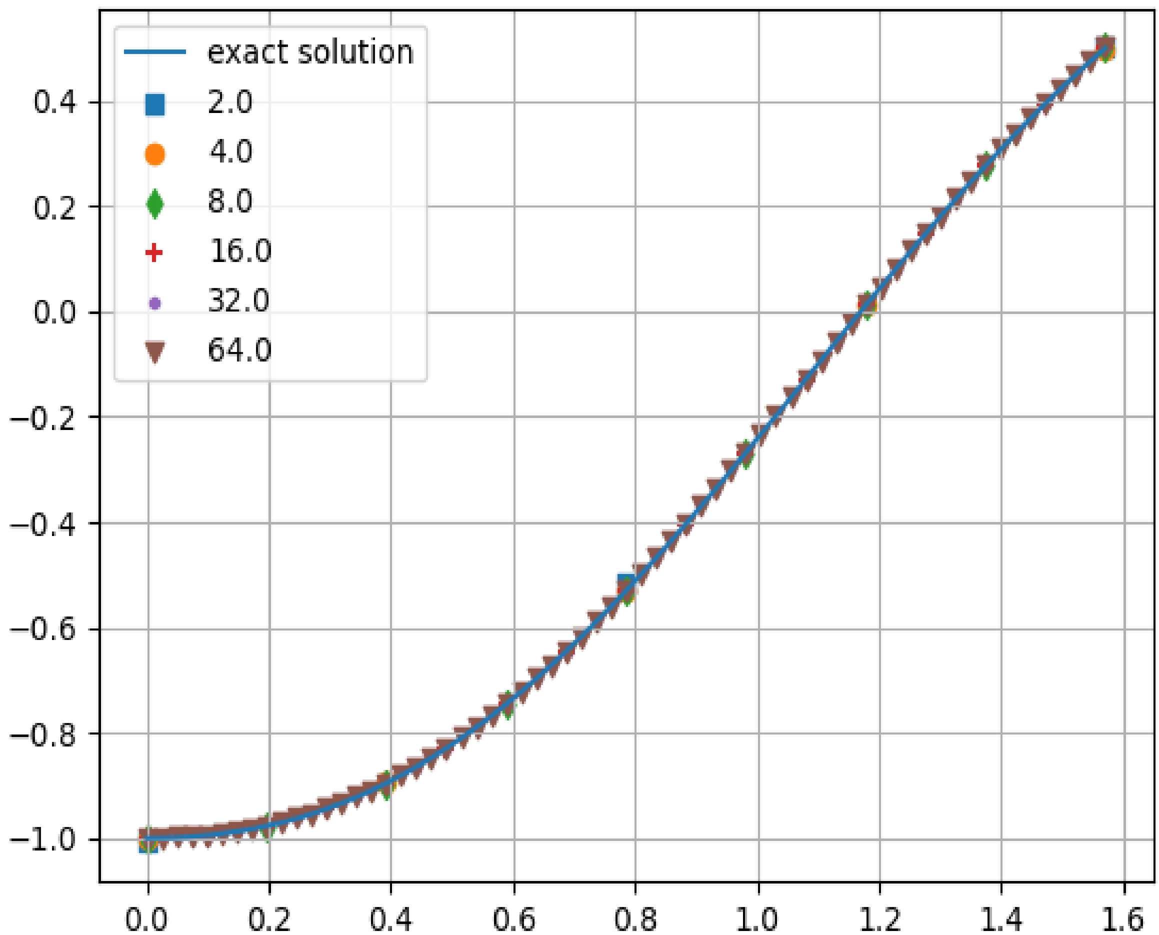
Number of points=4

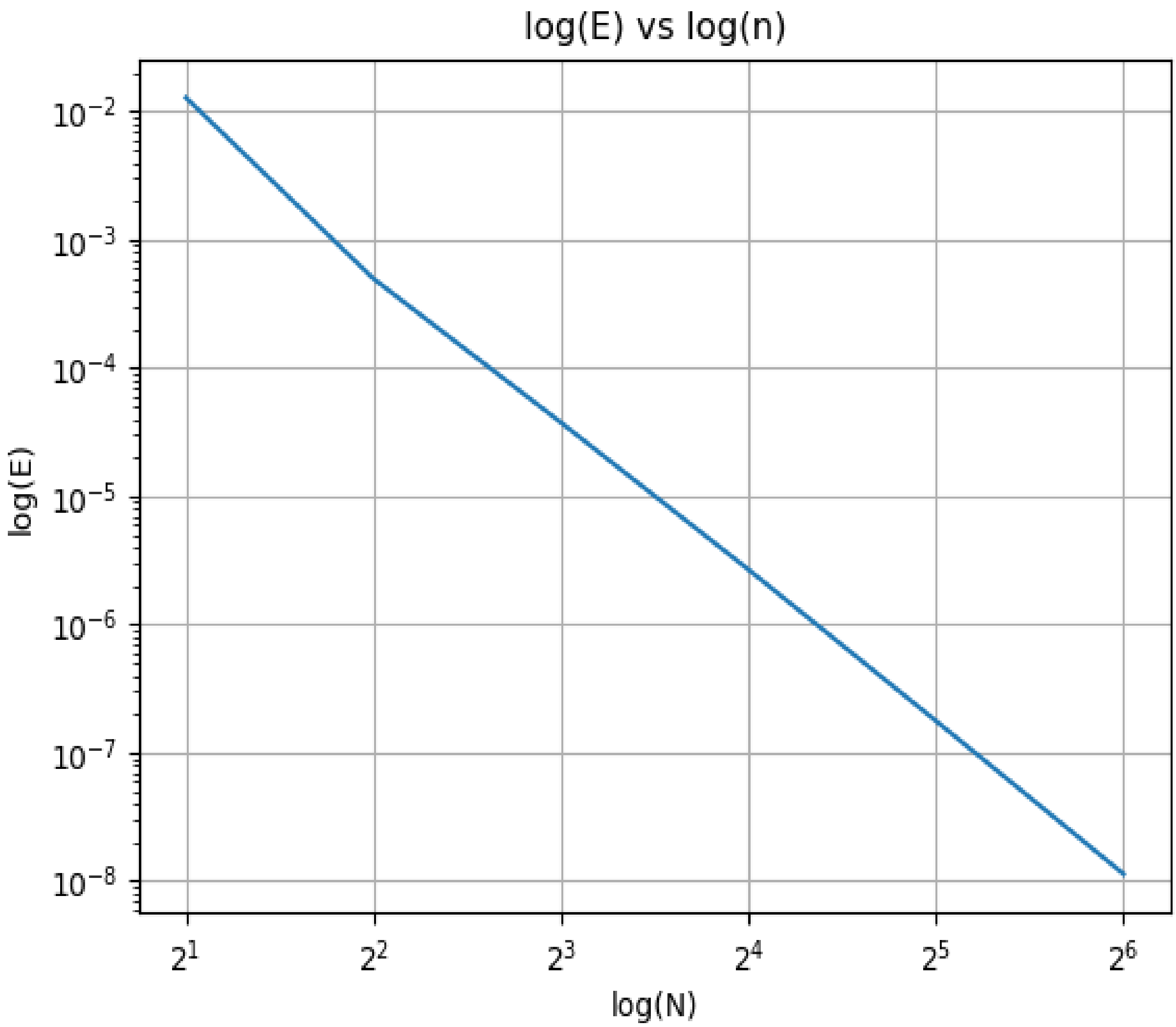
	x_i	y_num	y_anal	error
0	0.000000	-1.000199	-1.000000	0.000199
1	0.392699	-0.895434	-0.895858	0.000424
2	0.785398	-0.529832	-0.530330	0.000498
3	1.178097	0.011618	0.011607	0.000011
4	1.570796	0.499590	0.500000	0.000410



Number of points=8

	x_i	y_num	y_anal	error
0	0.000000	-1.000003	-1.000000	0.000003
1	0.196350	-0.977057	-0.977073	0.000016
2	0.392699	-0.895830	-0.895858	0.000028
3	0.589049	-0.745697	-0.745729	0.000031
4	0.785398	-0.530306	-0.530330	0.000024
5	0.981748	-0.268148	-0.268155	0.000008
6	1.178097	0.011595	0.011607	0.000012
7	1.374447	0.276609	0.276638	0.000029
8	1.570796	0.499962	0.500000	0.000038


exact solution
2.0
4.0
8.0
16.0
32.0
64.0
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6



	N	Absolute error	error ratio	RMS error	error ratio
0	2.0	1.284552e-02	NaN	8.554222e-03	NaN
1	4.0	4.979073e-04	25.799026	3.565444e-04	23.992026
2	8.0	3.763009e-05	13.231624	2.370216e-05	15.042694
3	16.0	2.683100e-06	14.024858	1.585662e-06	14.947805
4	32.0	1.772182e-07	15.140090	1.026945e-07	15.440572
5	64.0	1.135997e-08	15.600233	6.531689e-09	15.722501

slope,intercept: (-3.963505446041488, -0.8050378842152899)

Programming

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



def func_x(x,y_vec):
    ans_vec=np.zeros((2))
    ans_vec[0]=y_vec[1]
    ans_vec[1]=np.sin(3*x)+(-1*y_vec[0])
    return ans_vec

def row_cutting(mat,row_no):
    col_no=int(np.size(mat)/len(mat))
    new_mat=np.zeros((row_no,col_no))
    for i in range(row_no):
        new_mat[i,:]=mat[i,:]
    return new_mat

def graph_log_sketch(X,Y):
    fig,ax=plt.subplots()
    plt.plot(X,Y)
    ax.set_xscale("log",base=2)
    ax.set_yscale("log")
    ax.set_title("log(E) vs log(n)")
    ax.set_xlabel("log(N)")
    ax.set_ylabel("log(E)")
    plt.grid()
    plt.show()

def analy_y(x):
    return (3/8)*np.sin(x)-np.cos(x)-(1/8)*np.sin(3*x)

def analy_deriv_y(x):
    return (3/8)*np.cos(x)+np.sin(x)-(3/8)*np.cos(3*x)


def graph_sketching(x,y_mat,s_k,x_axis,y_axis,title,analy_y):
    fig,ax=plt.subplots()
    n=len(y_mat)
    for i in range(n):
        plt.plot(x,y_mat[i,:],"--",label="s="+str('%.4f'%(s_k[i])))
        plt.scatter(x,y_mat[i,:])
    y_true=[]
    analy_y=np.vectorize(analy_y)
    y_true=analy_y(x)
    plt.plot(x,y_true,label="analytic values")
    plt.grid()
    plt.legend()
    ax.set_title(title)
    ax.set_xlabel(x_axis)
    ax.set_ylabel(y_axis)
    plt.plot()
    plt.show()



def linear_shooting(a,b,a1,a2,a3,a4,b1,b2,func_x,no_pt,tol=None,s_0=0,s_1=1,N_max=50):

    def phi(s):
        if a2==0:
```

```
            para=[b1/a1,s]
        else:
            deriv_s=(b1-(a1*s))/a2
            para=[s,deriv_s]
        t=np.linspace(a,b,no_pt,float)
        ans_mat=RK_fourth_vec(t,para,func_x)
        last_val=a3*ans_mat[-1,0]+a4*ans_mat[-1,1]
        return abs(b2-last_val),ans_mat,t

    if tol==None:
        tol=-999

    s_k=[]
    y_mat_s=np.zeros((53,no_pt))
    y_mat_d_s=np.zeros((53,no_pt))

    err,ans_mat,t=phi(s_0)
    s_k.append(s_0)
    y_mat_s[0,:]=ans_mat[:,0]
    y_mat_d_s[0,:]=ans_mat[:,1]

    if err<tol or N_max==1:
        y_mat_s=row_cutting(y_mat_s,len(s_k))
        y_mat_d_s=row_cutting(y_mat_d_s,len(s_k))
        return s_k,ans_mat,y_mat_s,t,y_mat_d_s

    else:
        err,ans_mat,t=phi(s_1)
        s_k.append(s_1)
        y_mat_s[1,:]=ans_mat[:,0]
        y_mat_d_s[1,:]=ans_mat[:,1]

        if err<tol or N_max==2:
            y_mat_s=row_cutting(y_mat_s,len(s_k))
            y_mat_d_s=row_cutting(y_mat_d_s,len(s_k))
            return s_k,ans_mat,y_mat_s,t,y_mat_d_s

        else:
            step=2
            while step<N_max:
                s_2 = s_0 - (s_1-s_0)*phi(s_0)[0]/( phi(s_1)[0] - phi(s_0)[0] )
                s_k.append(s_2)
                s_0 = s_1
                s_1 = s_2
                step = step + 1
                diff,ans_mat,t=phi(s_2)
                y_mat_s[step-1,:]=ans_mat[:,0]
                y_mat_d_s[step-1,:]=ans_mat[:,1]
                if diff<tol:
                    y_mat_s=row_cutting(y_mat_s,len(s_k))
                    y_mat_d_s=row_cutting(y_mat_d_s,len(s_k))
                    return s_k,ans_mat,y_mat_s,t,y_mat_d_s
    if tol!=-999:
        print("tolerance not reached")
    y_mat_s=row_cutting(y_mat_s,len(s_k))
    y_mat_d_s=row_cutting(y_mat_d_s,len(s_k))
    return s_k,ans_mat,y_mat_s,t,y_mat_d_s

def calc_rms_error(y_vec1,y_vec2):
    sum_ele=0
    for i in range(len(y_vec1)):
        ele=(y_vec1[i]-y_vec2[i])**2
        sum_ele=sum_ele+ele
    ans=sqrt(sum_ele/len(y_vec1))
    return ans
```

```
a=0
b=(np.pi)/2
#part1 Dirichlet
alpha1=1
alpha2=1
beta1=-1
alpha3=0
alpha4=1
beta2=1
s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,8,tol=10**-6,s_0=0,s_1=1)
graph_sketching(t,y_mat_s,s_k,"x","y(x)","Dirichlet Conditions:y(0)=0.333 y(1)=0.25
    ",analy_y)
graph_sketching(t,y_d_s,s_k,"x","y'(x)","Dirichlet Conditions:y(0)=0.333 y(1)=0.25"
    ,analy_deriv_y)

s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,5,10**-6)
analy_y=np.vectorize(analy_y)
corr_list=analy_y(t)
err_list=np.abs(ans_mat[:,0]-corr_list)
data_mat1=np.column_stack((t,ans_mat[:,0],corr_list,err_list))
print("Number of points=4")
pdf1=pd.DataFrame(data_mat1,columns=["x_i","y_num","y_anal","error"])
print(pdf1)

print()

s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,9,10**-6)
analy_y=np.vectorize(analy_y)
corr_list=analy_y(t)
err_list=np.abs(ans_mat[:,0]-corr_list)
data_mat1=np.column_stack((t,ans_mat[:,0],corr_list,err_list))
print("Number of points=8")
pdf1=pd.DataFrame(data_mat1,columns=["x_i","y_num","y_anal","error"])
print(pdf1)



N_list=np.logspace(1.0,6.0,num=6,base=2)
err_abs=np.zeros(len(N_list))
err_rms=np.zeros(len(N_list))
ratio_err1=np.zeros(len(N_list))
ratio_err2=np.zeros(len(N_list))
marker_list=["s","o","d","+",".","v"]

count=0
for i in N_list:
    s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,
    beta1,beta2,func_x,int(i+1),tol=None,N_max=4)
    analy_y=np.vectorize(analy_y)
    err_abs[count]=np.max(np.abs(ans_mat[:,0]-analy_y(t)))
    err_rms[count]=calc_rms_error(ans_mat[:,0],analy_y(t))
    plt.scatter(t,ans_mat[:,0],label=str(i),marker=marker_list[count])
    if count==0:
        ratio_err1[count]=None
        ratio_err2[count]=None
    else:
        ratio_err1[count]=err_abs[count-1]/err_abs[count]
        ratio_err2[count]=err_rms[count-1]/err_rms[count]
    count=count+1
plt.plot(t,analy_y(t),label="exact solution")
data_mat2=np.column_stack((N_list,err_abs,ratio_err1,err_rms,ratio_err2))
print()
pdf2=pd.DataFrame(data_mat2,columns=["N","Absolute error","error ratio","RMS error"
    ,"error ratio"])
```

```python
print(pdf2)
plt.grid()
plt.legend()
plt.show()

graph_log_sketch(N_list,err_abs)
log_x=np.log10(N_list)
log_y=np.log10(err_abs)
print("slope,intercept:",linregress(log_x,log_y)[0:2])
```

```python
import numpy as np
def euler(f, initial_cond , t):
    """
    Finds the solution of a Differential Equation using Euler Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth
    order Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond

    ele = np.array([])

    for i in range(0 , len(t)-1):
        ele = mat[i,:] + np.multiply(h, f(t[i], mat[i,:]))
        mat[i+1,:] = ele

    return mat


def RK_2(f, initial_cond ,t):
    """
    Finds the solution of a Differential Equation using RK-2 Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth
    order Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
```

```python
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond

    k1 = np.array([])
    k2 = np.array([])

    for i in range(0 , len(t)-1):
        k1 = np.multiply(h, f(t[i], mat[i,:]))
        k2 = np.multiply(h, f(t[i]+h/2, mat[i,:]+ k1/2))
        sum = np.multiply((k1+k2),1/2)

        ele = mat[i,:] + sum
        mat[i+1,:] = ele

    return mat



def RK_fourth_vec(t, initial_cond, f):
    """
    Finds the solution of a Differential Equation using RK-4 Method.

    Parameters
    ---------
    f : function
        A Python function or method for which the solution is to be found.
    initial_cond : array
        An Array of the Initial Conditions.
    t : array
        The x-axis values.

    Returns
    ---------
    mat : matrix
        Returns a matrix with the solution of each Differential Equation the nth order Differential Equation was broken into.
    """
    h = t[1] - t[0]

    mat = np.array([],[])
    mat = np.zeros([len(t), len(initial_cond)])

    mat[0,:] = initial_cond

    k1 = np.array([])
    k2 = np.array([])
    k3 = np.array([])
    k4 = np.array([])
    ele = np.array([])

    for i in range(0 , len(t)-1):

        k1 = f(t[i], mat[i,:])
        k2 = f(t[i]+(h/2),(mat[i,:]+np.multiply(k1, (h/2))))
        k3 = f(t[i]+(h/2),(mat[i,:]+np.multiply(k2, (h/2))))
        k4 = f(t[i]+(h/1),(mat[i,:]+np.multiply(k3, (h/1))))
        sum = np.multiply((k1+np.multiply(k2,2)+np.multiply(k3,2)+k4), (1/6))

        ele = mat[i,:] + np.multiply((sum), h)
        mat[i+1,:] = ele

    return mat
```