Solution of the ‘Blasius Equation’ (Boundary Layer Theory) using the ‘Shooting Method’

MA 202: Mathematics-IV
Group Project
Group Number: 15
Project Number: 12

Aditya Gupte	21110012
Gaurav Kumar Rawat	21110066
Indira Gadhvi	21110080
Siddharth Shah	21110208
Siddhi Rajpurohit	21110209

Acknowledgement

We would like to convey our heartfelt gratitude to everyone who helped us finish this project on numerically solving the 'Blasius problem' using the 'Shooting method.'

First and foremost, we would like to thank Prof. Uddipta Ghosh for giving us this opportunity to conduct research in the area of the ‘Blasius equation’ and its solution.. His method of teaching has been invaluable throughout this course. We also appreciate his availability to answer our queries and provide us with the necessary resources to complete this project successfully.

We also like to thank IIT Gandhinagar for providing us with the necessary tools and software we needed to finish this project. Finally, we would like to extend our heartfelt gratitude to everyone who helped make this attempt a success.

Table of Contents

Abstract

The Blasius equation is a third-order Ordinary Differential Equation that provides the similarity solution for the velocity field in a hydrodynamic boundary layer formed over a flat plate with uniform flow. The Shooting technique is used in this project to develop a Python code to numerically solve the Blasius equation and tabulate the results for f, f'' and f'''.

The Shooting method is used to numerically solve the Blasius equation by transforming it into a boundary value problem with a shooting parameter. The 'Secant' method is used to solve for the shooting parameter, followed by the Runge-Kutta fourth-order approach to predict the value of the unknown initial condition for $f'(0)$ and then integrate the equation using a numerical integrator until the value of $f'(\infty)$ converges to 1. The process is repeated until the correct value of $f'(0)$ is obtained. The computed values of f, f'' and f''' are then recorded and plotted as functions of 'η.'

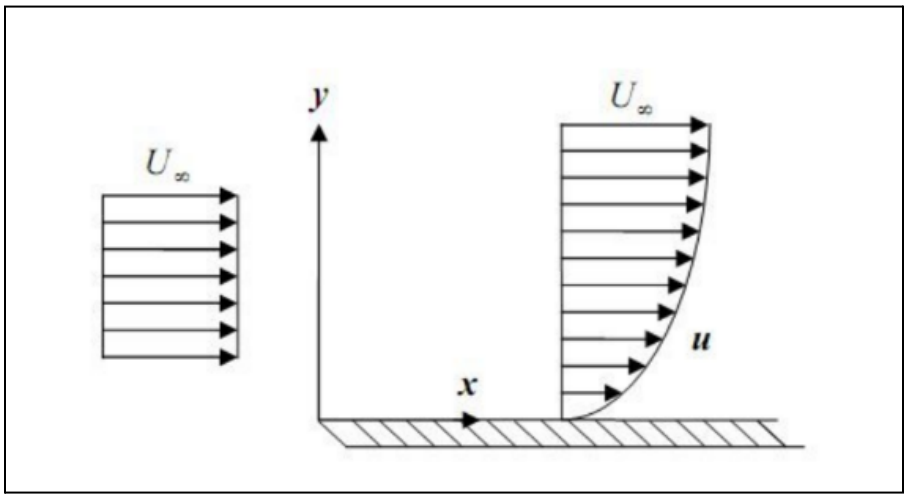
The results indicate that the functionf and its derivatives vary smoothly with respect to η, and that f'' and f''' are positive for all values of η. The project demonstrates the use of numerical methods to solve differential equations. Additionally, it offers a helpful method for estimating the velocity profile in a hydrodynamic boundary layer formed over a flat plate due to a uniform far-field flow.

Introduction

The Blasius equation is a classical fluid mechanics problem that describes a viscous fluid's flow over a flat plate. It is is vital in the study of boundary layers and provides a similarity solution for the velocity field in a hydrodynamic boundary layer that develops over a flat plate.

The dimensionless velocity profile, f, the similarity parameter, η, and the second and third derivatives of f with respect to η make up the Blasius equation. The similarity variable is denoted by the formula $\eta = y/(vx/U_\infty)^{1/2}$, where y is the distance from the surface of the plate, v is the kinematic viscosity of the fluid, x is the distance along the surface of the plate, and U_∞ is the free-stream velocity of the fluid.

While analytical solutions to this equation are not possible, numerical techniques can be used to obtain approximate solutions. The shooting method is one such technique that involves solving a boundary value problem by guessing the initial conditions and iteratively adjusting them until the desired boundary condition is satisfied. This paper focuses on solving the Blasius equation using the shooting method. We aim to provide a detailed analysis of the method and explore its effectiveness in obtaining accurate solutions to this problem. The insights gained from this research will be useful in the study of fluid mechanics and contribute to developing more efficient numerical techniques for solving differential equations.



Boundary layer for a uniform flow over a flat plate.

Numerical Discretization Techniques

Numerical discretization techniques, such as the secant method and the Runge-Kutta method, have been employed in the shooting method to solve the Blasius equation. These approaches assist in approximating the solution of the Blasius problem by breaking it down into a set of equations that may be solved iteratively.

1. The Secant Method:

The secant technique is used to solve for the initial unknown condition $f'(0)$. It entails first guessing $f'(0)$ and then computing $f(\infty)$ via numerical approaches. If $f(\infty)$ is not equal to 1, a new guess for $f'(0)$ is made utilising the previous iterations. The procedure is continued until the value of $f(\infty)$approaches 1. Derivation of the Secant method is explained below.

Newton's technique requires the evaluation of $f'(x)$ at different instances, which would not be feasible for some of the selections of $f(x)$. By approximating the derivative with a limited difference, the secant approach gets around this problem. As a consequence, rather than a tangent line through a single point on the graph, $f(x)$ can be represented by a secant line across two points.

Selecting two initial iterates, x_0, and x_1, is required since a secant line is formed using information from two points on the $f(x)$ graph instead of a tangent line, which only needs information from one point on the graph. The following value x_2 is then calculated by determining the value at which the secant line running between the points (x_0, $f(x_0)$) and (x_1, $f(x_1)$) has a y-coordinate of zero, much like in Newton's approach. It results in the equation:

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_1) + f(x_1) = 0$$

which has the solution:

$$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

Upon further simplification and rewriting,

$$\begin{aligned} x_2 &= x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= x_1 \frac{f(x_1) - f(x_0)}{f(x_1) - f(x_0)} - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= \frac{x_1(f(x_1) - f(x_0)) - f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)} \\ &= \frac{x_1 f(x_1) - x_1 f(x_0) - x_1 f(x_1) + x_0 f(x_1)}{f(x_1) - f(x_0)} \\ &= \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}. \end{aligned}$$

The general formula for the secant method is:

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}, \quad n = 1, 2, 3, \ldots.$$

2. The Runge Kutta Method:

The Runge-Kutta method is further used to solve the Blasius equation once the value of $f'(0)$ has been determined using the secant method. The Blasius equation is decomposed into a system of first-order ODEs, which are then solved numerically.

The derivation of the Range Kutta Method is as follows:

We start with the Taylor series:

$$y(t+k) = y + k\,y' + \frac{k^2}{2}y'' + \frac{k^3}{6}y''' + \ldots \qquad \text{(where } y,\ y',\ \text{etc. are evaluated at time } t\text{)}$$

$$= y + k f + \frac{k^2}{2}f' + \frac{k^3}{6}f'' + \ldots \qquad \text{Swap } y' \text{ for } f \text{ according to the ODE } y'(t) = f(t, y(t))$$

We now swap the derivatives f^k into partial derivatives f_t and f_y of f:

$$f'(t,y) = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt} = f_t + f\,f_y$$

$$f''(t,y) = \ldots \quad = f f_y^2 + f^2 f_{yy} + f_t f_y + 2 f f_{ty} + f_{tt},$$

Thus, the Taylor method for order 3 becomes:

$$y(t+k) = y + k f + \frac{k^2}{2}(f_t + f f_y) + \frac{k^3}{6}(f f_y^2 + f^2 f_{yy} + f_t f_y + 2 f f_{ty} + f_{tt})$$

Finding the coefficients of a general RK method of order 2:

$$\begin{array}{ll} d^{(1)} & = k f(t + 0 \cdot k, y) \\ d^{(2)} & = k f(t + c_1 \cdot k, y + a d^{(1)}) \\ \hline y(t+k) & = y(t) + b_1 d^{(1)} + b_2 d^{(2)} \end{array}$$

We Taylor expand d^1 and d^2 to second order and combine them:

$$y(t+k) = y(t) + k(b_1 + b_2) f + k^2 (b_2 c f_t + b_2 a f f_y) + O(k^3)$$

Similarly for fourth order Runge Kutta:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}\right) \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}\right) \\ k_4 &= f(t_n + h, y_n + h k_3)\,. \end{aligned}$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Numerical Solution Methodology - Code Explanation

The Shooting method is used to solve the the Blasius equation and we have implemented the same in the python code. In the Shooting method, we have used the 4th-order Runge Kutta method, as it was a classic method and was also taught in the course. The code attached to the report depicts the solution of the Blasius equation using the shooting method. The differential equation that we are required to solve is of third order. Hence, we must convert it into a first-order ordinary differential equation(ODE). This is implemented in the definition of function 'f'.

```
# We use Runge kutta order four for solving the system
def RK4_Method(f, a, L, u, N):

    """
    f : RHS fucntion of the system of ODE's
    u : the initial condition
    N : no. of interval points

    eta ranges between the domain interval: [a, L]

    """
    h = (L - a) / N         # step size between interval points
    answer = []             # To store the numerical solution
    interval_pts  = []      # To store the interval points
    answer.append(u)
    interval_pts.append(a)
    x = a
    for i in range(1, N + 1):
        k1 = f(u , x)
        k2 = f(u + 0.5*h*k1  , x+0.5*h)
        k3 = f(u + 0.5*h*k2  , x+0.5*h)
        k4 = f(u + h*k3 , x+h)

        u = u + h * (k1 + 2 * k2 + 2 * k3 + k4)/6   # Value at the next point
        x = x + h                                   # The next point after step increment
        answer.append(u)
        interval_pts.append(x)
    return np.array(interval_pts ), np.array(answer)
```

We now shift our focus to the conditions given to us. The Blasius equation is a boundary value problem. We solve it by converting it to an initial value problem by taking an initial guess for the value of the derivative of f. Then we solve the three first-order differential equations using the 4th-order Runge-Kutta method to check if the value of the function u_2 at the endpoints thus derived lies within a tolerance of the given final condition, which is called the target. This guess is called the shooting parameter 's'. In order to find the correct 's', we define the function

$F(s) = u_2$ (at L for given s) - target and find its zeroes using the secant method. Note that we approximate infinity with a large value L. An important idea in this code was to condense all three equations and solve them simultaneously as an array, so that at each new iteration, the values of u1 and u2 could be accessed to compute u3.

```
# initial conditions with the shooting parameter s are assigned through an array
def U_fixed(s):
    return np.array([0, 0, s])

# F(s) = u_2(s, L) - 1: Aim for s such that F(s) becomes zero.
def F(s, L):
    u = U_fixed(s)
    interval_pts , answer = RK4_Method(f, a, L, u, N)
    F = (answer[len(answer)-2])[1] - target
    return F

#formulate as a first-order system and  define the right side function for the nonlinear ode
def f(u , t):
    f = np.zeros(3)
    f[0] = u[1]
    f[1] = u[2]
    f[2] = -0.5 * u[0] * u[2]
    return f
# the secant method for solving the algebric equation F(s) = 0
# We apply the secant method to find the zeroes for F(s,L)
```

We used the secant method to solve for the roots because it is harder to compute the value of the derivative of F with respect to s. Since we don't know the form of f', we have used secant method instead of the Newton's method. Since these methods are more accurate, we have used the secant method over the Bisection method.

```
def secant_method(F,s0,s1,etol,L,maxiter):
  global s
  '''s0 and s1 are initial guesses.'''
  iterations = 0
  F0 = F(s0,L)
  F1 = F(s1, L)
  while abs(F1) > etol and iterations < maxiter:
    slope = (F1 - F0)/(s1 - s0)
    s0 = s1
    s1 = s1 - F1/slope
    iterations += 1
    F0 = F1
    F1 = F(s1,L)
  if abs(F1) > etol :
    print('max iterations reached.')
  return s1, iterations
```

While taking the inputs, we initialised the shooting parameter 's' through an array and set the tolerance as 10^{-6}. We took data points of f and repeated the iterations 6 times. We gave the output to plot the graph obtained by implementing our code. All the inputs are present in the code, and it just needs to be run. The initial guesses have been put after several trial and error inputs. For guesses with large numbers, the function does not return any value. Hence, the guess values have been kept small.

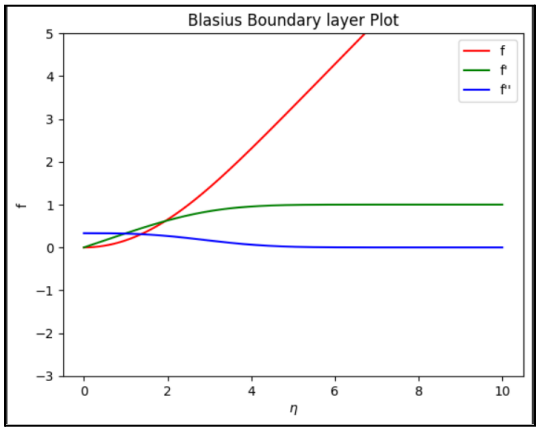
```
# Parameters
a = 0
L = 10
N = 1000
target = 1

# initializing the parameter s, and setting the tolerance error
s0 = 1.0
s1 = 2.0
tol = 1.0e-6
s, tot_iters = secant_method(F, s0, s1, tol, L,200)
print('shooting parameter =', s)
print('number of iterations = ', tot_iters)
print()

# Now we use the approperaite shooting parameter in the initial condition and solve for the system of ODE
u = U_fixed(s)
interval_pts , answer = RK4_Method(f, a, L, u, N)

print("Data for f: ")
df_f = pd.DataFrame(answer[:, 0])
print(df_f,"\n")

print("Data for f': ")
df_f1 = pd.DataFrame(answer[:, 1])
print(df_f1,"\n")

print("Data for f'': ")
df_f2= pd.DataFrame(answer[:, 2])
print(df_f2,"\n")
```

The code was inspired from the research article [1].

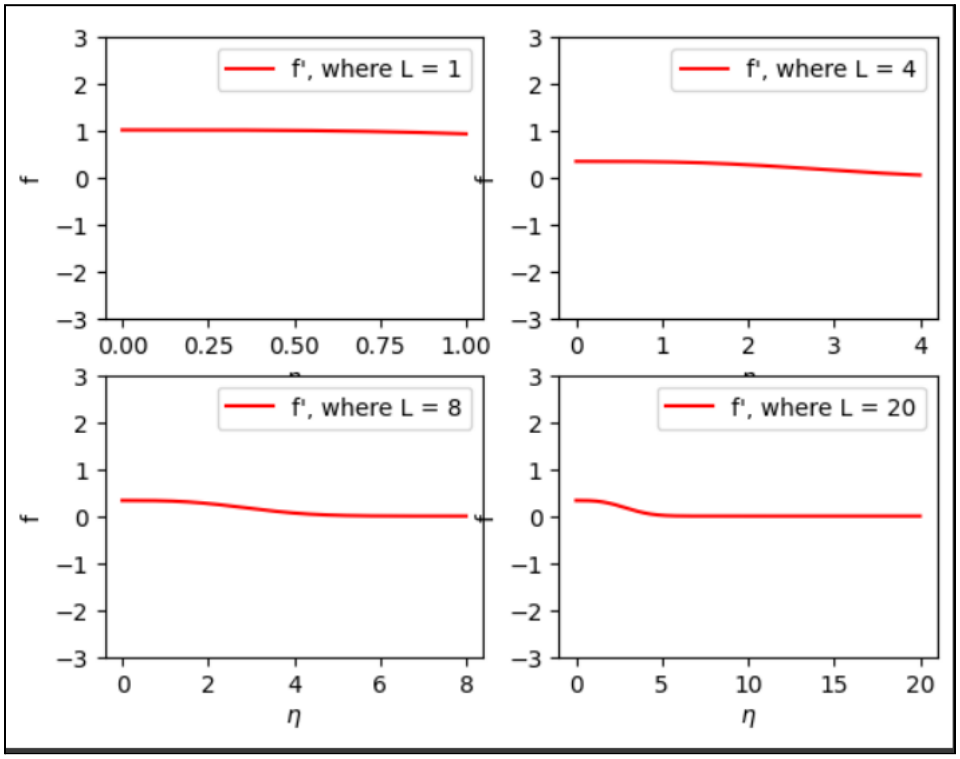
Graphs

After plotting the results, the graphs we obtained are shown below:

The following plot is obtained when $L = 10$.



The set of graphs depicted below illustrate the representation of values of L in the plot. We observe the graph for different values of L and the varying slopes that correspond with each value. As the L value increases, we note a rapid convergence of f' towards zero.



Applications of the Blasius Equation

The Blasius equation is a nonlinear ordinary differential equation that describes the steady, laminar flow of an incompressible fluid over a flat plate. This equation has a wide range of applications in fluid mechanics, and some of them are:

1. **Boundary Layer Theory:** The thin fluid layers that form close to solid surfaces as a result of viscous processes are known as boundary layers, and the Blasius equation is a crucial tool for understanding them. The velocity and thickness of the boundary layer, crucial variables in the heat transfer and fluid dynamics analysis, may be roughly determined from the Blasius equation's solution.

2. **Drag Coefficient:** The drag coefficient, a gauge of an object's resistance to motion through a fluid, may be calculated using the Blasius equation. By integrating the Blasius equation solution, one may get a formula for the drag coefficient in terms of the Reynolds number, a dimensionless quantity that describes the fluid flow.

3. **Heat Transfer:** The convective heat transfer coefficient, a measurement of the rate of heat transfer between the fluid and the surface, may be calculated using the velocity and temperature profiles derived from the Blasius equation's solution.

4. **Fluid Mechanics:** The study of laminar boundary layer flow across curved surfaces, the computation of skin friction drag, and the identification of the commencement of turbulence in a fluid flow are just a few examples of the many fluid mechanics applications of the Blasius equation.

5. **Mechanical and Thermodynamic Systems:** A variety of engineering systems, including heat exchangers, turbines, and airplane wings, are optimized using the Blasius equation.

References

1. Non-linear ode: Shooting method python code - researchgate (no date). Available at: https://www.researchgate.net/publication/347387708_Non-Linear_ODE_Shooting_Method_Python_Code (Accessed: April 24, 2023).

2. The secant method - USM (no date). Available at: https://www.math.usm.edu/lambers/mat772/fall10/lecture4.pdf (Accessed: April 26, 2023).

3. Derivation of runge-kutta methods - university of Colorado Boulder (no date). Available at: https://www.colorado.edu/amath/sites/default/files/attached-files/rk_derivation_0.pdf (Accessed: April 26, 2023).