Numerical Analysis of Boundary Value Problem Using the Shooting Method



Cluster Innovation Centre, University of Delhi

GC Narang Road, Delhi, 110007

PARTIAL DIFFERENTIAL EQUATIONS PROJECT REPORT

**III.3** **Modeling change in the world around us: Partial Differential Equations**

December, 2022

Report By: Mentor:

Kritika Verma Dr. Sonam Tanwar

B. Tech (IT and MI)

Semester III

152123

Certificate of Originality

This is to certify that this project report on, “Numerical Analysis of Boundary Value Problem Using the Shooting Method” written by Kritika Verma for the paper, **“Modeling change in the world around us: Partial Differential Equations”** does not contain any plagiarism of any form**.**



Kritika Verma

Acknowledgement

I would like to express my gratitude towards Dr. Sonam Tanwar who gave the opportunity to write this report which led to me gaining valuable lessons as well as a deep learning and understanding of the topic. I would also like to thank her for her generous guidance and mentorship throughout the project.

I would also like to extend my gratitude to my family, friends as well as peers for their whole hearted support and help during the completion of this project.

**Numerical Analysis of Boundary Value Problem Using the Shooting Method**

By

**Kritika Verma**

B Tech (IT and MI) 2021-25, Cluster Innovation Centre

University of Delhi

This report focuses on numerical analysis of boundary value problems for Linear and Non-Linear ODEs using the shooting method. In the shooting method a boundary value problem is solved using the initial condition The Euler’s Method as well as the Runge-Kutta second derivative method are used to solve the problem and later compared on the basis of their efficiency.

Index

1. Introduction

1.1 Background and Context

1.2 Scope and Objectives

1.3 Achievements

2. Formulation of the Problem

2.1 Problem Statement

2.2.1 For Linear Second Order ODE

2.2.1.1 Euler’s Method

2.2.1.2 Runge-Kutta 2nd Method

2.2 Methodology

2.3 Results

3. Conclusion

4. References

5. Appendix

1. Introduction

1.1 Background and Context

Shooting method is a numerical method used for numerical analysis of boundary value problems by using the given boundary conditions to “shoot” an initial condition with satisfies those boundary conditions. It involves finding a solution initial condition for different initial value problems until we find the solution which also satisfies the boundary conditions of the boundary value problem. In other words, we shoot out trajectories in different directions towards a particular boundary condition until we find an initial condition that hits it. Hence, its name the shooting method.

1.2 Scope and Objectives

* To learn the numerical analyses method that is shooting method.
* Using the shooting method to solve various ODEs with the help of Euler’s Method.
* To shoot or find an initial condition for a linear and non-linear ODE which satisfies the given boundary condition.
* To shoot our predictions on the basis of the already used values.
* Comparing the solution and accuracy of Euler’s Method with that of Runge-Kutta Second Derivative method.
* Using MATLAB to make guesses for the initial conditions.

1.3 Achievements

We learned how the shooting method (a numerical analysis technique) works and how it can be used to predict an initial condition and then land on the correct initial condition verifying it with the boundary condition provided. We solved linear and non-linear ODEs with shooting method using Euler’s method as well as Runge-Kutta Method and compared the efficiency of both the methods by the amount of guess it took to solve a problem by both the methods. We also used MATLAB to find the next value to shoot so we can get close to the actual initial condition.

2. Formulation of the Problem

2.1 Problem Statement

While solving Ordinary Differential Equations with given boundary value problems it can be really cumbersome, futile and difficult. If given an initial problem with the equation, the process to solve that ordinary differential equation can become more easier as well as efficient to solve. Shooting method provides with a way to find the initial value problem even when only boundary value problem is given with the equation with the help of methods to solve ODEs like Euler’s method or Runge-Kutta method.

2.2 Methodology

2.2.1 For Linear Second Order ODE

Let’s say we have a linear boundary value problem in the form:

A + B + C y + D x = 0 (1)

With boundary conditions

y(x1) = y1 (2)

And

y(x2) = y2 (3)

For example, we have equation (4)

- 2y- 8x (9-x) = 0 (4)

With boundary conditions

y(o) = 0(5)

And

y(9) = 0(6)

Let’s shoot an initial condition say,

y’(0) = 4(7)

2.2.1.1 Euler’s Method

The Euler’s method is a first-order numerical procedure for solving ordinary differential equations (ODE) with a given initial value. Euler’s method uses the first given value to give the next values using the general formula.

*yi*+1*= yi +h f(xi,yi)* (8)

where,

* *yi+1*is the next estimated solution value;
* *yi* is the current value;
* *h* is the interval between steps;
* *f* (*xi,yi*)is the value of the derivative at the current (*xi,yi*) point.

Since, Euler’s method is a first order method, we first convert the equation and make it a first order ODE.

*y*’’ = *f* (*x, y,*  *)* (9)

- 2*y-* 8*x* (9*-x*) *=* 0 (10)

1. Let’s take

=*z = f1* (*x,y,z*) (11)

for the given boundary condition *y*(0)=0.

2) Similarly,

*y’’=*  = *f2* (*x,y,z*) =*2y+8x(9-x)* (12)

for our initial condition *y’*(0)=z(0)=4.

For Euler’s method we have,

1. *yi*+1*= yi +f1(xi,yi,*,*zi)\*h* (13)
2. *zi+1= zi +f2(xi,yi,*,*zi)\*h* (14)

We are taking ***h=3****.*

**At *i=0***

*y*1*= y0 +f1(x0,y0,z0)\*h* (15)

*z*1*= z0 +f2(x0,y0,*,*z0)\*h* (16)

*x0=0; y0=0; z0=4; h=3*

Calculation with equation 15 we get,

*y1= y0 +f1(x0,y0,z0)\*h*

*= 0+f1(0,0,4)\*3=0+4\*3=12* (17)

*z*1*= z0 +f2(x0,y0,*,*z0)\*h*

*= 4+[2y+8x(9-x)]\*3=4+0=4* (18)

*y(x1)y1=12* (19)

*z(x1)z1=4* (20)

*x1=3* (21)

**At *i=1***

*y*2*= y1 +f1(x1,y1,z1)\*h* (22)

*z*2*= z1 +f2(x1,y1,*,*z1)\*h* (23)

*x1=x0+h=0+3=3; y1=12; z1=4; h=3*

Calculation with equation 22 we get,

*y*2*= y1 +f1(x1,y1,z1)\*h*

*= 12+f1(3,12,4)\*3=12+4\*3=24*  (24)

*z*2*= z1 +f2(x1,y1,*,*z1)\*h*

*= 4+[2y+8x(9-x)]\*3=4+68\*3=508* (25)

*y(x2)y2=24* (26)

*z(x2)z2=508* (27)

*x2=6* (28)

**At *i=2***

*y*3*= y2 +f1(x2,y2,z2)\*h* (29)

*z*3*= z2 +f2(x2,y2,*,*z2)\*h* (30)

*x2=x1+h=3+3=6; y1=24; z1=508; h=3*

Calculation with equation 29 we get,

*y*3*= y2 +f1(x2,y2,z2)\*h*

*= 24+f1(6,24,508)\*3=1548*  (31)

*y(x3)y3=1548* (32)

*x3=9* (33)

Equation 32 implies,

*y(x3)=y(9)=1548*  (34)

Since, our guess was wrong, lets take any other arbitrary value for initial value problem.

Say,

*y’*(0)=-24

After going through all the steps of the Euler’s Method we get,

|  |  |  |  |
| --- | --- | --- | --- |
| *i* | *xi* | *yi* | *zi* |
| 0 | 0 | 0 | -24 |
| 1 | 3 | -72 | -24 |
| 2 | 6 | -144 | -24 |
| 3 | 9 | -216 | \_ |

Table 1.1 Solutions of Euler’s Method at y’(0)=4

*y(x3)=y*(9)=-216

Interpolation Technique

Since, we can’t keep making random guesses. Fortunately, we can make are next value more accurate by using the formula to get straight line with two points.

guess 3=guess 2+m(target−solution 2) (35)

m=(guess 1−guess 2)/(solution 1−solution 2) (36)

Using formula 35 and 36 we get

*y’*(0)=-21.57

The solution we from this guess are:

|  |  |  |  |
| --- | --- | --- | --- |
| *i* | *xi* | *yi* | *zi* |
| 0 | 0 | 0 | -21.57 |
| 1 | 3 | 7 | -21.57 |
| 2 | 6 | -123.42 | -24 |
| 3 | 9 | 0.09 | \_ |

Table 1.2 Solutions of Euler’s Method at y’(0)=-20.57

*y(x3)=y*(9)=0.09

Thus, on the third guess, we get the solution close our target value.

2.2.1.2 Runge-Kutta 2nd Method

Using the Runge-Kutta second derivative method for

*y*’(0) = 4

we get,  
 *y*(9)=25200

Taking our second guess as

*y*’(0) = -24

we get,

*y*(9)=-1512

Using formula 35 and 36 we get

*y’*(0)=-18.8679

We got

*y*(9)= 3384.0234

As we can see that well have to use the method again a few times in order to get correct solution while using the Runge-Kutta Method.

2.2.1 For Non-Linear Second Order ODE

To get to a solvable ODE, we start with the conservation of momentum equation (i.e., Navier–Stokes equation) in the x-direction:

u+v +=ν

and the conservation of mass equation:

+ =0

where u is the velocity component in the xx-direction, v is the velocity component in the y-direction, and ν is the fluid’s kinematic viscosity. The boundary conditions are that *u*=*v*=0 at *y*=0, and that u=U∞=U∞ as y→∞ where U∞ is the free-stream velocity.

Blasius solved this problem by converting the PDE into an ODE, by recognizing that the boundary layer thickness is given by δ(x)∼√xν/U∞, and then nondimensionalizing the position coordinates using a similarity variable.

η=*y*

By introducing the stream function, ψ(x,y), we can ensure the continuity equation is satisfied:

*u*=, *v*=

Let’s check this, using SymPy (Appendix 1.4)

True

Using the boundary layer thickness and free-stream velocity, we can define the dimensionless stream function f(η):

f(η)=

which relates directly to the velocity components:

*u==*

*u=. f’().*

*u= f’(*

*v==-(+)= (*

We can insert these into the x-momentum equation, which leads to an ODE for the dimensionless stream function f(η):

*f′′′+ff′′=0*

with the boundary conditions *f=f′=0* at *η=*0, and*f′*=1 as  *η→∞*.

This is a 3rd-order ODE, which we can solve by converting it into three 1st-order ODEs:

*y1=f y′1=y2*

*y2=f’ y′2=y3*

*y3=f’’ y′3=-y1y3*

and we can use the shooting method to solve by recognizing that we have two initial conditions, *y1(0)=y2(0)= 0*, and are missing *y3*(0). We also have a target boundary condition: *y*2(∞)=1.

(Note: obviously we cannot truly integrate over 0≤*η*<∞. Instead, we just need to choose a large enough number. In this case, using 10 is sufficient.)

We created a MATLAB function to evaluate the derivative (appendix 1.1).

Trying the same three-step approach we used for the simpler example, taking two guesses and then using linear interpolation to find a third guess (appendix 1.2) we get:

|  |  |  |
| --- | --- | --- |
| *i* | y(xi) | *y’*(*xi*) |
| 1 | 1 | 1.6553 |
| 2 | 0.1 | 0.3566 |
| 3 | 0.54587 | 1.1056 |

Table 1.3 Solution for Non-Linear PDE using 3 iterations

Our target value was 1.00.

So, for this problem, using linear interpolation did not get us the correct solution on the third try. This is because the ODE is nonlinear. But, you can see that we are converging towards the correct solution—it will just take more tries.

Rather than manually take an unknown (and potentially large) number of guesses, we automate this with a while loop (appendix 1.3).

2.3 Results

We were successful in solving the linear ordinary differential equation using Shooting method with the help of Euler’s Method which was found to be more efficient than Runge-Kutta method in the three iteration.

For non-linear ODE we had put make a while loop for when the condition was satisfied rather than using just 3 iteration and the result came out be that 7 iterations were needed to get to the target value.

|  |  |  |
| --- | --- | --- |
| *i* | y(xi) | *y’*(*xi*) |
| 1 | 1 | 1.6553 |
| 2 | 0.1 | 0.3566 |
| 3 | 0.54587 | 1.1056 |
| 4 | 0.48301 | 1.019 |
| 5 | 0.46922 | 0.99951 |
| 6 | 0.46957 | 1 |
| 7 | 0.46957 | 1 |

Table 1.3 Solution for Non-Linear PDE using while loop

3. Conclusion

Shooting method can be useful in solving second order ODEs where only boundary conditions are given. It is easy and uses Euler’s method or Runge-Kutta method to find the initial value problem with accuracy. Like how a cannon shooting out an object and adjusting its trajectory after every shot to land the object on a given mark, shooting method shoot the initial condition towards what can actually be the condition which satisfies the boundary conditions. The guess is adjusted with the help of interpolation which makes the next guess more accurate and hence, closer to the target.

4. Reference

* Introduction to shooting method: [*https://kyleniemeyer.github.io/ME373-book/content/bvps/shooting-method.html*](https://kyleniemeyer.github.io/ME373-book/content/bvps/shooting-method.html)[accessed on 7 December 2022]
* Shooting Method Definition: [*https://pythonnumericalmethods.berkeley.edu/notebooks/chapter23.02-The-Shooting-Method.html*](https://pythonnumericalmethods.berkeley.edu/notebooks/chapter23.02-The-Shooting-Method.html)[accessed on 5 Dec 2022]
* Shooting method: [*https://en.wikipedia.org/wiki/Shooting\_method#:~:text=In%20numerical%20analysis%2C%20the%20shooting,to%20an%20initial%20value%20problem*](https://en.wikipedia.org/wiki/Shooting_method#:~:text=In%20numerical%20analysis%2C%20the%20shooting,to%20an%20initial%20value%20problem)*.* [accessed on 5 December 2022]
* Shooting method lesson 1: [*https://www.youtube.com/watch?v=ZMgikZ-lcS8&ab\_channel=numericalmethodsguy*](https://www.youtube.com/watch?v=ZMgikZ-lcS8&ab_channel=numericalmethodsguy)[accessed on 5 December 2022]
* Shooting method lesson 2: *https://www.youtube.com/watch?v=R7I8FrlB\_KM&ab\_channel=numericalmethodsguy* [accessed on 5 December 2022]
* Shooting method lesson 3: *https://www.youtube.com/watch?v=UmSVUqkKvrE&ab\_channel=numericalmethodsguy* [accessed on 5 December 2022]
* Shooting method lesson 4: *https://www.youtube.com/watch?v=Sc0Z\_qIndz0&ab\_channel=numericalmethodsguy* [accessed on 5 December 2022]
* BVPs and ODEs: [*http://www.math.pitt.edu/~sussmanm/2071Spring09/lab04/index.html*](http://www.math.pitt.edu/~sussmanm/2071Spring09/lab04/index.html)[accessed on 4 December 2022]
* Runge-Kutta Simulator: [*https://atozmath.com/CONM/RungeKutta.aspx?q=rk4&m=2&q1=2\*y%2B8x(9-x)%600%600%609%603%60rk2%60-24&dp=4&do=1#tblSolution*](https://atozmath.com/CONM/RungeKutta.aspx?q=rk4&m=2&q1=2*y%2B8x(9-x)%600%600%609%603%60rk2%60-24&dp=4&do=1#tblSolution)[accessed on 7 December 2022]

5.Appendix

Appendix 1.1

*%%file blasius\_rhs.m*

**function** dydx = blasius\_rhs(eta, y)

dydx = zeros(3,1);

dydx(1) = y(2);

dydx(2) = y(3);

dydx(3) = -y(1) \* y(3);

Appendix 1.2

clear all; clc

target = 1.0;

guesses = zeros(3,1);

solutions = zeros(3,1);

guesses(1) = 1;

[eta, F] = ode45('blasius\_rhs', [0 10], [0 0 guesses(1)]);

solutions(1) = F(**end**, 2);

guesses(2) = 0.1;

[eta, F] = ode45('blasius\_rhs', [0 10], [0 0 guesses(2)]);

solutions(2) = F(**end**, 2);

m = (guesses(1) - guesses(2))/(solutions(1) - solutions(2));

guesses(3) = guesses(2) + m\*(target - solutions(2));

[eta, F] = ode45('blasius\_rhs', [0 10], [0 0 guesses(3)]);

solutions(3) = F(**end**, 2);

tries = [1; 2; 3];

table(tries, guesses, solutions)

fprintf('Target: %5.2f\n', target);

Appendix 1.3

clear all; clc

target = 1.0;

*% get these arrays of stored values started.*

*% note: I'm only doing this to make it easier to show a table of values*

*% at the end; otherwise, there's no need to store these values.*

tries = [1; 2; 3];

guesses = zeros(3,1);

solutions = zeros(3,1);

guesses(1) = 1;

[eta, F] = ode45('blasius\_rhs', [0 10], [0 0 guesses(1)]);

solutions(1) = F(**end**, 2);

guesses(2) = 0.1;

[eta, F] = ode45('blasius\_rhs', [0 10], [0 0 guesses(2)]);

solutions(2) = F(**end**, 2);

num = 2;

solutions(3) = -1000.; *% doing this to kick off the while loop*

**while** abs(target - solutions(num)) > 1.e-9

num = num + 1;

m = (guesses(num-2) - guesses(num-1))/(solutions(num-2) - solutions(num-1));

guesses(num) = guesses(num-1) + m\*(target - solutions(num-1));

[eta, F] = ode45('blasius\_rhs', [0 1e3], [0 0 guesses(num)]);

solutions(num) = F(**end**, 2);

tries(num) = num;

*% we should probably set a maximum number of iterations, just to prevent*

*% an infinite while loop in case something goes wrong*

**if** num >= 1e4

**break**

**end**

**end**

table(tries, guesses, solutions)

fprintf('Number of iterations required: %d', num)

Appendix 1.4

*%%python*

import sympy as sym

sym.init\_printing()

x, y, u, v = sym.symbols('x y u v')

# Streamfunction

psi = sym.Function(r'psi')(x,y)

# Define u and v based on the streamfunction

u = psi.diff(y)

v = -psi.diff(x)

# Check the continuity equation:

print(u.diff(x) + v.diff(y) == 0)