# Numerical Solution of The Blasius Boundary Layer Equation over a Flat Plate

**ENGR 6201: Fluid Mechanics**

Professor Ida KarimFazli

**Student's Name**
Dara Rahmat Samii

**Student's ID number**
40281972

November 2023

# Chapter 1

# Introduction

The Blasius problem represents a classical challenge in the field of fluid dynamics, specifically addressing the laminar boundary layer flow over a flat plate.

Consider a hypothetical scenario where a flat plate is subjected to a uniform flow with a constant velocity denoted as $U_e$. This hypothetical scenario is depicted in Figure 1.1. The primary objective is to characterize the profiles of velocity and boundary layer thickness as the fluid progresses along the plate. The mathematical description of this scenario as shown in Eq. 1.1, 1.2 and 1.3 involves the incompressible Navier-Stokes equations, which, under carefully defined assumptions, transform into a specialized system recognized as the Blasius equations.

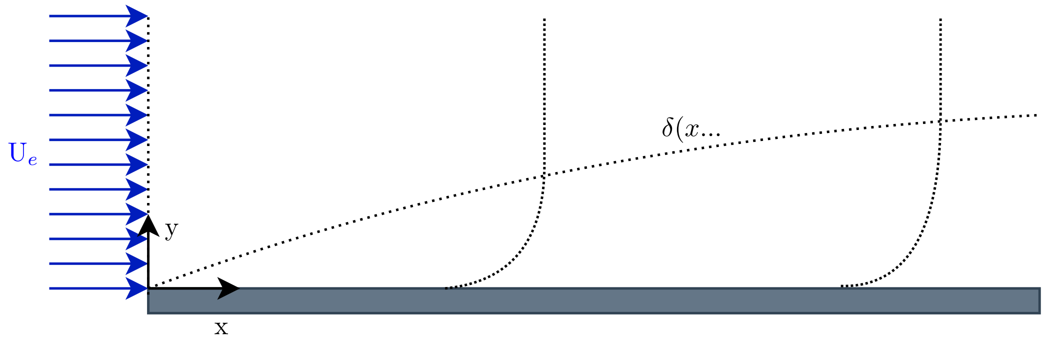

Figure 1.1: Schematic of Blasius plate problem[1]

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial x} + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \tag{1.1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial P}{\partial y} + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) \tag{1.2}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1.3}$$

The boundary conditions are as follows:

$$BC = \begin{cases} u(x=0,y) = U_e \\ u(x,y=0) = 0 \\ v(x,y=0) = 0 \\ u(y \to \infty) = U_e \end{cases} \tag{1.4}$$

For this practical problem, the following values are provided for the length of the plate, fluid density, and viscosity:

$$L = 1m \qquad \rho = 1.2\frac{kg}{m^3} \qquad \mu = 1.7 \times 10^{-7} Pa.s \tag{1.5}$$

[1]Schemaic made with https://app.diagrams.net

# Chapter 2

# Derivation

The Blasius equations model a steady, two-dimensional flow characterized by laminar boundary layers. Key assumptions inherent in this model include the neglect of buoyancy effects, a constant freestream velocity, and the consideration of a small boundary layer thickness. These assumptions collectively yield a third-order ordinary differential equation, known as the Blasius equation.[1],[2],[3]

## 2.1 Assumptions

There is assumptions for the blasuis equations for boundary layer:

1. The flow is steady state ($\frac{\partial}{\partial t} \approx 0$)
2. Boundary layer thickness ($\delta$) is small ($u \gg v$, $\frac{\partial}{\partial x} \ll \frac{\partial}{\partial y}$)
3. Boundary layer is laminar
4. Buoyancy effects are negligible
5. Freestream velocity $U_e$ is constant ($\frac{dU_e}{dx} \approx 0$, $\nabla P \approx 0$)

Applying the aforementioned assumptions to the complete equations governing motion in a two-dimensional (2D) flow yields a refined and simplified set of equations of Eq.2.1 and Eq. 2.2

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.1}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial y^2} \tag{2.2}$$

Defining the stream function allows for the elimination of the conservation of mass equation

$$u = \frac{\partial \psi}{\partial y} \qquad\qquad v = -\frac{\partial \psi}{\partial x} \tag{2.3}$$

By employing the separation of variables method Eq. 2.4 can be attained.

$$\Psi(x, y) = \lambda(x).f(\eta) \qquad\qquad \eta = \eta(x, y) \tag{2.4}$$

By substituting Equation 2.4 into Equation 2.3, the resultant expression of Eq.2.5 is obtained:

$$u = \frac{\partial \psi}{\partial y} = \lambda\frac{\partial \eta}{\partial y}.\frac{df}{d\eta} \qquad\qquad v = -\frac{\partial \psi}{\partial x} = -\frac{d\lambda}{dx}f - \lambda\frac{\partial \eta}{\partial x}.\frac{df}{d\eta} \tag{2.5}$$

By nondimensionalizing the variable $y$ and introducing a new nondimensional variable $\eta$:

$$\eta = y.g(x) \qquad\qquad or \qquad\qquad \eta = \frac{y}{\delta(x)} \tag{2.6}$$

Worth noting that ($g(x) = \frac{1}{\delta(x)}$, which facilitates obtaining derivatives easily. Here, $\delta(x)$ represents the boundary layer thickness and varies only with $x$. By substituting 2.6 into Eq.2.5, it results:

$$\frac{\partial \eta}{\partial x} = y\frac{dg}{dx} = \frac{\eta}{g}\frac{dg}{dx} \qquad\qquad \frac{\partial \eta}{\partial y} = g(x) \tag{2.7}$$

Incorporating Equation 2.7 and Equation 2.6 into Equation 2.5, the resulting set of equations is Eq.2.8.

$$u = \lambda g \frac{df}{d\eta} \qquad\qquad v = -\frac{d\lambda}{dx} f - \frac{\lambda \eta}{g} \frac{dg}{dx} \frac{df}{d\eta} \tag{2.8}$$

Based on the boundary conditions $u(x, \infty) = U_e$ if $\frac{df}{d\eta}(\infty) = 1$:

$$u = \lambda g \frac{df}{d\eta} \longrightarrow U_e = \lambda g \longrightarrow \lambda = \frac{U_e}{g} \tag{2.9}$$

And then we can obtain:

$$\frac{d\lambda}{dx} = \frac{d}{dx}(\frac{U_e}{g(x)}) = -\frac{U_e g'}{g^2} \tag{2.10}$$

By substituting Eq.2.9 and Eq.2.10 in Eq.2.8 the following equations (Eq. 2.11) yields:

$$u = U_e f' \qquad\qquad v = \frac{U_e g'}{g^2}(f - \eta f') \tag{2.11}$$

And Finally, we are able to compute the derivatives in the simplified 2d motion equation Eq.**??**:

$$\frac{\partial u}{\partial x} = U_e \frac{\eta g'}{f''} \qquad\qquad \frac{\partial u}{\partial y} = U_e g f'' \qquad\qquad \frac{\partial^2 u}{\partial y^2} = U_e g^2 f''' \tag{2.12}$$

By substituting the equations Eq.2.12in Eq.2.2:

$$\begin{aligned} \frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \nu \frac{\partial^2 u}{\partial y^2} \longrightarrow [U_e f'][U_e \frac{\eta g'}{g} f''] + [\frac{U_e g'}{g^2}(f - \eta f')][U_e g f''] = \nu U_e g^2 f''' \longrightarrow \\ \longrightarrow U_e[\cancel{U_e \eta \frac{g'}{g} f'} f'' + \frac{U_e g'}{g} f f'' - U_e \eta \frac{g'}{g} f f'' - \cancel{U_e \eta \frac{g'}{g} f' f''}] = \nu g^2 f''' U_e \longrightarrow \\ \longrightarrow \div \nu U_e g^2 \longrightarrow \boxed{f''' - \frac{U_e g'}{\nu g^3} f f'' = 0} \end{aligned} \tag{2.13}$$

Now we need to Find $g(x)$, and for this purpose, we make the assumption:

$$\frac{-U_e^2 g'}{g^3} = \frac{1}{2} \tag{2.14}$$

By integrating the equation Eq.2.14:

$$\frac{-U_e}{\nu} \frac{1}{g^3} \frac{dg}{dx} = \frac{1}{2} \longrightarrow \frac{-U_e}{\nu g^3} dg = \frac{1}{2} dx \longrightarrow -\frac{U_e}{\nu} \int \frac{dg}{g^3} = \int dx \longrightarrow \frac{U_e}{2\nu} \frac{1}{g^2} = \frac{x}{2} \longrightarrow \boxed{g(x) = \sqrt{\frac{U_e}{\nu x}}} \tag{2.15}$$

It is essential to recall that in Eq.2.6 $\eta = y.g(x)$:

$$\eta = y.g(x) \longrightarrow \eta = y\sqrt{\frac{U_e}{\nu x}} \tag{2.16}$$

Rewriting the velocity components $u$ and $v$:

$$u(\eta) = U_e f'(\eta) \tag{2.17}$$

$$v(\eta) = \nu g(x)[\eta f'(\eta) - f(\eta)] \tag{2.18}$$

Rewriting the boundary equations :

$$u(x, y = 0) = 0 \longrightarrow u(\eta = 0) U_e f'(\eta = 0) = 0 \longrightarrow f'(\eta = 0) = 0 \tag{2.19}$$

$$v(x, y = 0) = 0 \longrightarrow v(\eta = 0) U_e.g[\eta f'(\eta = 0) - f(\eta)] = 0 \longrightarrow f(\eta = 0) = 0 \tag{2.20}$$

$$u(x, y \rightarrow) = U_e \longrightarrow u(\eta \rightarrow) U_e.f'(\eta \rightarrow \infty) = U_e \longrightarrow f'(\eta \rightarrow \infty) = 1 \tag{2.21}$$

Also with knowing the $\frac{\partial u}{\partial y}$ the wall shear stress can be computed.

$$\tau = \mu \frac{\partial u}{\partial y}|_{y=0} = \mu U_e g(x) f''(\eta = 0) \tag{2.22}$$

By having Eq.2.22 the friction coefficient along the wall force can be computed:

$$C_F(x) = \frac{\tau(x)}{\frac{1}{2}\rho U_e} \tag{2.23}$$

Additionally, By having Eq.2.22 the total force per depth which fluid pushed the plate can be computed:

$$F_D = \int_0^L \tau(x) dx \tag{2.24}$$

and subsequently Drag coefficient can be computed as:

$$C_D = \frac{2F_D}{\rho U^2 A} = \frac{2\int_0^L \mu U_e g(x) f''(0) dx}{\rho U^2 A} \tag{2.25}$$

## 2.2 Summary

Finally, as depicted in the previous sections, the ultimate form of the Blasius ordinary differential equation is as follows:

$$f''' + \frac{1}{2} f f'' = 0 \qquad BC = \begin{cases} f(\eta = 0) = 0 \\ f'(\eta = 0) = 0 \\ f'(\eta \to \infty) = 1 \end{cases} \tag{2.26}$$

$$\eta = y.g(x0) = \frac{y}{\delta(x)} \qquad g(x) = \frac{1}{\delta(x)} = \sqrt{\frac{U_e}{\nu x}} \tag{2.27}$$

$$u(\eta) = U_e f'(\eta) \qquad v(\eta) = \nu g(x)[\eta f'(\eta) - f(\eta)] \tag{2.28}$$

$$\tau = \mu U_e g(x) f''(\eta = 0) \qquad F_D = \int_0^L \tau(x) dx \qquad C_D = \frac{F_D}{\rho U^2 A} \qquad C_f(x) = \frac{2\tau}{\rho U_e} \tag{2.29}$$

# Chapter 3

# Analytical Solution

Ö. Savas in his paper ”An approximate compact analytical expression for the Blasius velocity profile” [4] derived the analytical solution for Blasius equation.

$$f'(\eta) = (tanh[(G\eta)^n])^{\frac{1}{n}} \tag{3.1}$$

In Eq. 3.1, the term $n$ is $\frac{3}{2}$, and $G$ represents $f''(\eta = 0)$, which needs to be determined through a guessing process. The details of finding $G$ are thoroughly discussed in Section 4.1.

# Chapter 4

# Numerical Solution

## 4.1 Finding G

Numerical solution of the Blasius equation requires an initial guess for $G(0) = f''(\eta = 0)$. The final value of $f'$, where the function stabilizes at $\eta_{\text{max}}$, varies with the initial guess. In Fig. 4.1, the plot of $f'(\eta_{\text{max}})$ versus $G$ is illustrated. Based on the boundary conditions, it is known that as $\eta \longrightarrow \infty$, $f'(\eta_{\text{max}})$ should be 1. By utilizing the MATLAB[5] ‘shootingMethod‘ function, we determined that $G = 0.3320573$ satisfies the relevant boundary condition.

### 4.1.1 Shooting Method

The shooting method is a numerical technique widely employed for solving boundary value problems, particularly in the context of ordinary differential equations (ODEs). It transforms a boundary value problem into an initial value problem by introducing an initial guess for the solution. Let $y(x; \alpha)$ represent the solution of the ODE, where $\alpha$ is the vector of parameters that includes the initial guess.

The iterative process of the shooting method involves adjusting the initial guess $\alpha$ until the solution $y(x; \alpha)$ satisfies the prescribed boundary conditions. This is achieved by combining numerical integration, often employing methods like the Euler or Runge-Kutta methods, with a root-finding algorithm such as the Newton-Raphson method. The iterative update of the parameters is given by the formula:

$$\alpha_{\text{new}} = \alpha_{\text{old}} - \frac{F(\alpha_{\text{old}})}{F'(\alpha_{\text{old}})} \tag{4.1}$$

where $F(\alpha)$ is a vector function representing the discrepancy between the computed solution and the desired boundary conditions, and $F'(\alpha)$ is the Jacobian matrix of partial derivatives of $F$ with respect to the parameters. The Matlab code of this root finding method can be found at **??**

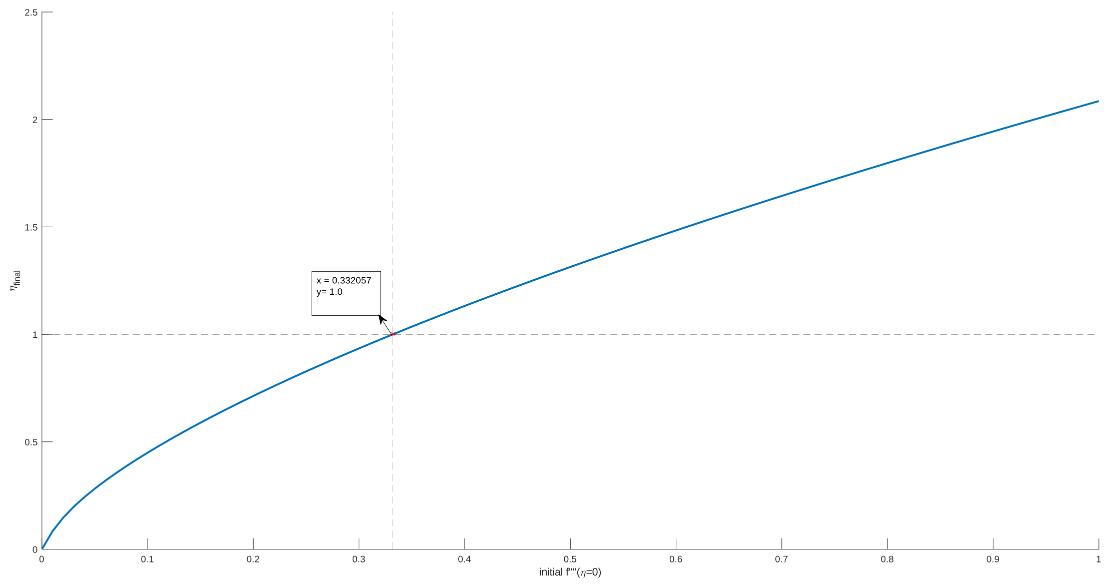

Figure 4.1: $f'(\eta_{max})$ vs the initial values of $G = f''(0)$ (Code A.3.7)

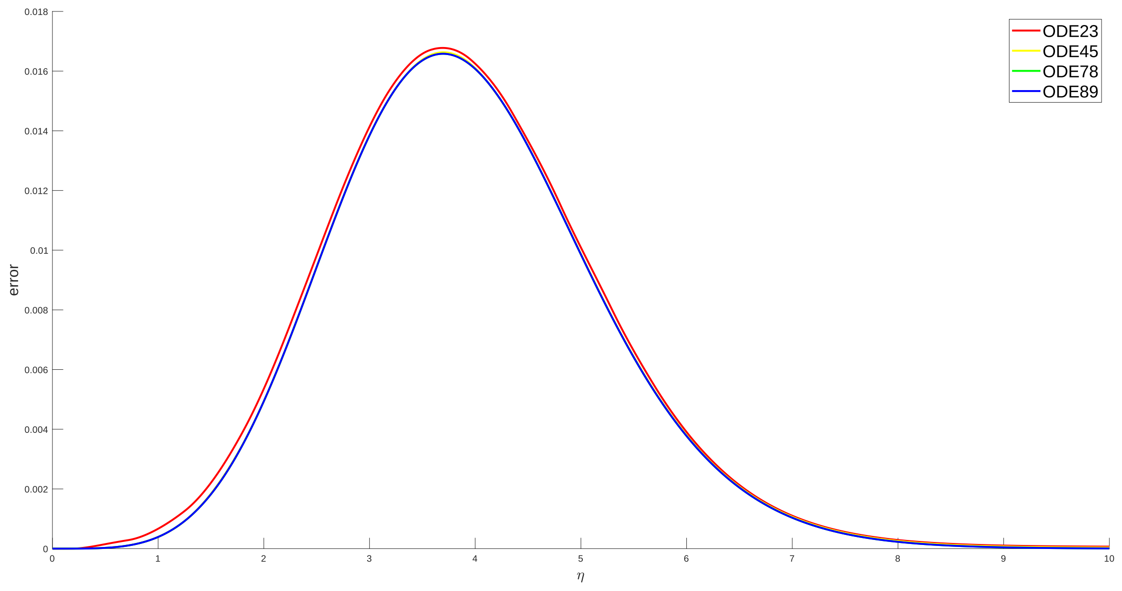

Figure 4.2: difference of different numerical methods from analytical solution with respect to $x$ (Code A.3.8)

## 4.2 Accuracy Assessment and Method Comparison

A set of different ordinary differential equation (O.D.E) solvers is provided by MATLAB. In this project, four methods are utilized and compared to solve the Blasius O.D.E. The solvers employed are as follows:

1. ODE23
2. ODE45
3. ODE78
4. ODE89

By comparing the numerical solutions with the analytical solutions and computing the Root Mean Square Error (RMSE) using Eq. 4.2 for the numerical methods, it is evident that all four methods yield relatively similar and sufficiently accurate results. The detailed table of RMSE values is presented in Table 4.1. Although the RMSE values for the methods are quite similar, ODE78 exhibits a slightly lower RMSE. Consequently, for the subsequent comparisons and visualizations, the solution obtained from ODE78 will be utilized.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}} \tag{4.2}$$

The error plot is depicted by 4.2 and as it can be clearly seen the highest error in the vicinity pf $\eta = 4$ and near the 0 and $\eta_{max}$ the error is nearly zero.

Table 4.1: Root Mean Square Error of numerical O.D.E solving methods

| method | RMSE |
|---|---|
| ODE23 | 0.00781895 |
| ODE45 | 0.00766837 |
| ODE78 | 0.00765809 |
| ODE89 | 0.00765810 |

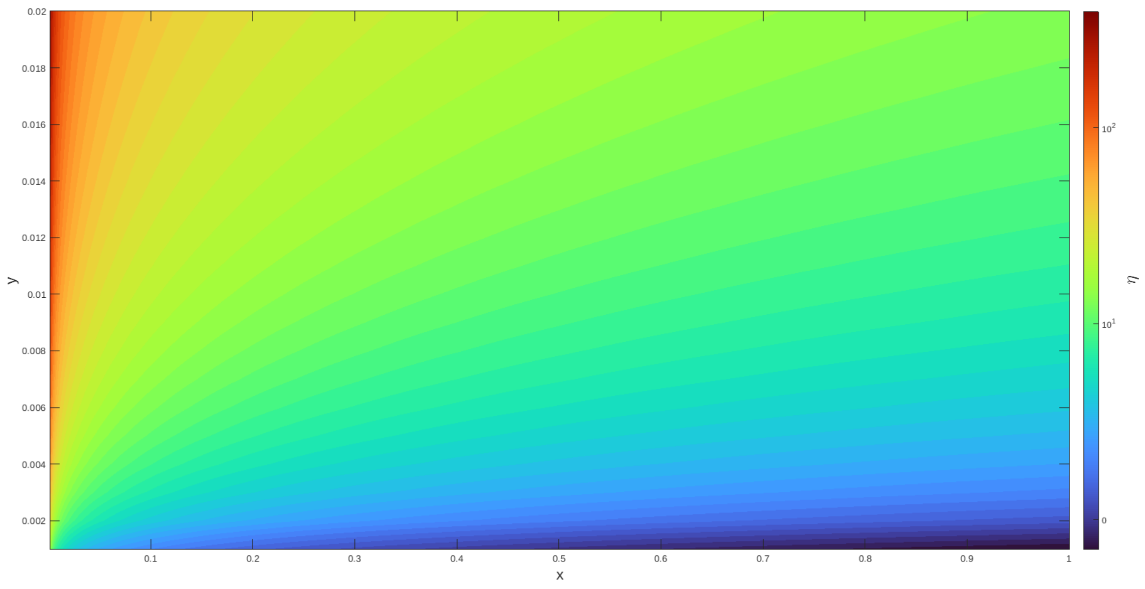

Figure 4.3: $\eta$ values based on $U_e = 50\frac{m}{s}$ and $\nu = 1.416 \times 10^{-7}\frac{m^2}{s}$ (Code A.3.1)

## 4.3 $\eta$ values and O.D.E solution

As demonstrated in Chapter 2, the combination method is employed to transform the partial differential equation (PDE) into an ordinary differential equation (ODE), introducing a new variable $\eta(x, y)$. The values of $\eta$ are computed using Eq. 2.27. In accordance with the given problem, the freestream velocity is $U_e = 50\frac{m}{s}$, and the kinematic viscosity is $\nu = 1.416 \times 10^{-7}\frac{m^2}{s}$. The contour plot in Fig. 4.3 illustrates the mapping of $x$ and $y$ to values of $\eta$ within the specified range.

The solution of the ODE78 has been utilized to compute $f'(\eta)$ and $f''(\eta)$, and the results are depicted in Fig. 4.4.

## 4.4 Finding $\eta_{max}$

Despite the boundary conditions indicating $f'(\eta \to \infty) = 1$, it is practically unattainable to numerically compute $\eta = \infty$. Therefore, a criterion is necessary to determine the value of $\eta$ at that point, denoted as $\eta_{\text{max}}$, and is approximated to be 1. The criterion chosen for this purpose is based on a difference less than $1 \times 10^{-5}$. This stringent condition ensures numerical stability and precision in handling the asymptotic behavior of the solution.

$$\eta_{max} = \eta_i \quad where \quad \frac{\eta_{i+1} - \eta_i}{\eta_i} \leq 10^{-5} \tag{4.3}$$

Using the mentioned criteria $\eta = 6.396396$ had been chosen as $\eta_{max}$. It means that from the $\eta = 6.396396.$, $f'(\eta)$ is approximately 1 and the with increasing of $\eta$ the $f'(\eta)$ changes less than $10^{-5}$.

Table 4.2: $\eta$ with respect to different criteria values

| criteria | $\eta$ | $f'(\eta)$ |
|---|---|---|
| $10^{-3}$ | 3.65365 | 0.92846 |
| $1.6 \times 10^{-4}$ | 5.005 | 0.9916 |
| $10^{-4}$ | 5.27527 | 0.99504 |
| $2.4 \times 10^{-5}$ | 6.006 | 0.99899 |
| $10^{-5}$ | 6.39640 | 0.99961 |
| $2.19 \times 10^{-6}$ | 7.007 | 0.99992 |
| $10^{-6}$ | 7.28729 | 0.99997 |

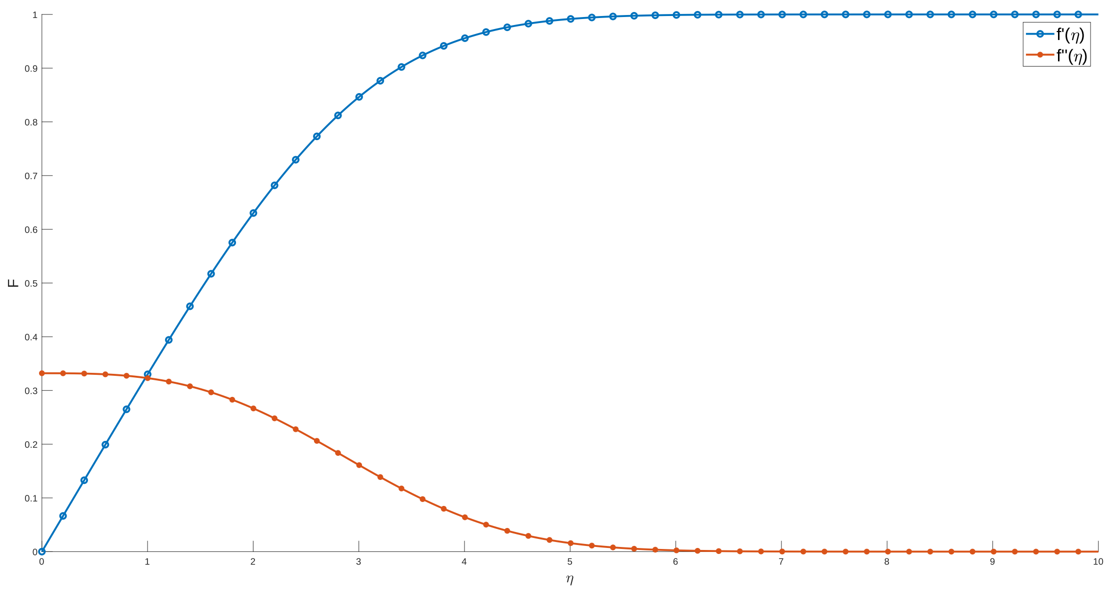

Figure 4.4: Blasius O.D.E solution for $f'(\eta)$ and $f''(\eta)$ (Code A.3.2)

## 4.5 Velocity components

Now, by having $\eta$ values in the given domain of $x$ and $y$, and also by having the mapping solution of $f(\eta)$ and $f'(\eta)$, by substituting the results into Eq.2.28, the values for $u$ and $v$ can be obtained. The contour plots of $u$ and $v$ are shown in Fig. 4.5 and 4.6, respectively. Additionally, the variation of $u$ with respect to $y$ at constant $x$s is illustrated in Fig. 4.7.

## 4.6 Shear Stress, Force and Drag Coefficient

Now, with $f(\eta)$ and its derivatives in hand, the shear stress, total force, and drag coefficient on the plate can be determined using Eq. 2.29. Figure 4.9 illustrates the relationship between shear stress ($\tau$) and $x$. The total force on the plate, assuming a depth of 1, is calculated as $1.0605\,\text{N}$, and the drag coefficient is computed as $3.535 \times 10^{-4}$.

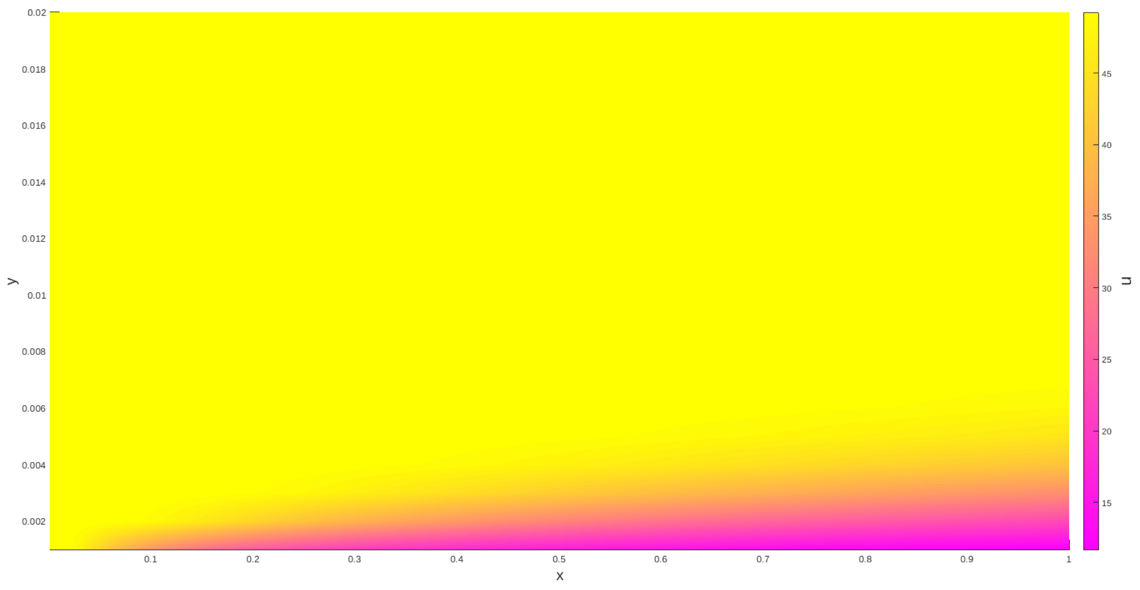

Figure 4.5: $u$ velocity values(Code A.3.3)

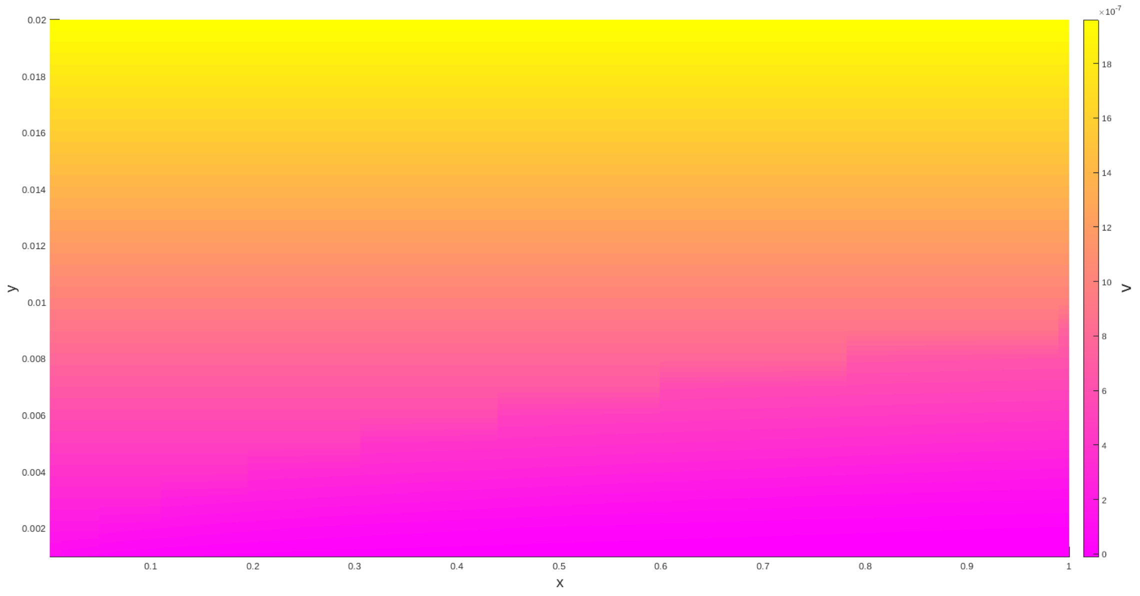

Figure 4.6: $v$ velocity values(Code A.3.4)

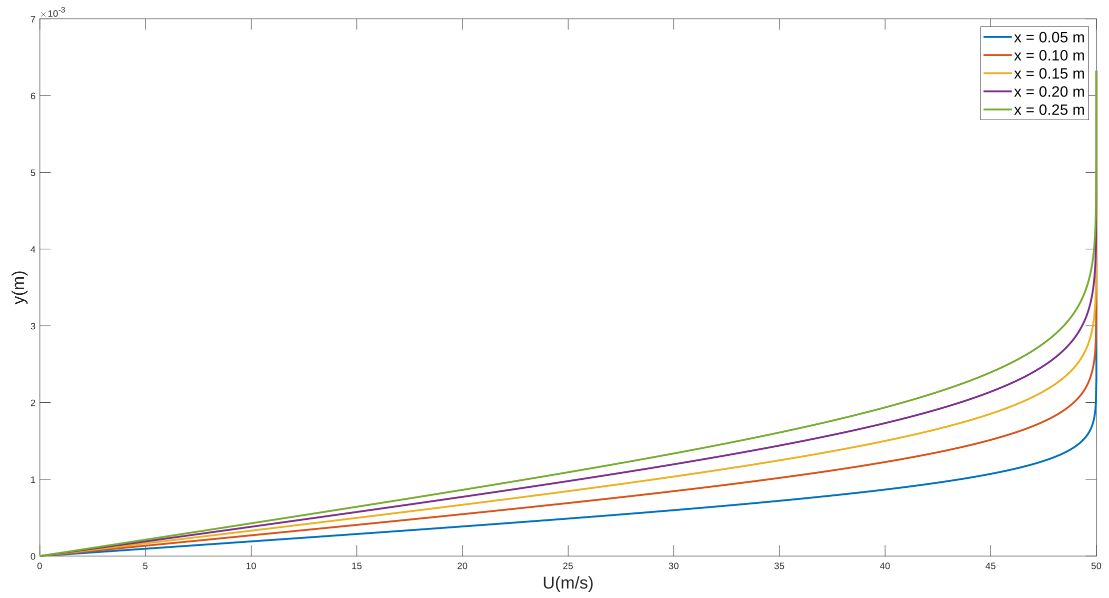

Figure 4.7: $u$ variation with respect to $y$ in constant $x$s(Code A.3.6)

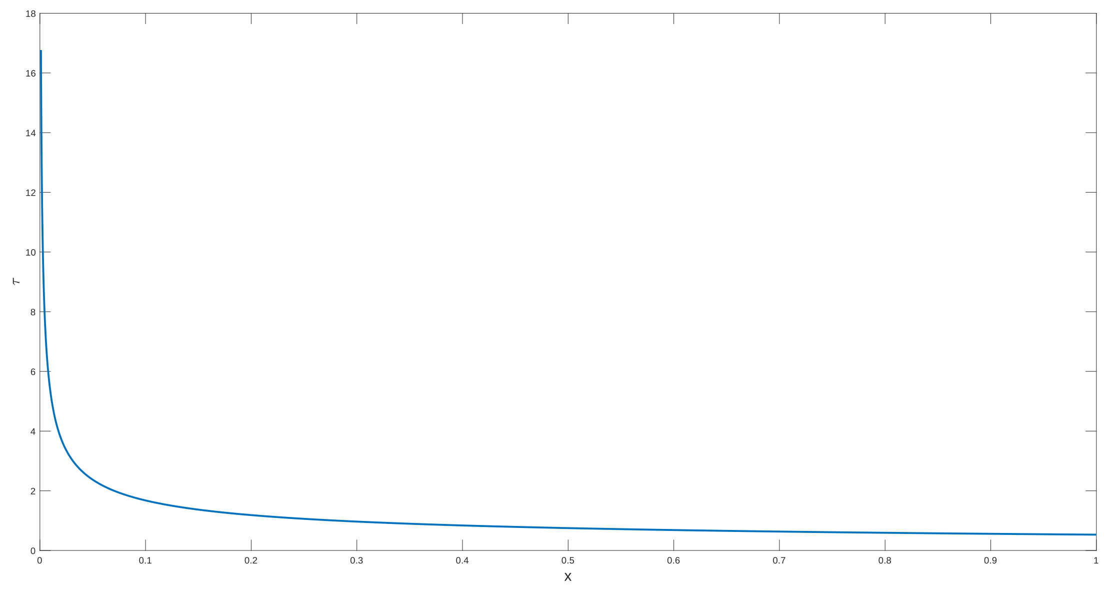

Figure 4.8: Shear stress($\tau$) vs. $x$ (Code A.3.5)

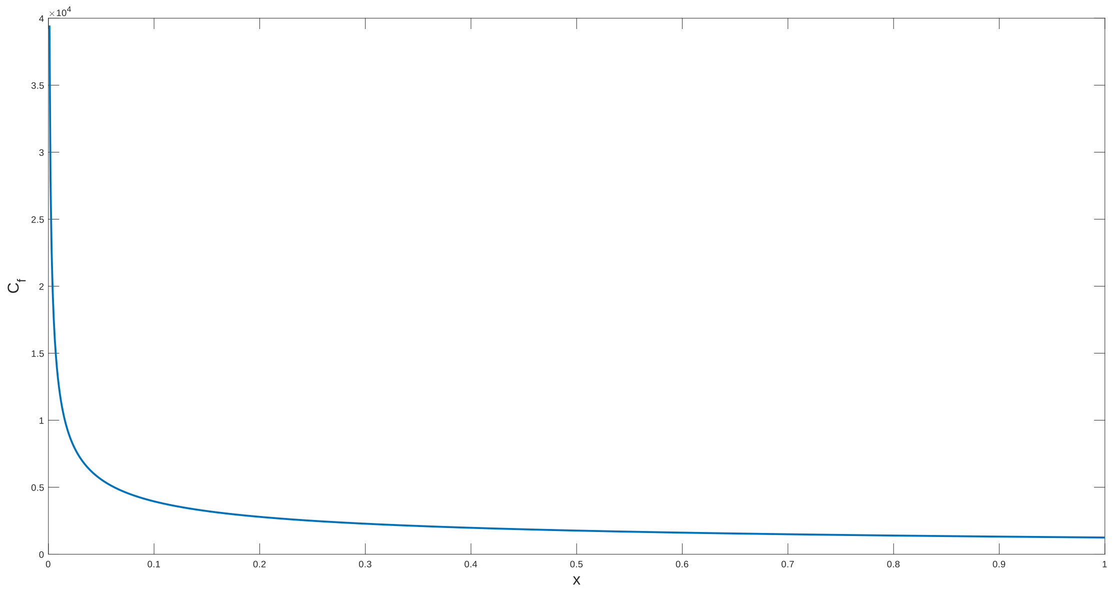

Figure 4.9: friction coefficient along the wall $C_f$ vs. $x$ (Code A.1.6)

# Acknowledgements

This project was undertaken as part of the requirements for ENGR 6201: Fluid Mechanics, lectured by Professor Karimfazli at Concordia University, Montreal, Canada. The report was meticulously prepared using the LaTeX typesetting language on the Overleaf platform. The MATLAB[5] R2022b Linux version was employed for code generation and visualization of graphs. Additionally, schematic diagrams were crafted using draw.io

# Bibliography

[1] D. Marshall, "Blasius Equation Derivation," 2002.

[2] K. P. Burr, T. R. Akylas, and C. C. Me, "Two-Dimensional Laminar Boundary Layers,"

[3] F. Mattioli, *Principles of Fluid Dynamics*, vol. 2. 2008.

[4] Savaş, "An approximate compact analytical expression for the Blasius velocity profile," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, 2012.

[5] T. M. Inc., "Matlab version: 9.13.0 (r2022b)," 2022.

# Appendix A

# Matlab Code

## A.1 functions

### A.1.1 Blasius O.D.E

```matlab
function dydt = blasius(eta, y)
       dy1dt = y(2);
       dy2dt = y(3);
       dy3dt = -0.5 * y(1) * y(3);
       dydt = [dy1dt; dy2dt; dy3dt];
```

### A.1.2 $\eta(x, y)$ function

The following function calculates the dimensionless variable $\eta$ as defined in Eq. 2.27.

```matlab
function eta_value = Eta_func(x,y,U_e, nu)
    nux = nu .* x;
    Unux = U_e .* (nux .^ -1);
    eta_value = y .* sqrt(Unux);
end
```

### A.1.3 finding appropriate $f''(\eta = 0)$

The following function is designed to determine the suitable value for $f''(\eta = 0)$, which satisfies the desired boundary conditions. This function will be used by ‘shootingMethod‘ function.

```matlab
function res = G_find(G)
    [~,f] = ode78(@blasius,[0 10],[0;0;G]);
    res = f(end,2) -1;
end
```

### A.1.4 Shooting Method

The following function is designed to determine the root of the given function to desired tolerance:

```matlab
function root = shootingMethod(inputFunction, initialGuess, tolerance)
    x0 = initialGuess;
    dx = 1e-6;
    maxIterations = 1000;
    for iteration = 1:maxIterations
        f = inputFunction(x0);
        dfdx = (inputFunction(x0 + dx) - f) / dx;
        x0 = x0 - f / dfdx;
        if abs(f) < tolerance
            root = x0;
            return;
        end
    end
```

```
end
```

### A.1.5 Shear Stress

The following function calculates the shear stress using the provided parameters. This function is based on Eq.2.22.

```
function shear = tau(x,U_e,mu, rho,G)
    nu = mu /rho;
    g = sqrt(U_e ./ ( nu * x));
    shear = mu * U_e * g * G;
end
```

### A.1.6 Friction Coefficient

The following function calculates the friction coefficient along the wall using the provided parameters. This function is based on Eq.2.23.

```
function Cf_value = Cf(x ,mu,rho, U_e,G)
    tau_value = tau(x, U_e, mu, rho, G);
    Cf_value = (2* tau_value) / (rho * U_e);
end
```

### A.1.7 Boundary Layer Thickness

The following function computes the boundary layer thickness in the give $x$ based on Eq.2.27.

```
function value = thickness(x,U_e,nu)
    value = sqrt(nu * x / U_e);
end
```

### A.1.8 Analytical Solution

The following function represents Eq.3.1 in Matlab code:

```
function fp = analyitical(eta,a,n)
    fp = (tanh((a*eta).^n)).^(1/n);
end
```

## A.2 Main Code

### A.2.1 Cleaning and defining hyper-paramaters

The following code serves the purpose of cleaning the workspace and setting up essential hyper-parameters and constant values for the subsequent numerical simulation.

```
%% cleaing
close all;clear;clc
format long


%% Setting hyperparameters and constant values
L = 1;
H = 0.02;

dy = 0.0001;
dx = 0.001;

```

```
13 x = 0.001:dx:L;
14 y = 0.001:dy:H;
```

If the $u$ velocity component along the $y$-axis at a specific $x$ is required, the following code can be used in place of lines 13 and 14 of the previous code:

```
x= [0.05,0.1,0.15,0.2,0.25];
H = tickness(x,U_e,nu);
y = 0:H/5000:H*20;
```

### A.2.2 Compute $\eta$ values

The following code is to generate the meshgird and compute $\eta$ values using function A.1.2.

```
%% Generating meshgrid and computing eta
[X,Y] = meshgrid(x,y);
eta = Eta_func(X,Y,U_e,nu);
```

### A.2.3 Finding the $f''(\eta = 0)'$

The following command finds the appropriate $f''(\eta = 0)$ which satisfies the boundary conditions with root finding function 'shootingMethod'. The 'G_find' function is defined in A.3.7.

```
tol = 0.00001;
G = shootingMethod(@G_find,1,tol)
```

### A.2.4 Solving the Blasius Equation

Different Solutions generated by the following code, the 'blasius' and 'analytical' functions are defined by A.1.1 and A.1.8, respectively :

```
%% solivng with different methods
sol23 = ode23(@blasius,[0 10],[0;0;G]);
sol45 = ode45(@blasius,[0 10],[0;0;G]);
sol78 = ode78(@blasius,[0 10],[0;0;G]);
sol89 = ode89(@blasius,[0 10],[0;0;G]);

t = linspace(0,10,1000);

f23 = deval(sol23,t);
f45 = deval(sol45,t);
f78 = deval(sol78,t);
f89 = deval(sol89,t);

f_analytic = analyitical(t,G,3/2);
```

### A.2.5 Root Mean Square Error

The following code is utilized to compute Root Mean Square Error defined by Eq.4.2:

```
%% compute root mean squeres
RMSE23 = sqrt(mean((f23(2,:) - f_analytic).^2));
RMSE45 = sqrt(mean((f45(2,:) - f_analytic).^2));
RMSE78 = sqrt(mean((f78(2,:) - f_analytic).^2));
RMSE89 = sqrt(mean((f89(2,:) - f_analytic).^2));
```

### A.2.6 Finding $\eta_{max}$

By varying 'criteria' in the followig code the table 4.2 is generated:

```
%% finding the eta_max
criteria = 0.000001;
index_eta_max = find(diff(f89(2,:))./f89(2,2:end)<criteria,1);
eta_max = t(index_eta_max);
```

### A.2.7 Computing $u$,$v$ velocity components

To create a function from the solution of ODE78 to be able to predict new values if needed, 'interp1' Matlab was used. to increase the accuracy the solution was divided to two parts. the first par is in the range of $0 < \eta < \eta_{max}$ which $f'(\eta)$ rises from 0 to 1 and the second part represents $\eta_{max} < \eta$ in which $f'(\eta)$ is approximately 1.

```
%% Computing u and v
fp1 = eta > eta_max;

F = f89(2,1:index_eta_max);
T = t(1:index_eta_max);

fp2 = zeros(size(eta,1), size(eta,2));
f = zeros(size(eta,1), size(eta,2));
for i=1:1:size(eta,2)
    fp2(:,i) = interp1(T,F,eta(:,i)) .* (eta(:,i) < eta_max);
    f(:,i) = interp1(T,F,eta(:,i));
end

fp1(isnan(fp1))=0;
fp2(isnan(fp2))=0;
f(isnan(f))=0;

fp = fp1 + fp2;

u = U_e .* fp;
v = nu .* thickness(x,U_e,nu) .* (eta .* fp - f);
```

### A.2.8 Shear Stress, Force and Drag Coefficient

```
%% shear stress
taus = tau(x,U_e,1.7*(10^-5),1.2,G);

fun = @(nx) tau(nx,U_e,1.7*(10^-5),1.2,G);

q = integral(fun,0.00,1);
C_D = q/(U_e^2 * rho)
```

## A.3 Plots

### A.3.1 Contour of $\eta$ values

Contour of $\eta$ values with respect to $x$ and $y$

```
%% ploting eta values
contourf(x,y,log10(eta),50,'edgecolor','none');

cb = colorbar('Ticks',[0,1,2,3,4],'TickLabels',{'0', '10^1','10^2','10^3','10^4'
    });
```

```

cb.Label.String = '\eta';
cb.Label.FontSize=18;

colormap turbo;
xlabel('x','FontSize',16);
ylabel('y','FontSize',16);
```

### A.3.2 Plot of $f'(\eta)$ and $f''(\eta)$

```
%% ploting f, fp

hold on
plot(t,f89(2,:),'-o','MarkerIndices',1:20:length(f89(2,:)),'LineWidth',2,'
    DisplayName','f''(\eta)');
plot(t,f89(3,:),'-*','MarkerIndices',1:20:length(f89(3,:)),'LineWidth',2,'
    DisplayName','f''''(\eta)');

xlabel('\eta','FontSize',16);
ylabel('F','FontSize',16);

lgd = legend();
lgd.FontSize = 18;
```

### A.3.3 Contour of $u$ velocity component

```
%% Plotting u
hold on
contourf(x,y,u,50,'edgecolor','none');

cb = colorbar;
cb.Label.String = 'u';
cb.Label.FontSize=18;

colormap spring;

xlabel('x','FontSize',16);
ylabel('y','FontSize',16);
```

### A.3.4 Contour of $v$ velocity component

```
%% Plotting v
hold on
contourf(x,y,v,50,'edgecolor','none');

cb = colorbar;
cb.Label.String = 'v';
cb.Label.FontSize=18;

colormap spring;

xlabel('x','FontSize',16);
ylabel('y','FontSize',16);
```

### A.3.5 Plot Shear Stress on the plate

```matlab
%% shear stress
taus = tau(x,U_e,1.7*(10^-5),1.2,G);

plot(x,taus,'LineWidth',2);

xlabel('x','FontSize',16);
ylabel('\tau','FontSize',16);
```

### A.3.6 Plot $u$ velocity at specific positions

Plotting $u$ velocity component along $y$-axis at specific $x$ positions

```matlab
plot(u,y,'LineWidth',2)
legend('x = 0.05 m','x = 0.10 m', 'x = 0.15 m', 'x = 0.20 m','x = 0.25 m','
    FontSize',16)
xlabel('U(m/s)',"FontSize",18);
ylabel("y(m)","FontSize",18);
```

### A.3.7 Plotting $f'(\eta_{max})$ vs. $f''(0)$

```matlab
x = linspace(0,1,100);
Y = zeros(1,100);

for i=1:size(x,2)
    Y(1,i) = G_find(x(i))+ 1;
end

hold on
plot(x,Y,'LineWidth',2);
plot(0.332057334445899,1,'r*')

xline(0.332057334445899,'--');
yline(1,'--')

xlabel("initial f''''(\eta=0)");
ylabel("\eta_{final}")
```

### A.3.8 Plotting numerical solution errors

```matlab
%% ploting errors
hold on
plot(t,(f23(2,:)-f_analytic),'r-','DisplayName','ODE23','LineWidth',2);
plot(t,(f45(2,:)-f_analytic),'y-','DisplayName','ODE45','LineWidth',2);
plot(t,(f78(2,:)-f_analytic),'g-','DisplayName','ODE78','LineWidth',2);
plot(t,(f89(2,:)-f_analytic),'b-','DisplayName','ODE89','LineWidth',2);

lgd = legend();
lgd.FontSize = 18;

%title("Numerical methods error in comparison to Analyitcal Solution",'FontSize
    ',18)
xlabel('\eta','FontSize',16);
ylabel('error','FontSize',16);
```