Shooting Method for BVPs

Author Name

Devika Santhosh (MSC21308)
Siyad Rahman.M (MSC21319)

Faculty Name

Dr. Dond Asha Kisan

School of Mathematics
IISER Thiruvananthapuram

November 19, 2022

Contents

1 Introduction

We have already seen several numerical schemes for solving different IVPs and BVPs(in particular 2 point BVPs).

Here,we will see another method called **shooting method** which is used to solve a 2 point Boundary Value Problems in an Ordinary Differential Equation. The basic idea of shooting method is to convert the given BVP into an equivalent IVP and then solve it using any of the method for IVPs.

In this project we will see a general shooting method which works for both linear and non-linear problems and the **Linear shooting method**.Then,we will discuss the convergence of shooting method. We will also provide the implemention using Matlab.

2 Shooting Method

2.1 Theory

Consider given BVP as:

$$y'' = f(x, y, y'), \quad a \leq x \leq b \quad \text{with} \quad y(a) = \alpha \quad \text{and} \quad y(b) = \beta \tag{1}$$

We convert BVP to system of first order IVPs by assuming y'=z and we get IVP:

$$y' = z, \quad z' = f(x, y, z), \quad y(a) = \alpha \tag{2}$$

Inorder to solve this IVP, we need one more initial condition at x=a. For that we will assume $z(a) = m_1$, where m_1 is an arbitrary constant,which is also represent the slope at x=a. Then, we solve IVP and estimate y(x) at x=b. If the estimated value, say β_1 and $\beta_1 = \beta$,then we will stop. Else, we contnue to solve IVP using another guess say $z(a) = m_2$. Again we check equality of β_2(the numerical solution at x=b with initial guess m_2) with β and repeat the procedure with computing the improved initial guess using some linear interpolation or extrapolation techniques so that $z(a) = m_3$ and y(b)=β. Here,

$$m_3 = m_2 + \frac{\beta - \beta 2}{\beta_2 - \beta 1}(m_2 - m_1) \tag{3}$$

Now, we repeat the process until a specific accuracy in final function value is obtained. If we are using Newton's method, the iterations for m is given by, Here,

$$m_{k+1} = m_k - \frac{y(b, m_k) - \beta}{z(b, m_k)} \tag{4}$$

2.2 Convergence of Shooting Method:

Theorem 1: Suppose that a numerical algorithm for the solution of the system of differential equations (2) gives the result $y_{i,j}(m)$, the numerical approximations to $y_i(x_j; m), i = 1, 2$ and $j = 1, 2, ..., n$, where the error satisfies

$$\max_{1 \leq j \leq n} |y_i(x_j; m) - y_{i,j}(m)| \leq C(m)h^s, \quad i = 1, 2, \tag{5}$$

for some $s \geq 0$; here $C(m)$ depends on bounds on the derivatives of y and $f(x, y)$, and on m. Suppose also that the Newton iteration is performed until

$$|y_{1,n}(m_k) - \beta| \leq \epsilon. \tag{6}$$

Then, $y_{1,j}(m_k)$ is an approximation to the solution of the boundary value problem which satisfies

$$\max_{1 \leq j \leq n} |y(x_j) - y_{1,j}(m_k)| \leq 2C(m_k)h^s + \epsilon. \tag{7}$$

2.3 MATLAB Implementation

2.3.1 Matlab Code

```
% Shooting Method
% BVP: y"=f(x,y,y'), y(1)=a,y(end)=b
clear all
close all
clc
% Define f,g
f=@(x,y,z) z;
g=@(x,y,z) (y.^2)+(6.*x)-(x.^6);
exactsol=@(x) x.^3; % exact solution
format long
a=0;b=1; % boundary conditions
tol=10^(-5);% tolerance
% loop for different mesh size
lel=1;
for lel=1:5
    N=5*lel;
x=linspace(0,1,N+1);
h=x(2)-x(1);
H(lel)=h;

% Initialize y,z
y=zeros(1,length(x));
z=zeros(1,length(x));
m1=1; % initialize m
u=RK2(a,N,f,g,x,y,z,m1);
i=1;
t=u(end)-b;
m2=m1-t/u(end);
while (abs(m2-m1)>tol)
    m1=m2;
    u=RK2(a,N,f,g,x,y,z,m1);
    t=u(end)-b;
    m2=m1-t/u(end);
    i=i+1;
    if i>100
        error("Not converge")
    end
end
[x',u']
hold on
plot(x,u,'-*'); % plot num sol
err(lel)=max(abs(u-exactsol(x))); % error
lel=lel+1;
end
error=err'
plot(x,exactsol(x),'-o'); % plot exact sol
hold off
%convergence rate
for k=1:lel-2
cgsrate(k)=log(err(k)/err(k+1))/log(H(k)/H(k+1));
end
```

```
convergence_rate=cgsrate'


% function to solve IVP by RK2 scheme with different m
function[u]=RK2(a,N,f,g,x,y,z,m)
x=linspace(0,1,N+1);
h=x(2)-x(1);
%Initialize y,z
y=zeros(1,length(x));
z=zeros(1,length(x));
%Boundary condition
y(1)=a;
%y(length(x))=b;
%initial condition
y(1)=a; z(1)=m;
%solve IVP by RK2 scheme
for i=2:length(x)
     k11=h*f(x(i-1),y(i-1),z(i-1));
     k21=h*g(x(i-1),y(i-1),z(i-1));
     k12=h*f(x(i),y(i-1)+k11,z(i-1)+k21);
     k22=h*g(x(i),y(i-1)+k11,z(i-1)+k21);

     y(i)=y(i-1)+(1/2)*(k11+k12);
     z(i)=z(i-1)+(1/2)*(k21+k22);
end
u=y;
end
```

2.3.2 Results:

```
% sample sol u for N=10
    ans =

                   0                   0
   0.100000000000000   0.001076503408833
   0.200000000000000   0.008153008406263
   0.300000000000000   0.027229338528859
   0.400000000000000   0.064304339257288
   0.500000000000000   0.125374348041266
   0.600000000000000   0.216430968953595
   0.700000000000000   0.343458112919827
   0.800000000000000   0.512428196146352
   0.900000000000000   0.729297253286655
   1.000000000000000   0.999998478453557

% error for different N
   error =

   0.001285033381414
   0.000458112919827
   0.000218011319048
   0.000127044515902
   0.000082240633254
```

```
convergence_rate =

    1.488030681084742
    1.831399804903395
    1.877104941310199
    1.948916021280892
```

2.3.3 Plot:

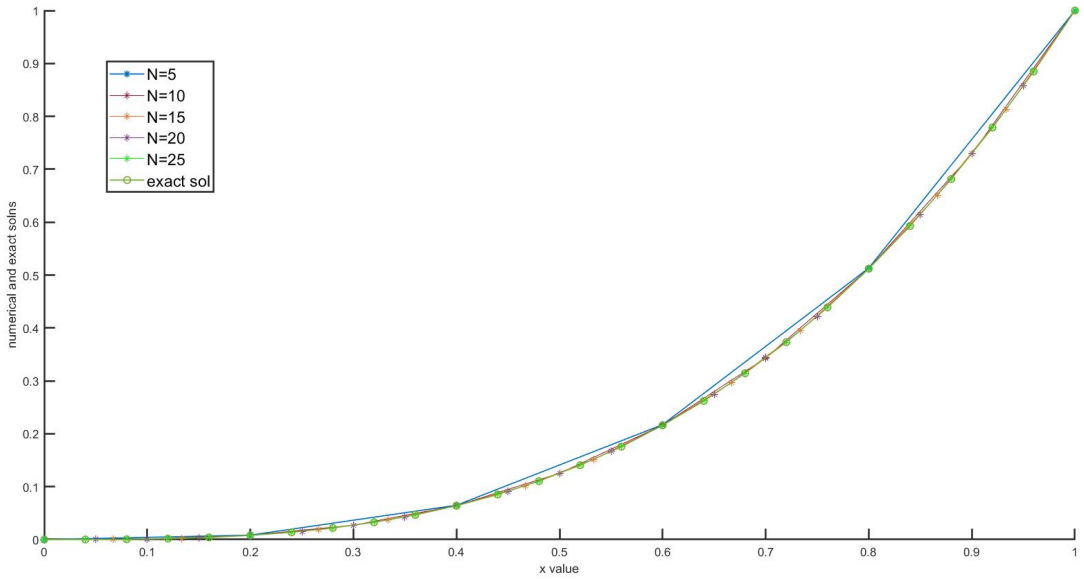

Figure 1: Plot of exact solution and numerical solution for different N

2.3.4 Observations:

We found error reduces as N increases(mesh size h decreases). Since, we used RK2 Method to solve IVP, we got the numerical convergence rate tending to 2, which was our theoretical expectation by the Theorem of Convergence for Shooting method. We have also plotted the numerical solutions for different N value and the exact solution. The convergence of scheme is also seen from the plot.

3 Linear Shooting Method

3.1 Theory

Here we will discuss the **Linear Shooting Method**, which is a shooting method explicitly used for soving linear BVPs. The basic technique of this method is the reduction of given two-point linear BVP to a two-point IVP.
Consider the given two-point BVP:

$$y'' = p(x)y' + q(x)y + r(x), \quad y(a) = \alpha, \quad y(b) = \beta \tag{8}$$

We solve this BVP by solving 2 closely related IVPs as follows. Let u(x) satisfy the non-homogeneous differential equation:

$$u'' = p(x)u' + q(x)u + r(x), \quad u(a) = \alpha, \quad u'(a) = 0 \tag{9}$$

and v(x) satisfies the homogeneous equation:

$$v'' = p(x)v' + q(x)v, \quad v(a) = 0, \quad v'(a) = 1 \tag{10}$$

Then, the linear combination $y(x) = u(x) + \lambda v(x)$, where λ is independent of x is a solution to eqn(7) as follows:

$$\begin{aligned} y'' &= u'' + \lambda v'' \\ &= p(x)u' + q(x)u + r(x) + \lambda(p(x)v' + q(x)v) \\ &= p(x)(u' + \lambda v') + q(x)(u + \lambda v) + r(x) \\ &= p(x)y' + q(x)y + r(x) \end{aligned}$$

Again $y(a) = u(a) + \lambda v(a) = \alpha + \lambda \times 0 = \alpha$ and $y(b) = u(b) + \lambda v(b)$. Now, select λ such that $y(b) = \beta$. Therefore,

$$\lambda = \frac{\beta - u(b)}{v(b)}, \quad \text{provided} \quad v(b) \neq 0.$$

Hence, the unique solution of BVP in eqn(7) is

$$y(x) = u(x) + \frac{\beta - u(b)}{v(b)} v(x) \tag{11}$$

3.2 MATLAB Implementation

3.2.1 Matlab Code

```
%Linear Shooting Method
%BVP: y"=p(x)y'+q(x)y+r(x), y(1)=c,y(end)=d
clear all
close all
clc

exactsol=@(x) x+sin(2*pi.*x);% exact solution
% loop for different mesh size
```

```
lel=1;
for lel=1:5
    N=5*lel;
x=linspace(0,1,N+1);
h=x(2)-x(1);
H(lel)=h;
%Initialize y,z
y=zeros(1,length(x));
z=zeros(1,length(x));
% IVP 1
a=0; b=0; % initial condition
g=@(x,y,z) (-x).*z+(x.^2).*y-(4*pi*pi*sin(2*pi.*x))+x+...
    (2*pi.*x.*cos(2*pi.*x))-(x.^3)-(x.*x.*sin(2*pi.*x));
u=RK4(a,b,N,g,x,y,z);
% IVP 2
a=0; b=1; % initial condition
g=@(x,y,z) (-x).*z+(x.^2).*y;
v=RK4(a,b,N,g,x,y,z);
d=1; % boundary condition
% loop for finding soln of BVP
for i=1:length(x)
    sol(i)=u(i)+(((d-u(end))/v(end))*v(i));
end
solution=sol'
hold on
plot(x,sol,'-*');% plot num sol
err(lel)=max(abs(sol-exactsol(x)));% error
lel=lel+1;
end
error=err'
plot(x,exactsol(x),'-o');% plot exact sol
hold off
%convergence rate
for k=1:lel-2
cgsrate(k)=log(err(k)/err(k+1))/log(H(k)/H(k+1));
end
convergence_rate=cgsrate'

% function to solve IVP by RK4 scheme
function[u]=RK4(a,b,N,g,x,y,z)
f=@(x,y,z) z;
x=linspace(0,1,N+1);
h=x(2)-x(1);
% initialize y,z
y=zeros(1,length(x));
z=zeros(1,length(x));
y(1)=a; z(1)=b;% initial condition
% RK4 scheme
for i=2:length(x)
    K1=f(x(i-1),y(i-1),z(i-1));
    R1=g(x(i-1),y(i-1),z(i-1));
    K2=f(x(i-1)+(h/2),y(i-1)+h*(K1)/2,z(i-1)+h*(R1)/2);
    R2=g(x(i-1)+(h/2),y(i-1)+h*(K1)/2,z(i-1)+h*(R1)/2);
    K3=f(x(i-1)+(h/2),y(i-1)+h*(K2)/2,z(i-1)+h*(R2)/2);
    R3=g(x(i-1)+(h/2),y(i-1)+h*(K2)/2,z(i-1)+h*(R2)/2);
    K4=f(x(i),y(i-1)+K3*h,z(i-1)+h*(R3));
```

```
    R4=g(x(i),y(i-1)+K3*h,z(i-1)+h*(R3));
    y(i)=y(i-1)+(h/6)*(K1+2*K2+2*K3+K4);
    z(i)=z(i-1)+(h/6)*(R1+2*R2+2*R3+R4);

end
u=y;
end
```

3.2.2 Results:

```
    % sample sol for N=10

    solution =

                    0
   0.687672014044071
   1.150864454379402
   1.250843167063663
   0.987610420765326
   0.499905061772159
   0.012209917541576
  -0.250992895582489
  -0.150967800877438
   0.312279283549215
   1.000000000000000

% error for different N values
error =

   0.003207832428086
   0.000213349231490
   0.000041646504271
   0.000013098221811
   0.000005374243446


convergence_rate =

   3.910309960835636
   4.029231963067499
   4.020900373716418
   3.992311972535564
```

3.2.3 Plot:

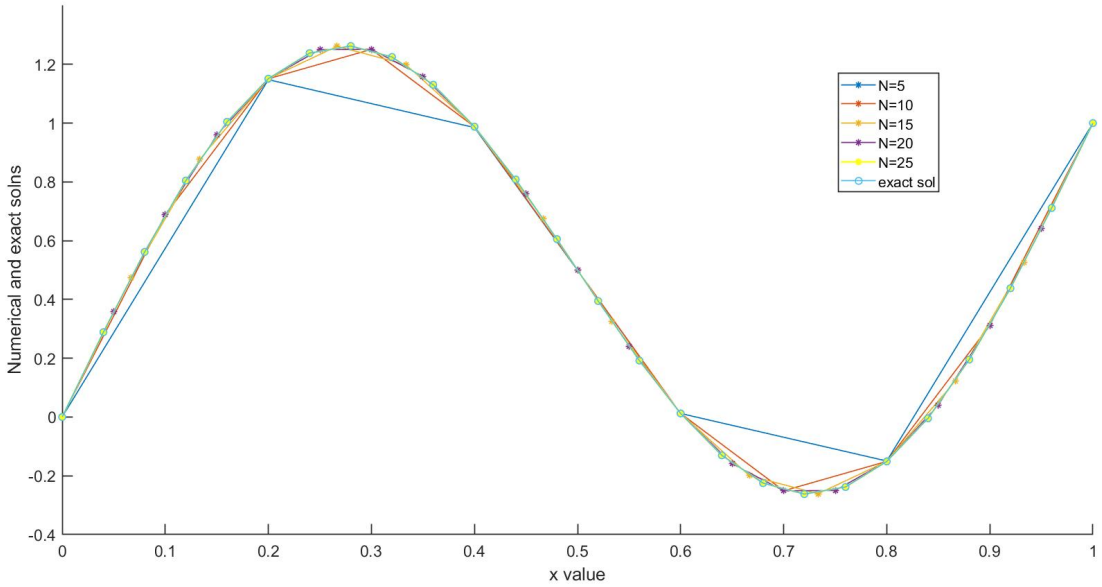

Figure 2: Plot of exact solution and numerical solution for different N

3.2.4 Observations:

We found error reduces as N increases(mesh size h decreases). Since, we used RK4 Method to solve IVP, we got the numerical convergence rate tending to 4, which was our theoretical expectation by Theorem of Convergence for Shooting method. We also plotted the numerical solutions for different N value and the exact solution. The convergence of scheme is also seen from the plot.

References

[1] Srimanta Pal(2009) *Numerical Methods:Principles, Analyses and Algorithms*, Oxford University Press.

[2] Endre Suli and David F. Mayers(2003) *An Introduction to Numerical Analysis*, Cambridge University Press.