Assignment 10 - Non 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

Date / / Page no.

Theory:

Shooting method to solve the Non-linear two-point BVP of the form:

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

with the robin boundary conditions:

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

- If $\alpha_2 = \beta_2 = 0$

i.e $y(a) = \alpha_3$ and $y(b) = \beta_3$

→ Dirichlet Boundary conditions

- If $\alpha_1 = \beta_1 = 0$

$y'(a) = \alpha_3$ and $y'(b) = \beta_3$

→ Neumann Boundary conditions

Date / /
Page no.

In the case of Dirichlet Boundary conditions:
The solution to the BVP is approximated by using the solutions to a sequence of initial value problems involving a parameter S having the form,

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

with $y(a) = \alpha$ and $y'(a) = S$
↓
(Guess).

This converts our problem into an IVP.
Let the solⁿ to this IVP be: $y(x, s)$.

The solⁿ of this problem should satisfy:
$$y(b, s) = \beta.$$

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

Thus the problem reduces to finding $S = S^*$ such that $\phi(S^* = 0)$ which is in general a non-linear equation which may be solved iteratively by Newton-Raphson or secant method.

Iteratively new values of S are taken, each which involves solving one IVP also,

Date / / Page no.

Thus, we approximate the soln to the BVP by using the soln to a sequence of IVPs involving a parameter 's'.

This is done by choosing the parameters $S = S_k$ in a manner to ensure that.

$$\lim_{k \to \infty} y(b, S_k) = y(b) = \beta$$

where $y(x, S_k)$ denotes the soln to the IVP with $S = S_k$

while $y(x)$ denotes the soln to the BVP.

If $y(b, S_0)$ is not sufficiently close to β, the approximation is corrected by choosing $S = S_1, S_2 ..$ and so on until $y(b, S_k)$ is sufficiently close to "hitting" β.

$$y(b, S_k) - \beta = 0.$$

This is a non-linear equation that is solved by the Newton-Raphson or secant method.

DELTA My NoteBook

Date / / Page no.

In the case of <u>Neumann Boundary Conditions</u> :

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

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

Use the initial conditions
$$y(a) = S \quad \text{and} \quad y'(a) = \alpha$$

S is chosen such that,

$$\phi(S) = y'(b,S) - y'(b)$$
$$= y'(b,S) - \beta$$
$$= 0$$

Where, $y(x) \to$ soln of BVP
and $y(x,S) \to$ soln of IVP with $y(a) = S$.

Date / / Page no.

In the case of Robin Boundary Conditions:

Let us take an example where:

$$y(a) = \alpha$$

$$\beta_1 \, y(b) + \beta_2 \, y'(b) = \beta$$, is given.

we have to guess $y'(a) = s$
for this to become an initial value problem.

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 values of s iteratively such that
$\phi(s)$ approaches 0.

{RK4 used for solving the IVP}

Similarly, it can be done for the different cases.

DELTA My NoteBook

Scanned with CamScanner

Date / /

Page no.

Note : Secant Method

Here we need two initial approximations, S_0 and S_1

and then the remaining terms of the sequence are generated by :

$$S_k = S_{k-1} - \frac{(S_{k-1} - S_{k-2})\, \phi(S_{k-1})}{\phi(S_{k-1}) - \phi(S_{k-2})}$$

$$k = 2, 3 \ldots\ldots$$

The ZVP is initially solved for two values S_0 and S_1.

The iteration is stopped when $|\phi(S_k)| < \text{tolerance}$.

Programming

```
import numpy as np
import matplotlib.pyplot as plt
from IVP_Module import *

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

def analy_deriv_y(x):
    return -1/((x+3)**2)


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_4(func_x,para,t)
        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((N_max,no_pt))
    y_mat_d_s=np.zeros((N_max,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


a=0
b=1
#part1 Dirichlet
alpha1=1
alpha2=0
beta1=1/3
alpha3=1
alpha4=0
beta2=1/4
s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,8,10**-6)
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)


#part-2 Neumann Conditions
alpha1=0
alpha2=1
beta1=-1/9
alpha3=0
alpha4=1
```

```
beta2=-1/16
s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,8,10**-6,s_0=0.6,s_1=0.8)
graph_sketching(t,y_mat_s,s_k,"x","y(x)","Neumann Conditions:y'(0)=0.1111,y'(1)
    =-1/16",analy_y)
graph_sketching(t,y_d_s,s_k,"x","y'(x)","Neumann Conditions:y'(0)=0.1111,y'(1)
    =-1/16",analy_deriv_y)


#part-3 mixed with robin
alpha1=3
alpha2=-9
beta1=2
alpha3=1
alpha4=0
beta2=1/4
s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,8,10**-6)
graph_sketching(t,y_mat_s,s_k,"x","y(x)","robin a part",analy_y)
graph_sketching(t,y_d_s,s_k,"x","y'(x)","robin a part",analy_deriv_y)

#part-4 mixed with robin
alpha1=1
alpha2=0
beta1=1/3
alpha3=2
alpha4=2
beta2=3/48
s_k,ans_mat,y_mat_s,t,y_d_s=linear_shooting(a,b,alpha1,alpha2,alpha3,alpha4,beta1,
    beta2,func_x,8,10**-6)
graph_sketching(t,y_mat_s,s_k,"x","y(x)","robin b part",analy_y)
graph_sketching(t,y_d_s,s_k,"x","y'(x)","robin b part",analy_deriv_y)
```

Discussion

Dirichlet Conditions

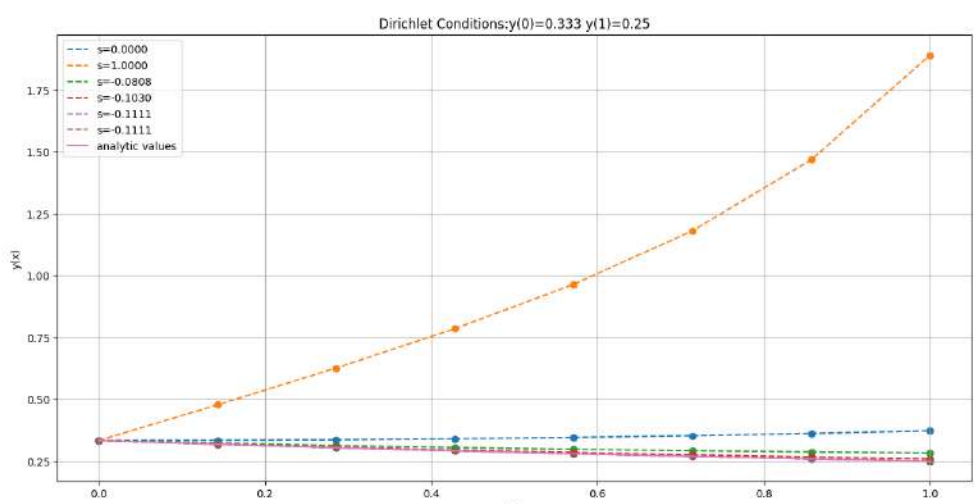

Figure 1: x vs y

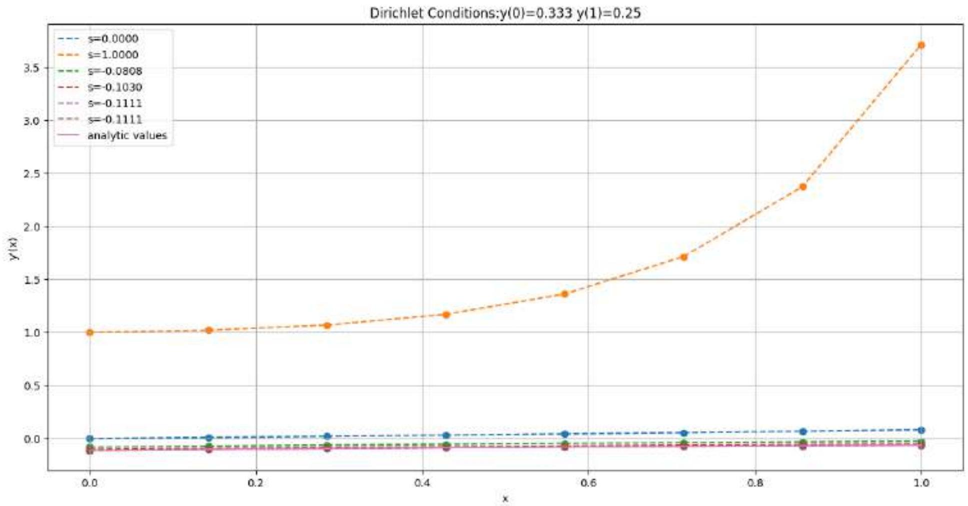

Figure 2: x vs y'

Neumann Conditions

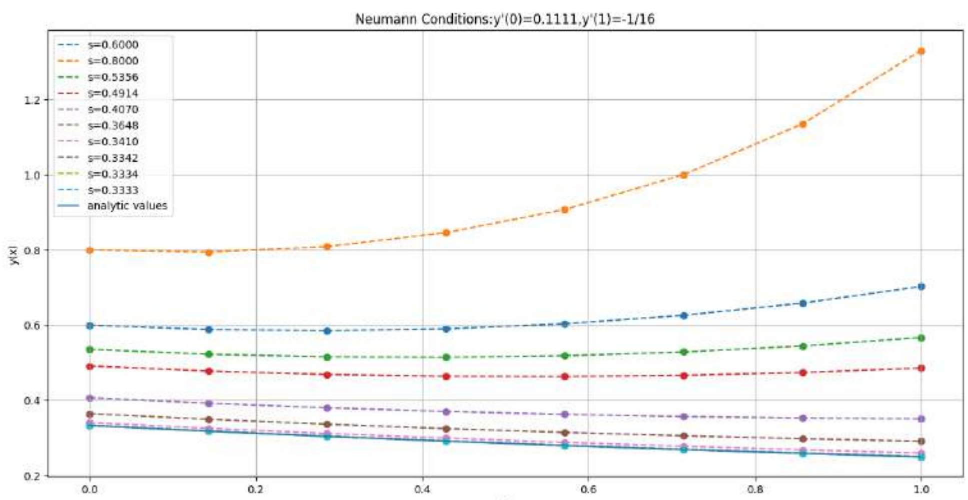

Figure 3: x vs y

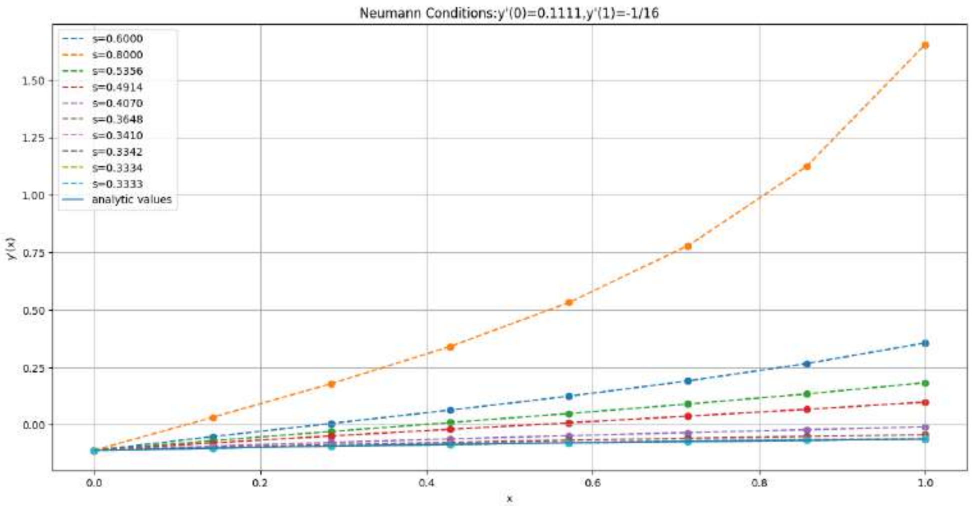

Figure 4: x vs y'

Robin Conditions 1

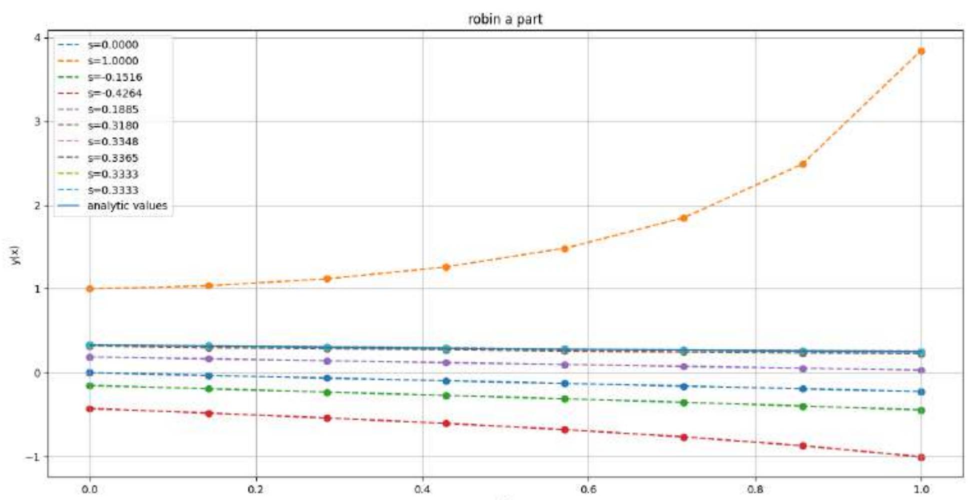

Figure 5: x vs y

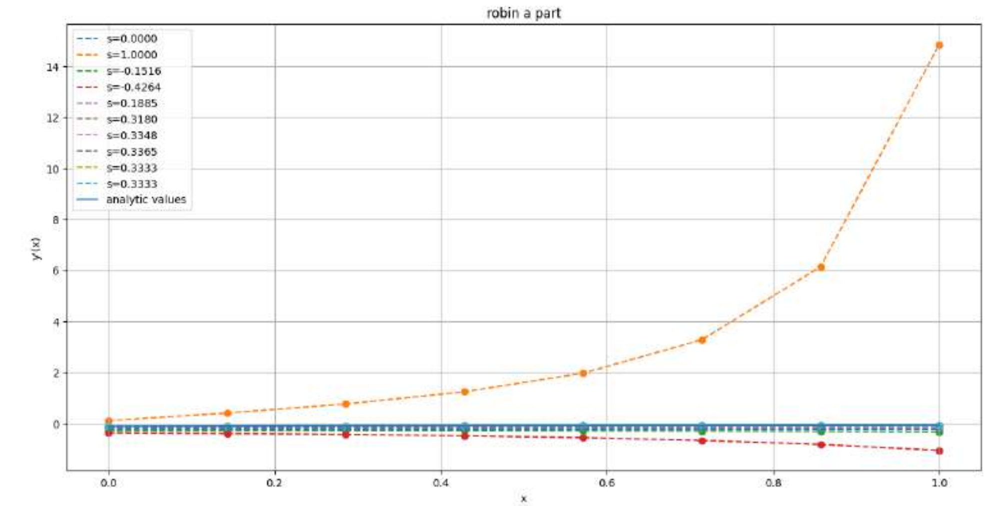

Figure 6: x vs y'

Robin Conditions 2

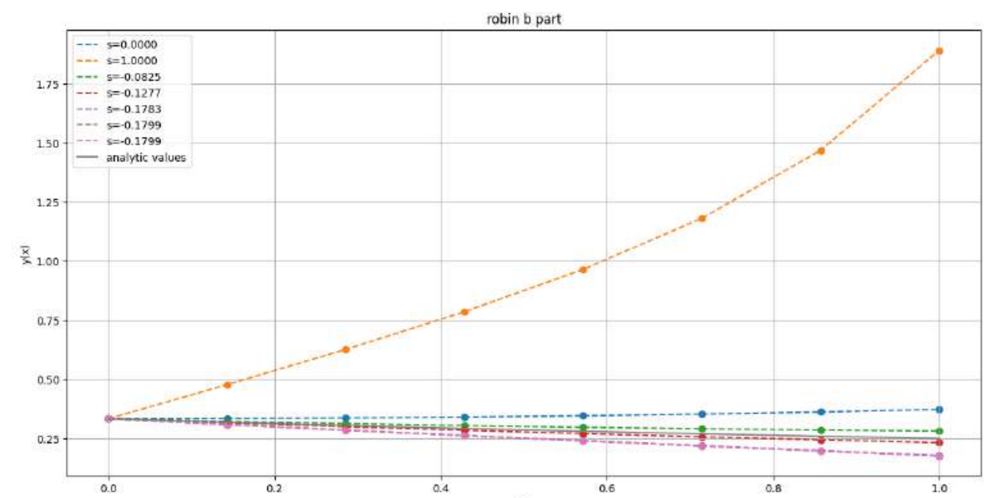

Figure 7: x vs y

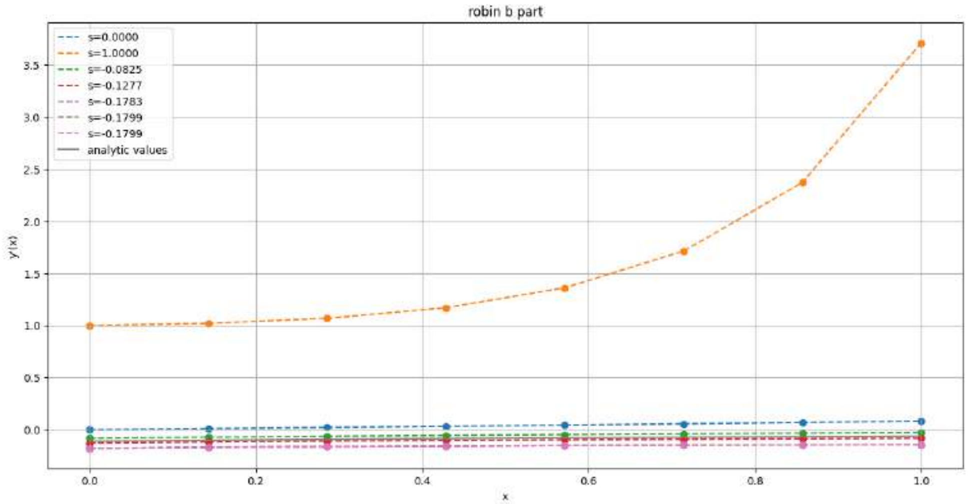

Figure 8: x vs y'

The solution of the BVP was reached in a few iterations within the given tolerance of 10^{-6}.