Boundary Value Problem:Linear Shooting

Prateek Bhardwaj
(2020PHY1110)
(20068567042)

Monu Chaurasiya
(2020PHY1102)
(20068567035)

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

Practical Report Submitted to

Dr. Mamta

"32221401 - Mathematical Physics III"

Table of Contents

1 Theory 1

2 Results and Discussion 6

A Program 8

1 Theory

(a) What is the difference between an initial value and boundary value problem? What is a two point 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 $D(x)y(x) = \lambda y(x)$ where $D(x)$ is the differential operator , λ is any constant. 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 $x = x_1$, then the problem becomes an initial value problem.But if n_1 conditions are given at the initial point $x_1, y_1(x_1), ..., y_n(x_1)$ and that $n_2(n-n_1)$ are given at the final point $x_2, y_1(x_2)$ then it is a Two point boundary value problem.

(b)

1. Write down the general form of the second order boundary value problem (BVP).

General Form:

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

Where,

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

2. Discuss the three types of Boundary conditions.

- 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.

3. **What are homogeneous and non-homogeneous BVP?**

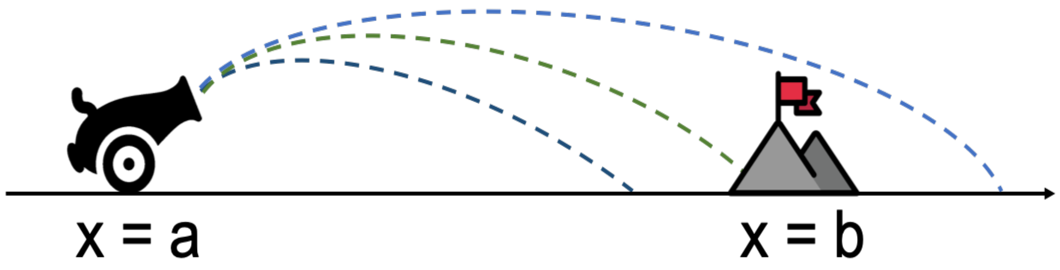
Suppose we have second order differential equation :

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

A B.V.P is homogenous if in addition to g(x)=0 , we have $y_0 = 0$ and $y_1 = 0$ where $y_0 = 0$ and $y_1 = 0$ are the values of the function at the boundary points.
If a B.V.P is not homogenous it is known as non-homogenous B.V.P .

4. **Explain the concept of Shooting method.**

The shooting methods are developed with the goal of transforming the ODE boundary value problems to an equivalent initial value problems, then we can solve it using the methods we learned from the previous chapter. In the initial value problems, we can start at the initial value and march forward to get the solution. But this method is not working for the boundary value problems, because there are not enough initial value conditions to solve the ODE to get a unique solution. Therefore, the shooting methods was developed to overcome this difficulty.



The name of the shooting method is derived from analogy with the target shooting: as shown in the above figure, 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. Let's see how the shooting methods works using the second-order ODE given $f(a) = f_a$ and $f(b) = f_b$.

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

- 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.
- Using what we learned from previous chapter, i.e. we can use Runge-Kutta method, to integrate 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 $g(\alpha) - f_b = 0$.

5. Write down the general form of the second order boundary value problem (BVP).

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

With robin boundary conditions:

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

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

Discuss the Neumann and Dirichlet conditions.

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

  It becomes Dirchilet Boundary Conditions.

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

With $y(a) = \alpha$ and $y(b) = \beta$

To solve it as initial value problem, we obtain y' at $x = a$, So make an assumption for it.

Let $y'(a) = z$,

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

With $y(a) = \alpha$ and $y'(a) = z$,now we have an I.V.P .

Let $y(x, z)$ be its solution

The solution to this problem satisy,

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

If $\phi(z) = y(b,z) - \beta$
The problem reduces to finding $z = z^*$ such that $\phi(z^*) = 0$
Thus, we solve that BVP by using the solution of a sequence of IVP's involving parameter 'z'.
We take $z = z_k$ such that:

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

Where $y(x,z_k)$ denotes the solution of the IVP with $z = z_k$, while $y(x)$ denotes the solution of the BVP.
We choose the values of the x_k until $y(b,z_k)$ is suefficiently close to β.

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

This can be solved using Newton Raphson or Secant Method.

- Case 2: If $\alpha_1 = \beta_1 = 0$ i:e $y'(a) = \alpha$ and $y'(b) = \beta$
  It becomes Neumann Boundary Conditions.

$$y'(\alpha) = \alpha \;\; y'(b) = \beta$$

Now $y(a)$ is approximated and then improved in each iteration.
We use the initial conditions:

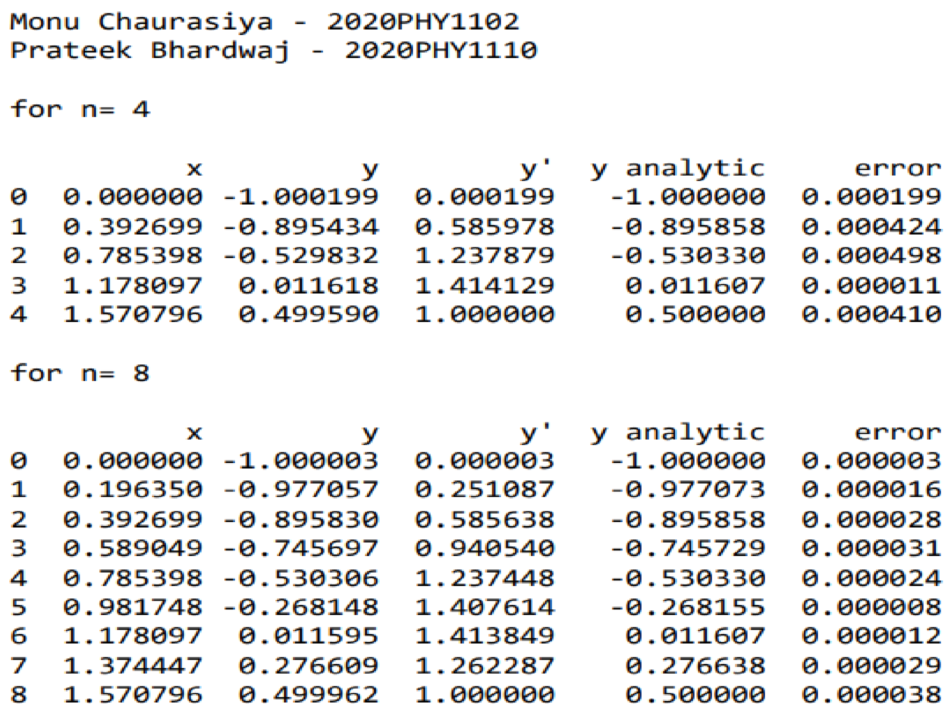
$$y(a) = z \;\; y'(a) = \alpha$$

Where z is chosen s.t

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

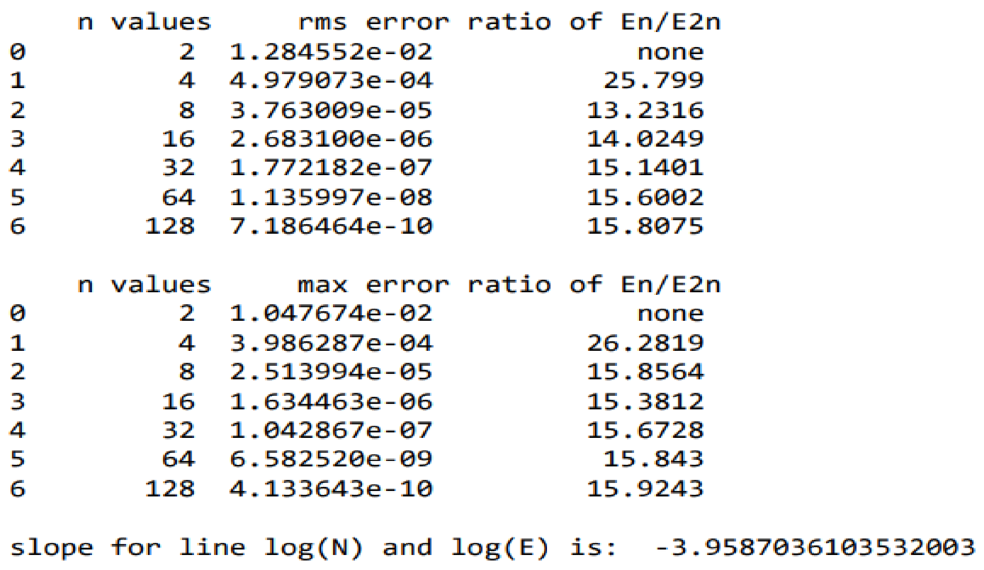
Where $y(x)$ is the BVP and $y(x,z)$ is the solution of IVP with $y(a) = z$.

2 Results and Discussion



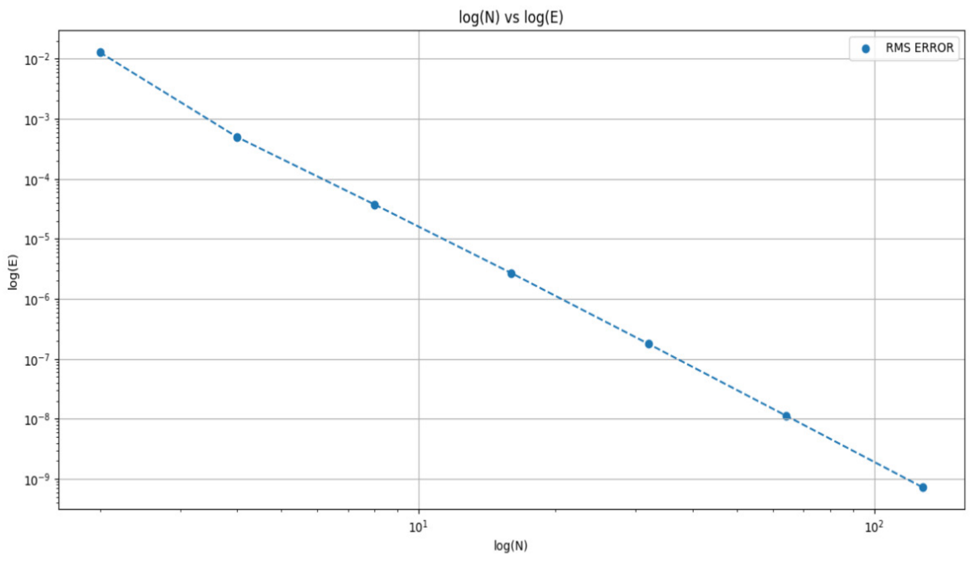
```
Monu Chaurasiya - 2020PHY1102
Prateek Bhardwaj - 2020PHY1110

for n= 4

          x         y        y'  y analytic     error
0  0.000000 -1.000199  0.000199   -1.000000  0.000199
1  0.392699 -0.895434  0.585978   -0.895858  0.000424
2  0.785398 -0.529832  1.237879   -0.530330  0.000498
3  1.178097  0.011618  1.414129    0.011607  0.000011
4  1.570796  0.499590  1.000000    0.500000  0.000410

for n= 8

          x         y        y'  y analytic     error
0  0.000000 -1.000003  0.000003   -1.000000  0.000003
1  0.196350 -0.977057  0.251087   -0.977073  0.000016
2  0.392699 -0.895830  0.585638   -0.895858  0.000028
3  0.589049 -0.745697  0.940540   -0.745729  0.000031
4  0.785398 -0.530306  1.237448   -0.530330  0.000024
5  0.981748 -0.268148  1.407614   -0.268155  0.000008
6  1.178097  0.011595  1.413849    0.011607  0.000012
7  1.374447  0.276609  1.262287    0.276638  0.000029
8  1.570796  0.499962  1.000000    0.500000  0.000038
```

Table shows the value y and y' for n=4 and n=8 for x =[0,$\frac{\pi}{2}$] with error in the values calculated.

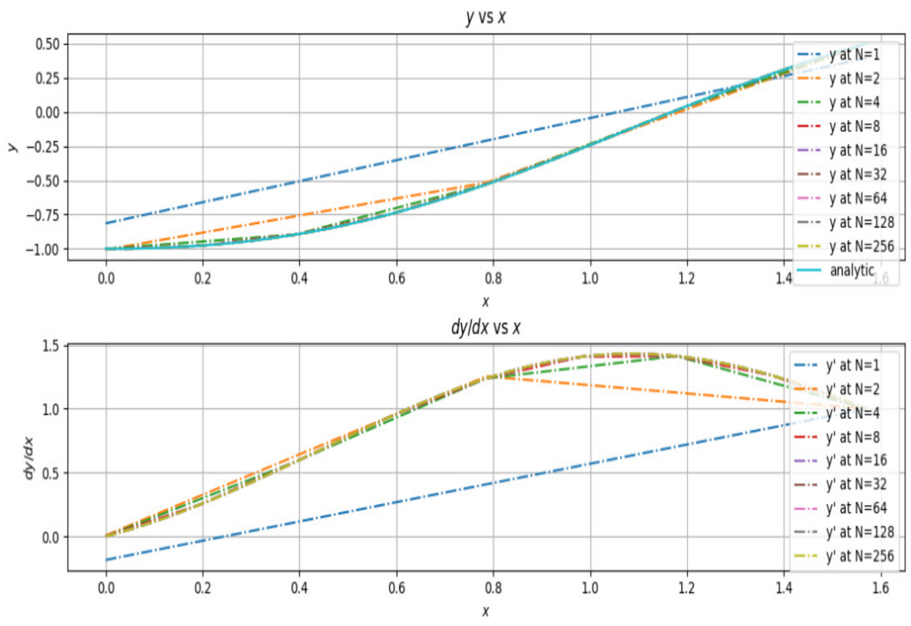


```
   n values     rms error ratio of En/E2n
0         2  1.284552e-02            none
1         4  4.979073e-04          25.799
2         8  3.763009e-05         13.2316
3        16  2.683100e-06         14.0249
4        32  1.772182e-07         15.1401
5        64  1.135997e-08         15.6002
6       128  7.186464e-10         15.8075

   n values     max error ratio of En/E2n
0         2  1.047674e-02            none
1         4  3.986287e-04         26.2819
2         8  2.513994e-05         15.8564
3        16  1.634463e-06         15.3812
4        32  1.042867e-07         15.6728
5        64  6.582520e-09          15.843
6       128  4.133643e-10         15.9243

slope for line log(N) and log(E) is:  -3.9587036103532003
```

Table shows the error-rms and error-max and the slope of line log(N) and log(E)



This is the plot of log(N) vs log(E) where E is the rms error . From the plot we can see that as n increases the error decrease.



This is the plot of y vs x and y’ vs x by shooting method for different N(no.of steps) and we can see that as N increases the the line overlap with the analytical solution.

A Program

```python
'''
Monu Chaurasiya - 2020PHY1102
Prateek Bhardwaj - 2020PHY1110
'''

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from MyIVP import RK4
from scipy import stats

print("Monu Chaurasiya - 2020PHY1102")
print("Prateek Bhardwaj - 2020PHY1110")
def Func(x,var):
    y,z=var
    f1=z
    f2=np.sin(3*x) - y
    t=np.array([f1,f2])
    return t

def secant(x0,x1,P0,P1):
    x2 = x1 - ((x1-x0)/(P1-P0))*P1
    return x2

def Lin_shooting(Func,a,b,N,x0,x1,tol,nmax):

    IC_1=np.array([-1-x0,x0],dtype=float)
    s1=RK4(Func,IC_1,a,b,N)
    y1,y1_der=s1[0].T

    IC_2=np.array([-1-x1,x1],dtype=float)
    s2=RK4(Func,IC_2,a,b,N)
    y2,y2_der=s2[0].T

    yb_der=1           #given boundary condition for y'(pi/2)=1

    P0=1-y1_der[-1]    #comparing the boundary values with the values
    generated by RK4 for initial guess condition.
    P1=1-y2_der[-1]
    l1=[];l2=[];l3=[]
    for i in range(nmax):
        if abs(P0) < tol:
            break
        else:
            new_guess=secant(x0,x1,P0,P1)

            x0=x1
            x1=new_guess
            IC_3=np.array([-1-x1,x1],dtype=float)

            z=RK4(Func,IC_3,a,b,N)
            l1,l2=z[0].T
            l3=z[1]

            y_der_new=yb_der - l2[-1]
            P0=P1
            P1=y_der_new

    return l1,l2,l3

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

def table(Func,a,b,N,x0,x1,tol,nmax):
    y,y_der,x = Lin_shooting(Func,0,np.pi/2,N,x0, x1, tol, nmax)
    print()
    print("for n=",N,"\n")
    lis1 = []
    for i in x:
        lis1.append(analytic(i))
    lis2 = []
    for i in range(len(x)):
        lis2.append(abs(y[i] - lis1[i]))
    data1 = {"x": x, "y": y, "y'": y_der, "y analytic": lis1, "error":
    lis2}
    print(pd.DataFrame(data1))

x0=1
x1=0
tol=10**(-10)
nmax=100

table(Func,0,np.pi/2,4,x0,x1,tol,nmax)
table(Func,0,np.pi/2,8,x0,x1,tol,nmax)



f1=lambda x: 3/8*np.sin(x)- np.cos(x) - 1/8*np.sin(3*x)
x_t=np.linspace(0,np.pi/2,100)

fig=plt.figure()
(ax1),(ax2) = fig.subplots(2,1)
for i in range(0,9):
    N=2**i
    arr1,arr2,arr3=Lin_shooting(Func,0,np.pi/2,N,x0,x1,tol,100)
    ax1.plot(arr3,arr1,label="y at N="+str(N),ls="dashdot")
    ax2.plot(arr3,arr2,label="y' at N="+str(N),ls="dashdot")

ax1.plot(x_t,f1(x_t),label="analytic")
ax2.set_title("$dy/dx$ vs $x$")
ax2.set_xlabel("$x$")
ax2.set_ylabel("$dy/dx$")
ax2.legend()
ax2.grid()
ax1.set_title("$y$ vs $x$")
ax1.set_xlabel("$x$")
```

```python
ax1.set_ylabel("$y$")
ax1.legend()
ax1.grid()
plt.tight_layout()
plt.show()


def error(a):
    y_arr,y_Der_arr,x_arr=Lin_shooting(Func,0,np.pi/2,a,0,1,tol,nmax)
    y_analytic=f1(x_arr)
    err_rms=0
    err_max=0
    for j in range(len(x_arr)):
        err_rms += (y_arr[j] - y_analytic[j])**2
    err_rms=np.sqrt((err_rms/a))
    err_max=max(abs(y_arr-y_analytic))

    return x_arr,y_arr,y_analytic,err_rms,err_max

N_val=[]
for k in range(1,8):
    N=2**k
    N_val.append(N)
print()

l11,l22,l33,l44,l55=[],[],[],[],[]
ratio_rms=["none"];ratio_max=["none"]

for i in N_val:
    t1,t2,t3,t4,t5=error(i)
    l11.append(t1),l22.append(t2),l33.append(t3),l44.append(t4),l55.
   append(t5)
    if i!=2:
        ratio_rms.append(l55[-2]/l55[-1])
        ratio_max.append(l44[-2]/l44[-1])

data4={"n values":N_val,'rms error':l55,'ratio of En/E2n':ratio_rms}
print("\n",pd.DataFrame(data4))

data5={"n values":N_val,'max error':l44,'ratio of En/E2n':ratio_max}
print("\n",pd.DataFrame(data5))


fig=plt.figure()
ax1=fig.subplots(1,1)
ax1.scatter(N_val,l55,label="RMS ERROR")
ax1.plot(N_val,l55,ls="--")
ax1.set_xscale("log")
ax1.set_yscale("log")
ax1.set_title("log(N) vs log(E)")
ax1.set_xlabel("log(N)")
ax1.set_ylabel("log(E)")
ax1.legend()
ax1.grid()
plt.tight_layout()
plt.show()

slope=stats.linregress(np.log(N_val),np.log(155))
print("\nslope for line log(N) and log(E) is: ",slope[0])
```