fluids

MDPI

Article

A CFD Tutorial in Julia: Introduction to Laminar Boundary-Layer Theory

Furkan Oz * and Kursat Kara

School of Mechanical and Aerospace Engineering, Oklahoma State University, Stillwater, OK 74078, USA; kursat.kara@okstate.edu

* Correspondence: foz@okstate.edu

Abstract: Numerical simulations of laminar boundary-layer equations are used to investigate the origins of skin-friction drag, flow separation, and aerodynamic heating concepts in advanced undergraduate- and graduate-level fluid dynamics/aerodynamics courses. A boundary-layer is a thin layer of fluid near a solid surface, and viscous effects dominate it. Students must understand the modeling of flow physics and implement numerical methods to conduct successful simulations. Writing computer codes to solve equations numerically is a critical part of the simulation process. Julia is a new programming language that is designed to combine performance and productivity. It is dynamic and fast. However, it is crucial to understand the capabilities of a new programming language before attempting to use it in a new project. In this paper, fundamental flow problems such as Blasius, Hiemenz, Homann, and Falkner-Skan flow equations are derived from scratch and numerically solved using the Julia language. We used the finite difference scheme to discretize the governing equations, employed the Thomas algorithm to solve the resulting linear system, and compared the results with the published data. In addition, we released the Julia codes in GitHub to shorten the learning curve for new users and discussed the advantages of Julia over other programming languages. We found that the Julia language has significant advantages in productivity over other coding languages. Interested readers may access the Julia codes on our GitHub page.

Keywords: CFD; Julia; Blasius; Hiemenz; Homann; Falkner–Skan; boundary-layer

Citation: Oz, F.; Kara, K. A CFD Tutorial in Julia: Introduction to Laminar Boundary-Layer Theory. *Fluids* **2021**, *6*, 207. https://doi.org/10.3390/fluids6060207

Academic Editor: Ashwin Vaidya

Received: 13 May 2021
Accepted: 28 May 2021
Published: 3 June 2021

1. Introduction

Computational fluid dynamics (CFD) simulation is one of the vital steps of the design of a product that includes fluid motion. Since fluid dynamics are extremely complex and the motion equations have nonlinear terms, usage of numerical approaches is inevitable to simulate or predict fluid motion. Predictions can also be done with experiments. However, it is usually costlier than a regular CFD simulation. On the other hand, fundamental knowledge about the flow, that is planned to simulate, is necessary because of the required numerical approach decision. Learning the canonical flows well is extremely important in this point because most of the complex flow consists of a combination of a couple of canonical flows. An airfoil CFD simulation can consist of boundary-layer flow over the smooth part of the airfoil, mixing layer flow where the tail ends, and blunt body flow in the wake region. One airfoil simulation consists of three different canonical flows. A full understanding of the canonical flows is extremely important to model an accurate airfoil simulation.

Prandtl [1] stated that some of the terms in the Navier–Stokes equations can be neglected for the boundary-layer flows. As a result of this assumption, well-known boundary-layer equations arose. These approaches are still valid after a century and the resultant system of equations inspired lots of researchers in their studies. One of them was Blasius who is the Ph.D. student of Prandtl. Blasius [2] worked on the same problem as Prandtl did. However, he aimed to overcome the enigma of turbulence by

considering the phenomenon of boundary-layer flow explained by Hager [3]. He further simplified the boundary-layer equations for a flat plate. He assumed that the flow is parallel. In other words, the velocity component in the parallel direction is not zero and the velocity in the transverse direction is zero. Moreover, he represented the resultant system of equations with a third-order ordinary differential equation, which is known as the Blasius similarity solution.

The Falkner–Skan similarity solution is another laminar similarity solution. Cebeci [4] explains the Falkner–Skan equations named after V. M. Falkner and Sylvia W. Skan. Falkner and Skan generalized the Blasius similarity solution for non-parallel flows, such as wedge flows and corner flows. The resultant equation can be used to predict the boundary-layer thickness for wedge flow. It has to be noted that these assumptions are available in the laminar region. Once the turbulence occurs, both of these similarity solutions will be inaccurate. In the incompressible region, the Falkner–Skan equation can be used without additional equation; however, after the compressibility limit, the temperature effect must be introduced to the system as an additional equation. If the Falkner–Skan equation is solved with the energy equation, the boundary-layer profile can be obtained in the compressible region.

One of the other Ph.D. students of Prandtl is Karl Hiemenz who worked on Hiemenz flow which is a type of stagnation point flow. Hiemenz [5] formulated and calculated the stagnation point problem as explained in the Schlichting [6]. The problem was a special case of the Falkner–Skan similarity solution. Howarth [7] also worked on the same problem and concluded similar results. Another Ph.D. student of Prandtl, Fritz Homann [8] worked on the same problem for axisymmetric bodies as Schlichting [6] explains. Homann's similarity formulation was for a sphere while Hiemenz's similarity formulation was for a cylinder. Both similarity solutions are being widely used for stagnation flows.

The aforementioned papers are the origins of the boundary-layer theory. As it is mentioned, the boundary-layer theory is the origin of many problems, such as the laminar to turbulent boundary-layer transition and flow separation. The main concern of these researches is to increase the performance of the vehicle because the transition increases the heat transfer and the vehicle requires a better thermal protection system at high speeds due to increased heat transfer rate. On the other hand, flow separation may lead to a lift force loss on the wing as a result, the performance of aircraft decreases. Since the focus of the present paper is the fundamentals of laminar boundary-layer theory, the details will not be provided about advanced researches. However, readers who are interested in details may check [9–12] for subsonic boundary-layer transition, Ref. [13–21] for supersonic/hypersonic boundary-layer transition, and Ref. [22–24] for flow separation. The other researches where boundary-layer flow is involved are [25–30].

Understanding the aforementioned fundamental flows are crucial for a senior undergraduate student or a graduate student in order to simulate more complex flows. However, modeling these equations in a computer environment requires more than knowledge about canonical flows instead it requires knowledge about programming languages as well. There are several learning modules and papers [31–35] for computational fluid dynamics simulation coding; however, there is not enough publication for Julia language [36]. It is a relatively new coding language among the other coding languages such as Fortran, C/C++, Python, and MATLAB but it is getting popular fast because it is trying to fill the gap between the language of the state-of-art CFD codes, Fortran and C/C++, and straightforward/user-friendly languages, Python and MATLAB. Most of the state-of-art CFD codes are written in Fortran and C/C++ because it is so fast and random access memory usage (RAM) can be reduced drastically. On the other hand, MATLAB and Python have user-friendly and easy syntax which makes them favorites of students, with a price, which is the speed of computation. Julia is a language that tries to fill this gap between two different coding styles. It has a user-friendly environment, while also being fast. It provides a working space that helps users to write clear, high-level, generic code that resembles

mathematical formulas. Julia's ability to combine high-performance with productivity makes it a great choice for researchers working in different areas.

In this paper, fundamental flow problems such as Hiemenz flow, Homann flow, Blasius flow, and Falkner–Skan flow will be derived from scratch and modeled in the Julia environment. The finite-difference discretization in space with Thomas algorithm as linear system solver is used. Outputs of the code are provided and they are compared with the literature. Additionally, the advantages of the Julia language over other languages will be discussed. The computer codes and the implementation instructions will help students to understand the fundamental flows which will provide insight for student's course work and researches. Using these examples, they can solve more complex flow types and develop their own codes. We make all these codes available on GitHub and they are accessible to everyone. We provide installation instruction for Julia and the required packages in Appendix A. The GitHub link of the codes also can be found in Appendix A.

2. Laminar Boundary-Layer Theory

Understanding canonical flows is crucial to understanding more complex flows such as flow over a wing or an airfoil. Hosseini et al. [37] studied flow over a wing section with a direct numerical simulation (DNS) study. They created a great video about how flow is formed over the aircraft. The video shows the development of the boundary-layer, how the trailing edge looks like a mixing-layer flow, and how Karman vortex street type of wake structures occurs. If the pre-study work is examined, it can be seen that the mesh structure is built on the flow prediction. Using a finer mesh on the critical regions is a key point of the CFD and it requires predictions about the possible flow behavior. The boundary-layer is the origin of many engineering problems in aerodynamics, including wing stall, the skin friction drag on an object, and the heat transfer that occurs in high-speed flight. In this present paper, boundary-layer theory will be examined under two subsections, which are laminar boundary-layer problems and stagnation point problems. In this present paper, fundamental fluid dynamics problems related to boundary-layer theory will be derived and implemented in a relatively new programming language, Julia. The contribution is employing the Julia language. The governing equations and solution methodologies are already published in the literature. The interested reader should refer to the additional references [6,38] for detailed derivations. The manuscript may enable students to adopt the programming language easily.

2.1. Laminar Boundary-Layer Flow Problems

Velocity distribution over a flat plate can be represented with a similarity solution. In this subsection, Blasius and Falkner–Skan similarity solutions will be derived from scratch and they will be solved numerically in the Julia environment. The Julia codes will be available to shorten the learning curve.

2.1.1. Blasius Flow Problem

Schematic description of flow over a flat plate, in other words, Blasius boundary-layer flow can be illustrated as in Figure 1. The governing equations for Blasius flow are boundary-layer equations. These equations can be obtained by non-dimensionalizing the two-dimensional Navier–Stokes equations that are:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2}$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \tag{3}$$

where u and v are the velocities in the x-direction and y-direction, respectively. p is the pressure, ν is the kinematic viscosity, ρ is the density. If Equations (1)–(3) are non-dimensionalized with:

$$u^* = \frac{u}{U_\infty}, \quad v^* = \frac{v}{U_\infty}, \quad p^* = \frac{p}{\rho U_\infty^2}, \tag{4}$$

$$x^* = \frac{x}{L}, \quad y^* = \frac{y}{L}, \tag{5}$$

where L is the plate length and U_∞ is the free-stream velocity. The two-dimensional, incompressible non-dimensional Navier–Stokes equations can be shown as:

$$\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} = 0 \tag{6}$$

$$u^*\frac{\partial u^*}{\partial x^*} + v^*\frac{\partial u^*}{\partial y^*} = -\frac{\partial p^*}{\partial x^*} + \frac{1}{Re_\infty}\left(\frac{\partial^2 u^*}{\partial x^{*2}} + \frac{\partial^2 u^*}{\partial y^{*2}}\right) \tag{7}$$

$$u^*\frac{\partial v^*}{\partial x^*} + v^*\frac{\partial v^*}{\partial y^*} = -\frac{\partial p^*}{\partial y^*} + \frac{1}{Re_\infty}\left(\frac{\partial^2 v^*}{\partial x^{*2}} + \frac{\partial^2 v^*}{\partial y^{*2}}\right). \tag{8}$$

Star superscript ($[\ ]^*$) corresponds to non-dimensional variable. In a boundary-layer flow, some variables are smaller than others. For example, $u^* = O(1)$ and $x^* = O(1)$ so $\frac{\partial u^*}{\partial x^*} = O(1)$. In the continuity equation, two terms must be in the same order to obtain 0 so $\frac{\partial v^*}{\partial y^*} = O(1)$. It can be seen that from Figure 1, $y = O(\delta^*)$ where δ^* is the non-dimensionalized boundary-layer thickness. From here, it can be concluded that $v = O(\delta^*)$. It has to be noted that $\delta^* << 1$. If the same approach is applied to momentum equations, the final system of equations will be:

$$\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} = 0 \tag{9}$$

$$u^*\frac{\partial u^*}{\partial x^*} + v^*\frac{\partial u^*}{\partial y^*} = -\frac{\partial p^*}{\partial x^*} + \frac{1}{Re_\infty}\frac{\partial^2 u^*}{\partial y^{*2}} \tag{10}$$

$$\frac{\partial p^*}{\partial y^*} = 0. \tag{11}$$

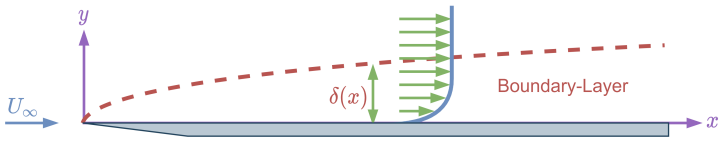

Figure 1. Schematic description of the flow over a flat plate.

For an inviscid flow, where the Reynolds number is so high, the viscous term can be neglected. The pressure is constant as obtained in the Equation (11) and $v = 0$. Equation (10) will be:

$$-\frac{dP_e^*}{dx^*} = U_e^*\frac{dU_e^*}{dx^*}, \tag{12}$$

where U_e and P_e are the inviscid velocity and pressure. Finally, the two-dimensional boundary-layer equations in the dimensionless form can be written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{13}$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = U_e\frac{dU_e}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2}. \tag{14}$$

It has to be noted that star superscript is suppressed to avoid confusion. In order to solve this problem, similarity transformation can be used. The similarity parameter (η) can be chosen as:

$$\eta = \frac{y}{\sqrt{\frac{\nu x}{U_\infty}}}. \tag{15}$$

After that, velocity components can be shown as:

$$u(x,y) = U_\infty F(\eta) \tag{16}$$

$$v(x,y) = \frac{G(\eta)}{\sqrt{\frac{x}{U_\infty \nu}}}. \tag{17}$$

If Equations (13) and (14) are rearranged with the given velocities, the boundary-layer equations will be:

$$G' = \frac{\eta}{2} F' \tag{18}$$

$$F'' = -\frac{1}{2}\eta F F' + G F'. \tag{19}$$

In order to make the next procedure easier, it can be assumed that $F = f'$. After that, Equation (18) can be integrated and put into Equation (19). The final ordinary differential equation can be obtained as:

$$2f''' + f f'' = 0. \tag{20}$$

The boundary conditions of the system will be:

$$u(x,0) = U_\infty F(\eta = 0) = 0 \longrightarrow F(0) = f'(0) = 0 \tag{21}$$

$$v(x,0) = \frac{G(\eta = 0)}{\sqrt{\frac{x}{U_\infty \nu}}} = 0 \longrightarrow G(0) = f(0) = 0 \tag{22}$$

$$u(x, y \to \infty) = U_\infty F(\eta \to \infty) = 1 \longrightarrow F(\infty) = f'(\infty) = f(\infty) = 1. \tag{23}$$

The given assumptions reduced the system of second-order partial differential equations into a third-order ordinary differential equation. The computational approach will be examined in the Julia framework.

2.1.2. Numerical Solution of Blasius Flow Problem

In this computational section, the solution of the third-order ordinary differential equation will be transformed into two equations and they will be solved with Thomas algorithm [39] which is one of the best methods for the tridiagonal matrices. It has to be noted that the same equations can be solved with any other methods such as Runge-Kutta, Runge-Kutta-Fehlberg, compact finite difference, high-order finite-difference; however, in this present paper, authors used the finite difference method for spatial discretization with the Thomas algorithm for the linear system solution. If it is assumed that $f' = h$ and $f = p$ in Equation (20), the system of equations will be:

$$2h'' + ph' = 0 \tag{24}$$

$$p' - h = 0, \tag{25}$$

with the following boundary conditions:

$$p(0) = 0 \tag{26}$$

$$h(0) = 0 \tag{27}$$

$$h(\infty) = 1. \tag{28}$$

If the second-order central finite difference method is applied to h variable:

$$h''_n = \frac{h_{n+1} - 2h_n + h_{n-1}}{(\Delta\eta)^2} + O((\Delta\eta)^2) \tag{29}$$

$$h'_n = \frac{h_{n+1} - h_{n-1}}{2\Delta\eta} + O((\Delta\eta)^2). \tag{30}$$

In this paper, details of the finite difference and derivation of it from Taylor's series will not be covered. If the reader is curious about the derivation of them, the book of Moin [40] can be checked. If the finite difference approach for h is substituted into the Equation (24), the final equation will be:

$$A_n h_{n+1} + B_n h_n + C_n h_{n-1} = 0, \tag{31}$$

where the A_n, B_n, and C_n are:

$$A_n = \frac{2}{\Delta\eta^2} + \frac{p_n}{2\Delta\eta} \tag{32}$$

$$B_n = \frac{-4}{\Delta\eta^2} \tag{33}$$

$$C_n = \frac{2}{\Delta\eta^2} - \frac{p_n}{2\Delta\eta}, \tag{34}$$

with the boundary conditions for the system the tridiagonal system ($Ax = b$) can be shown as:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ A_2 & B_2 & C_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ 0 & 0 & A_{N-2} & B_{N-2} & C_{N-2} & 0 \\ 0 & 0 & 0 & A_{N-1} & B_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{N-2} \\ h_{N-1} \\ h_N \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{b} \tag{35}$$

It has to be noted that the first and last rows of the matrix are known and they are coming from the boundary conditions. In order to solve this problem, the Thomas algorithm can be applied. If relation between h_n and h_{n-1} is:

$$h_n = G_n + H_n h_{n+1} \tag{36}$$

$$h_{n-1} = G_{n-1} + H_{n-1} h_n. \tag{37}$$

If Equation (37) is substituted into Equation (31), the final equation will be:

$$h_n = \frac{-C_n G_{n-1}}{B_n + C_n H_{n-1}} + \frac{-A_n}{B_n + C_n H_{n-1}} h_{n+1}. \tag{38}$$

If Equations (36) and (38) are compared, G_n and H_n coefficients can be found as:

$$G_n = \frac{-C_n G_{n-1}}{B_n + C_n H_{n-1}} \tag{39}$$

$$H_n = \frac{-A_n}{B_n + C_n H_{n-1}}. \tag{40}$$

The problem here is that one has to know G_1 and H_1 in order to start the calculation. H_1 can be assumed as 0 and the G_1 can be calculated as 0 from the boundary conditions. Once G_1 and H_1 are calculated, G_n and H_n can be calculated from Equations (39) and (40). The given formulations can be implemented as it is shown in Listing 1.

The problem in Listing 1 is the usage of the p and h values which are not defined yet. In order to start to iteration, the initial assumption for h must be given. Most of the time, the linear assumption is the best assumption. Figure 2 shows the schematic of the profile transformation after some iterations. Once the initialization of h is done, p can be calculated from the integration of $p' = h$. The discrete integration of p can be written as:

$$p_n = p_{n-1} + \int_{\eta_{n-1}}^{\eta_n} h d\eta. \tag{41}$$

Listing 1. Implementation of Thomas algorithm for Blasius profile.

```
A = [ 2/Δη^2 + p[i]/(2*Δη) for i=1:N]
B = [−4/Δη^2                for i=1:N]
C = [ 2/Δη^2 − p[i]/(2*Δη) for i=1:N]
D = [ 0 for i=1:N]

for i=2:N−1
G[i] = − ( C[i]*G[i−1] + D[i] )/(B[i] + C[i] * H[i−1])
H[i] = −                 A[i]  /(B[i] + C[i] * H[i−1])
end

for i=N-1:-1:2
h[i] = G[i] + H[i] * h[i+1]
end
```

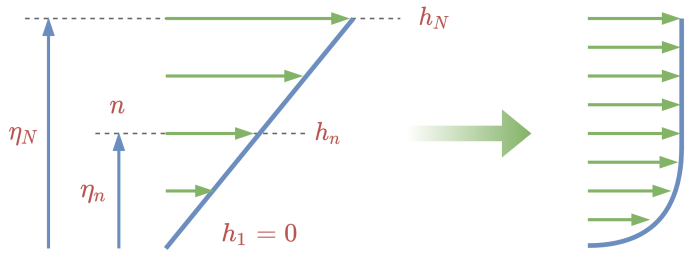

Figure 2. Schematic description of the change in the velocity profile from initial guess to final solution.

If the integration is done numerically with the trapezoidal rule [41], the initial guess for h and p can be calculated in Julia as shown in Listing 2

After the initialization is done, Listing 1 can be run with p calculation of Listing 2, until the change in h is smaller than an arbitrary parameter ϵ which can be taken as 1×10^{-8}. The final profile will be as shown in Figure 3. The profile is also compared with Schlichting [6] in order to validate the results. It has to be noted that, Schlichting used $\eta = \frac{y}{\sqrt{\frac{2\nu x}{U_\infty}}}$ as similarity coordinate so the figure is plotted according to this nondimensionalization.

Listing 2. Implementation of initial velocity guess and initial p for Blasius profile.

```
h = [(i−1)/(N−1)  for i=1:N]

for i = 2:N
p[i] = p[i−1] + (h[i] + h[i−1])*Δη/2
end
```

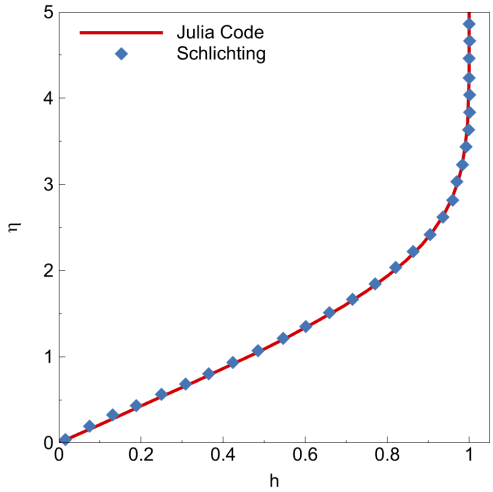

Figure 3. The velocity distribution of the Blasius similarity solution obtained by given Julia code and data digitized from Schlichting [6].

2.1.3. Falkner–Skan Flow Problem

The Falkner–Skan similarity solution can be considered as a family of the similarity solutions since it is a general solution that includes Blasius flow, Hiemenz flow (see Section 2.2.1), and more. Falkner–Skan equation cannot represent Homann flow (see Section 2.2.3) because Homann flow is an axisymmetric flow. On the other hand, Falkner–Skan flow is a two-dimensional flow. The name 'similarity solution' arises from the solutions at two arbitrary stations which are related to one another by means of a scale factor. A flow that can be represented with boundary-layer Equations (13) and (14), and satisfies the following equation:

$$\frac{u(x_1), \frac{y}{g(x_1)}}{U_e(x_1)} = \frac{u(x_2), \frac{y}{g(x_2)}}{U_e(x_2)}, \tag{42}$$

can be considered as self-similar. If boundary-layer Equations (13) and (14) are considered and similarity a transformation is assumed as:

$$u(x,y) = U_e(x) f'(\xi), \quad \xi = \frac{y}{g(x)}. \tag{43}$$

If the continuity equation of boundary-layer Equation (13) is modified with the given transformation, the final equation will be:

$$\frac{\partial v}{\partial \xi} = -U_e' f' g + U_e f'' g' \xi. \tag{44}$$

If Equation (44) is integrated over ξ to find the velocity v, the v velocity will be:

$$v(x,\xi) = -(U_e g)' f + U_e \xi g' f' + H(x). \tag{45}$$

$H(x)$ can be calculated from the boundary conditions on the wall where ξ is zero and correspondingly, velocities and $f'(0)$ are zero. $H(x)$ will be 0 if $f(0)$ is chosen as 0. If calculated derivatives and velocities are substituted into Equation (14), after some simplification, the final equation will be:

$$f''' + \frac{g}{\nu}(U_e g)' f f'' + \frac{g^2 U_e'}{\nu}(1 - f') = 0. \tag{46}$$

Since f must be the function of ξ only and must not be a function of x, the coefficients of second and third terms must be constant. The final Falkner–Skan equation for the family of self-similar solutions:

$$f''' + \alpha f f'' + \beta(1 - f'^2) = 0, \tag{47}$$

where $\alpha = \frac{g}{\nu}(U_e g)'$ and $\beta = \frac{g^2 U_e'}{\nu}$. The α and β can be further solved for the velocity scale and length scale. If the following consideration is applied:

$$\alpha^* = g(U_e g)' \tag{48}$$

$$\beta^* = g^2 U_e'. \tag{49}$$

If $(U_e g^2)'$ is written in terms of α^* and β^*, the obtained equation integrated with respect to x, the resultant equation will be:

$$U_e g^2 = (2\alpha^* - \beta^*)x + c. \tag{50}$$

The constant of the integration represents a shift in the origin on x. Hence it doesn't affect the result and it also can be calculated from the stagnation point where $x = 0$ and $U_e = 0$ as a result, $c = 0$. If relation of $\beta^* = g^2 U_e'$ is divided by Equation (50) and integrated with respect to x, U_e can be calculated as:

$$U_e = c_1 x^m, \tag{51}$$

where $m = \frac{\beta^*}{2\alpha^* - \beta^*}$ and c_1 is a positive or negative constant which depends on the sign of U_e. It can be concluded from these calculations that similar solutions exist when the inviscid velocity is proportional to x raised to some power. Next, Equation (50) can be used by taking $c = 0$ to calculate the g which is:

$$g = \sqrt{\frac{2\alpha^*}{c_1(1+m)} x^{1-m}}. \tag{52}$$

Self-similar boundary-layers occur when the external velocity is the simple power law ($U_e = U_0(x/L)^m$), where the arbitrary constants U_0 and L have the same sign as U and x. The similarity variable for these kinds of flows can be written as:

$$\eta = \frac{y}{\delta} = \frac{y}{\sqrt{\pm\frac{\nu x}{U_e}}} = \frac{y}{\sqrt{\pm\frac{\nu L}{U_0}(\frac{x}{L})^{1-m}}}. \tag{53}$$

When U_e and x have the same signs, the Falkner–Skan equation can be written as:

$$f''' + \frac{1}{2}(m+1) f f'' + m(1 - f'^2) = 0. \tag{54}$$

The two arbitrary constants α and β have been reduced to one constant m by fixing the scale for the function $\delta(x)$. The boundary conditions of the equation are:

$$f(0) = 0 \tag{55}$$

$$f'(0) = 0 \tag{56}$$

$$f'(\eta \to \infty) = 1. \tag{57}$$

If the Falkner–Skan Equation (54) is carefully examined, it can be seen that when the constant m is 0, the equation will be Blasius flow. Moreover, if the constant m is 1, the equation will be Hiemenz flow (see Section 2.2.1). This important point is stated before and it is emphasized one more time after the derivation. Another great usage of the Falkner–Skan equation is to simulate the boundary-layer over a wedge with half-angle $\theta = \frac{m\pi}{m+1}$ when the m is between 0 and 1. If the m is in between 1 and 2, the Falkner–Skan equation

will solve a corner flow with $\theta > \frac{\pi}{2}$. The visual schematic of four different physical flows that can be calculated from the Falkner–Skan equation can be seen in Figure 4. One interesting point of the equation is that when $m = -0.0904$, the obtained profile will have zero-shear at the wall which corresponds to the verge of the separation point for all x stations.

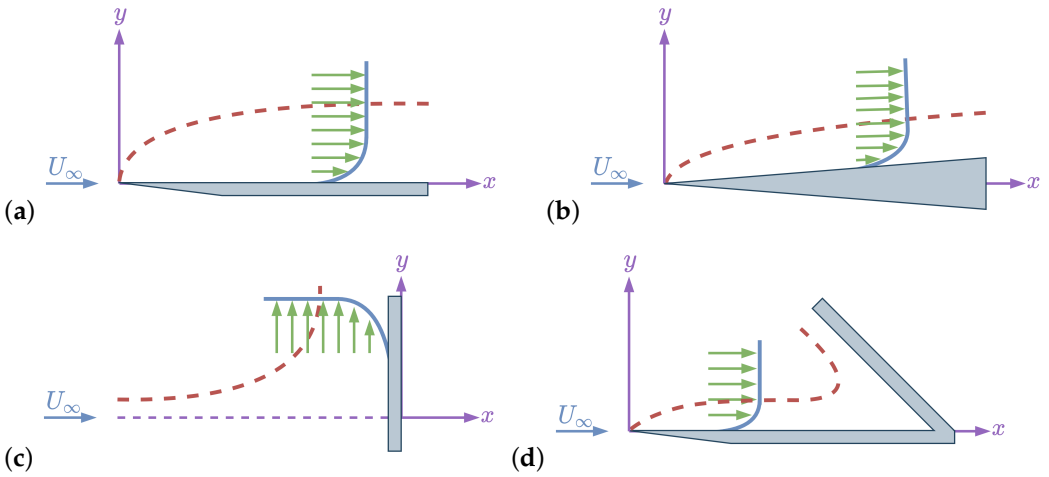

Figure 4. The representation of flow types that can be calculated with the Falkner–Skan equation. (**a**) Blasius flow ($m = 0$), (**b**) wedge flow ($0 < m < 1$), (**c**) Hiemenz flow ($m = 1$), and (**d**) corner flow ($1 < m < 2$).

2.1.4. Numerical Solution of Falkner–Skan Flow Problem

In this section, the Falkner–Skan equation will be solved with the Thomas algorithm and central finite difference scheme. In order to do that, the third-order ordinary differential equation should be reduced to second-order and first-order differential equations as it is done in Blasius flow. If it is assumed that $f' = h$ and $f = p$ in Equation (54), the system of equations will be:

$$h'' + \frac{1}{2}(m+1)ph' + m(1-h^2) = 0 \tag{58}$$

$$p' - h = 0, \tag{59}$$

where the boundary conditions of the system are:

$$p(0) = 0 \tag{60}$$

$$h(0) = 0 \tag{61}$$

$$h(\infty) = 1 \tag{62}$$

If finite difference scheme is applied to h variable, the final system of equations will be:

$$A_n h_{n+1} + B_n h_n + C_n h_{n-1} + D_n = 0, \tag{63}$$

where the A_n, B_n, C_n, and D_n are:

$$A_n = \frac{1}{\Delta\eta^2} + (m+1)\frac{p_n}{4\Delta\eta} \tag{64}$$

$$B_n = \frac{-2}{\Delta\eta^2} - mh_n \tag{65}$$

$$C_n = \frac{1}{\Delta\eta^2} - (m+1)\frac{p_n}{4\Delta\eta} \tag{66}$$

$$D_n = m. \tag{67}$$

The relation between h_n and h_{n-1} is taken as it is done in Blasius flow. When Equation (37) is substituted into Equation (63), the final G_n and H_n will be:

$$G_n = \frac{-C_n G_{n-1} + D_n}{B_n + C_n H_{n-1}} \tag{68}$$

$$H_n = \frac{-A_n}{B_n + C_n H_{n-1}}. \tag{69}$$

The implementation of these variables in the Julia environment can be seen in Listing 3.

Listing 3. Implementation of the Thomas algorithm for the Falkner–Skan profile.

```
m = 0.5
h = [(i-1)/(N-1)  for i=1:N]

for i = 2:N
p[i] = p[i-1] + (h[i] + h[i-1])*Δη/2
end

while 1e-8<=errorProfile

A = [ 1/Δη^2 + (m+1)*p[i]/(4*Δη) for i=1:N]
B = [-2/Δη^2 - m*h[i]             for i=1:N]
C = [ 1/Δη^2 - (m+1)*p[i]/(4*Δη) for i=1:N]
D = [ m for i=1:N]

for i=2:N-1
G[i] = - ( C[i]*G[i-1] + D[i] )/(B[i] + C[i] * H[i-1])
H[i] = -                  A[i]  /(B[i] + C[i] * H[i-1])
end

hp = copy(h)

for i=N-1:-1:2
h[i] = G[i] + H[i] * h[i+1]
end

errorProfile = maximum(abs.(hp-h))

for i = 2:N
p[i] = p[i-1] + (h[i] + h[i-1])*Δη/2
end
end
```

The final profiles for the varying m values can be seen in the Figure 5. The profiles are also compared with Schlichting [6] in order to validate the results. It has to be noted that, Schlichting used $\eta = \sqrt{\frac{m+1}{2}} \frac{y}{\sqrt{\frac{\nu x}{U_\infty}}}$ as similarity coordinate so the figure is plotted according to this nondimensionalization.

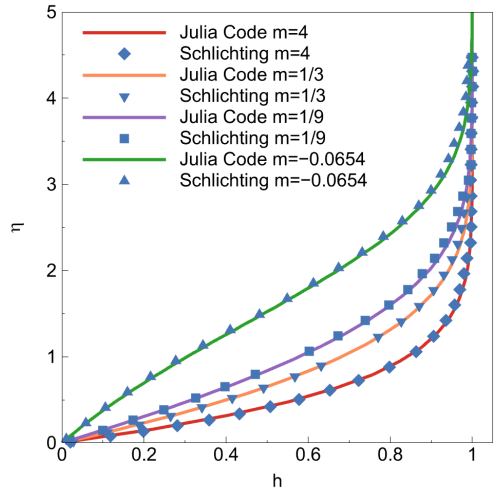

Figure 5. The velocity distribution of the Falkner–Skan similarity solution obtained by Julia code and data digitized from Schlichting [6] for varying m values.

2.2. Stagnation Point Flow Problems

Boundary-layer velocity distribution over a wall that is perpendicular to the flow velocity vector can be represented with a similarity solution. In this subsection, Hiemenz and Homann similarity solutions will be derived from scratch and they will be solved numerically in the Julia environment. The Julia codes will be available to shorten the learning curve.

2.2.1. Hiemenz Flow Problem

This problem is that of a fluid flow that is parallel to the y-axis in the far-field impinging on a wall that coincides with the x-axis. The flow which is perpendicular to a cylinder can be assumed as the Hiemenz flow around the stagnation point. The schematic description of the Hiemenz flow can be seen in Figure 6. In the Hiemenz flow, viscous forces away from the wall become so small in comparison with the inertia forces, particularly when the Reynolds number is large. In this case, inviscid irrotational flow assumption ($\xi = \nabla \times U = 0$) can be done. The velocity can be represented with a scalar function, ϕ, which is the velocity potential. The velocities in the x-direction and y-direction can be written as:

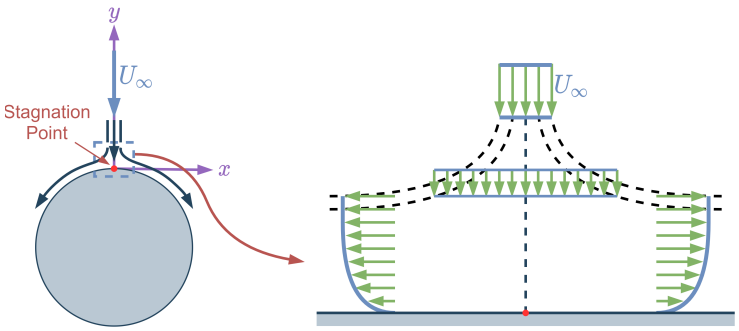

Figure 6. Schematic description of the Hiemenz and Homann flow. **Left** side corresponds to general flow and the **Right** side is the extended vision of the dashed rectangle. The flow represents Hiemenz flow if the circle is the projection of a cylinder and Homann flow if the circle is the projection of a sphere.

$$u = \frac{\partial \phi}{\partial x}, \quad v = \frac{\partial \phi}{\partial y}. \tag{70}$$

If the dimensional Navier–Stokes equations (Equations (1)–(3)) are considered for this flow as well, and the continuity equation becomes:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0. \tag{71}$$

On the wall, in the absence of viscosity, the flow can slip in the x-direction but in y-direction the velocity must be zero ($\frac{\partial \phi}{\partial y}|_{y=0} = 0$) because of the no-penetration boundary condition. The inviscid flow solution for the potential function was found to be:

$$\phi_{(x,y)} = \frac{a}{2}(x^2 - y^2), \tag{72}$$

where a is a constant that depends on the freestream flow and the body shape. This solution satisfies the governing equations for inviscid, irrotational flow, and the boundary conditions which can be shown as:

$$u = \frac{\partial \phi}{\partial x} = ax, \quad v = \frac{\partial \phi}{\partial y} = -ay. \tag{73}$$

Since potential the function is obtained, the stream-function can be calculated from the potential function. the velocity components, in terms of stream-function, $\psi_{(x,y)}$:

$$u = ax = \frac{\partial \psi}{\partial y}, \quad v = -ay = -\frac{\partial \psi}{\partial x}. \tag{74}$$

If Equation (74) is integrated for x and y, the stream function can be found as:

$$\psi = axy + c. \tag{75}$$

In order to find the pressure in the inviscid flow, Bernoulli equation [42] can be used. The pressure from the Bernoulli equation is:

$$p_0 - p = \frac{a^2 \rho}{2}(x^2 + y^2), \tag{76}$$

where p_0 is the stagnation pressure and ρ is the density. So far, flow is assumed as inviscid flow; however viscous forces will modify the inviscid solution, in particular in the y-direction. Hence, the viscous velocity components can be assumed as:

$$u(x, y) = xg(y) \tag{77}$$

$$v(x, y) = -f(y) \tag{78}$$

If Equations (77) and (78) are substituted in the Navier–Stokes equations (Equations (1)–(3)), the final system of equations will be:

$$g = f' \tag{79}$$

$$xg^2 - xfg' = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu x g'' \tag{80}$$

$$ff' = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu f''. \tag{81}$$

Once Equation (79) is substituted into Equations (80) and (81), two non-linear ordinary differential equations can be obtained as:

$$xf'^2 - xff'' = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu x f''' \tag{82}$$

$$ff' = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu f'' \tag{83}$$

It has to be noted that these two equations are coupled, which means they have to be solved together. If the x-momentum and y-momentum are solved for pressure, it can be seen that the pressure will be in the form of:

$$p(x,y) = \frac{1}{2}x^2(constant) + H(y), \tag{84}$$

where $H(y)$ is a function depends on only y. On the other hand, if the inviscid pressure compared with the viscous pressure as $y \to \infty$, the pressure gradient in x-direction and y-direction can be found as:

$$\frac{\partial p}{\partial x} = -a^2\rho x \tag{85}$$

$$\frac{\partial p}{\partial y} = -\frac{a^2\rho}{2}F'(y). \tag{86}$$

If the pressure gradients are substituted into Equations (82) and (83), the resultant system of ordinary differential equations can be found as:

$$f'^2 - ff'' = a^2 + \nu f''' \tag{87}$$

$$ff' = \frac{a^2}{2}F' - \nu f''. \tag{88}$$

Momentum equations are decoupled in this system of equations so once, $f(y)$ is calculated from Equation (87), $F(y)$ in the Equation (88) can be solved. The boundary conditions of the system are:

$$u(x,0) = 0 \to f'_{(y=0)} = 0 \tag{89}$$

$$v(x,0) = 0 \to f_{(y=0)} = 0 \tag{90}$$

$$u(x, y \to \infty) = ax \to f'_\infty = a \tag{91}$$

$$v(x, y \to \infty) = -ay \to f''_\infty = 0 \tag{92}$$

Additionally, $F(y \to \infty) = y^2$ can be obtained from y-momentum equation as $y \to \infty$. The final Equations (87) and (88) can be solved with the given boundary conditions; however, it is possible to get rid of the dependence on a. In order to do that, affine transformation [43] can be used. If it is assumed that $f(y) = A\varphi(\eta)$ and $y = B\eta$ where A and B are constant to be determined, the f derivative functions can be calculated as:

$$f' = \frac{A}{B}\varphi' \tag{93}$$

$$f'' = \frac{A}{B^2}\varphi'' \tag{94}$$

$$f''' = \frac{A}{B^3}\varphi''' \tag{95}$$

If Equations (93)–(95) substituted into the Equations (87) and (88) and assume that $\frac{\nu}{AB} = 1$ and $\frac{a^2B^2}{A^2} = 1$, the A and B coefficients can be calculated as:

$$A = \sqrt{a\nu} \tag{96}$$

$$B = \sqrt{\frac{\nu}{a}} \tag{97}$$

The given transformation allows to reduce two ordinary differential equations into one third-order ordinary differential equation as:

$$\varphi''' + \varphi\varphi'' - \varphi'^2 + 1 = 0 \tag{98}$$

where the boundary conditions are:

$$f'(0) = 0 \rightarrow \varphi'(0) = 0 \tag{99}$$

$$f(0) = 0 \rightarrow \varphi(0) = 0 \tag{100}$$

$$f'(\infty) = a \rightarrow \varphi'(\infty) = 1 \tag{101}$$

and the velocities:

$$u = ax\varphi'(\eta) \tag{102}$$

$$v = -\sqrt{a\nu}\varphi(\eta) \tag{103}$$

the resultant equation along with the boundary conditions can be solved with different numerical approaches such as Runge-Kutta, Runge-Kutta-Fehlberg, compact finite difference, high-order finite difference; however, in this present paper, the Thomas algorithm with finite difference discretization will be used to solve the Hiemenz profile in the Julia framework.

2.2.2. Numerical Solution of Hiemenz Flow Problem

The computational approach for the Hiemenz flow is similar to the Blasius solution since the final ordinary equation of the Hiemenz flow (Equation (98)) is similar to the Blasius similarity solution (Equation (20)). If it is assumed that $\varphi' = h$ and $\varphi = p$ in Equation (98), the system of equations will be:

$$h'' + ph' - h^2 + 1 = 0 \tag{104}$$

$$p' - h = 0, \tag{105}$$

where the boundary conditions of the system are:

$$p(0) = 0 \tag{106}$$

$$h(0) = 0 \tag{107}$$

$$h(\infty) = 1. \tag{108}$$

If finite difference scheme is applied to the h variable as in Blasius flow. The final system of equations will be:

$$A_n h_{n+1} + B_n h_n + C_n h_{n-1} + D_n = 0, \tag{109}$$

where the A_n, B_n, C_n, and D_n are:

$$A_n = \frac{1}{\Delta\eta^2} + \frac{p_n}{2\Delta\eta} \tag{110}$$

$$B_n = \frac{-2}{\Delta\eta^2} - h_n \tag{111}$$

$$C_n = \frac{1}{\Delta\eta^2} - \frac{p_n}{2\Delta\eta} \tag{112}$$

$$D_n = 1. \tag{113}$$

The relation between h_n and h_{n-1} is taken as it is done in Blasius flow. When Equation (37) is substituted into the Equation (109), the final G_n and H_n coefficients can be found as:

$$G_n = \frac{-C_n G_{n-1} + D_n}{B_n + C_n H_{n-1}} \tag{114}$$

$$H_n = \frac{-A_n}{B_n + C_n H_{n-1}}. \tag{115}$$

In order to start the calculation, H_1 can be assumed as 0 and the G_1 can be calculated as 0 from the boundary conditions. Once G_1 and H_1 are calculated, G_n and H_n can be calculated from Equations (114) and (115). p can be calculated as it is done for Blasius flow:

$$p_n = p_{n-1} + \int_{\eta_{n-1}}^{\eta_n} h d\eta. \tag{116}$$

The system of equations can be implemented in the Julia environment as it is shown in Listing 4. The linear profile assumption is taken as the initial condition of h (see Figure 2) as it is done for the Blasius solution. The reason hp and $errorProfile$ variables are used in the Listing 4 is that the linear profile converges to the Hiemenz profile in each iteration so in order to check the difference between previous and present profiles, the solution vector from the previous iteration is copied and it is compared with the new solution vector. If the difference between these two solution vectors is less than ϵ, which is an arbitrary limit and can be taken as 1×10^{-8}, then it can be said that the solution has converged. It has to be noted that ϵ can be taken as any number; however, if it is small, the solution will be more accurate. The final result of the Hiemenz flow can be seen in Figure 7. The results are also validated with White [44].

2.2.3. Homann Flow Problem

Homann flow is similar to Hiemenz flow. The only difference is that Homann flow is an axisymmetric version of the Hiemenz flow. The same schematic (Figure 6) can represent this flow as well; however, in this flow, the circle is the projection of a sphere. On the other hand, it was the projection of a cylinder in Hiemenz flow. Derivation of the Homann similarity solution has the almost same procedure as well but it uses cylindrical coordinates instead of Cartesian coordinates. The velocity components of the flow can be shown as:

$$v_r = v_r(r, z) \tag{117}$$

$$v_\theta = 0 \tag{118}$$

$$v_z = v_z(r, z) \tag{119}$$

$$p = p(r, z). \tag{120}$$

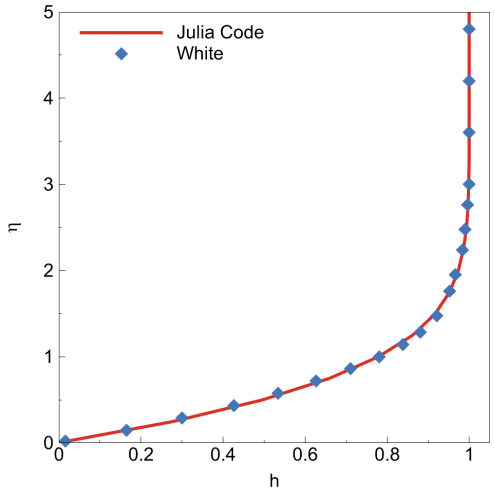

Figure 7. The velocity distribution of Hiemenz similarity solution obtained by Julia code and data digitized from White [44].

As axisymmetric assumptions, derivative with respect to θ ($\frac{\partial}{\partial\theta}$) and v_θ can be assumed as zero. The Navier–Stokes equations in cylindrical coordinates can be written as:

$$\frac{\partial v_r}{\partial r} + \frac{v_r}{r} + \frac{\partial v_z}{\partial z} = 0 \tag{121}$$

$$v_r\frac{\partial v_r}{\partial r} + v_z\frac{\partial v_z}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial r} + \nu\left(\frac{\partial^2 v_r}{\partial r^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} - \frac{v_r}{r^2} + \frac{\partial^2 v_r}{\partial z^2}\right) \tag{122}$$

$$v_r\frac{\partial v_z}{\partial r} + v_z\frac{\partial v_z}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left(\frac{\partial^2 v_z}{\partial r^2} + \frac{1}{r}\frac{\partial v_z}{\partial r} + \frac{\partial^2 v_z}{\partial z^2}\right). \tag{123}$$

If the same procedures applied to Navier–Stokes in Cartesian coordinates for Hiemenz flow, are applied in cylindrical coordinates, the final potential function and stream function will be:

$$\phi(r,z) = k(\frac{r^2}{2} - z^2) \tag{124}$$

$$\psi(r,z) = kzr^2, \tag{125}$$

where k is a constant which depends on the freestream flow and the body shape. The corresponding velocities are:

$$v_r = \frac{\partial\phi}{\partial r} = \frac{1}{r}\frac{\partial\psi}{\partial z} = kr \tag{126}$$

$$v_z = \frac{\partial\phi}{\partial z} = \frac{-1}{r}\frac{\partial\psi}{\partial r} = -2kz. \tag{127}$$

Once velocities are calculated, the pressure from the Bernoulli can be calculated as:

$$p_0 - p = \frac{1}{2}\rho k^2(r^2 + 4z^2). \tag{128}$$

The viscous velocity components and pressure can be calculated as:

$$v_r = rg(z) \tag{129}$$

$$v_z = -2f(z) \tag{130}$$

$$p_0 - p = \frac{1}{2}\rho k^2(r^2 + F(z)). \tag{131}$$

Listing 4. Implementation of Thomas algorithm for Hiemenz profile.

```
h = [(i−1)/(N−1)  for i=1:N]

for i = 2:N
p[i] = p[i−1] + (h[i] + h[i−1])*Δη/2
end

while 1e−8<=errorProfile

A = [ 1/Δη^2 + p[i]/(2*Δη) for i=1:N]
B = [−2/Δη^2 − h[i]         for i=1:N]
C = [ 1/Δη^2 − p[i]/(2*Δη) for i=1:N]
D = [ 1 for i=1:N]

for i=2:N−1
G[i] = − ( C[i]*G[i−1] + D[i] )/(B[i] + C[i] * H[i−1])
H[i] = −                  A[i]  /(B[i] + C[i] * H[i−1])
end

hp = copy(h)

for i=N−1:−1:2
h[i] = G[i] + H[i] * h[i+1]
end

errorProfile = maximum(abs.(hp−h))

for i = 2:N
p[i] = p[i−1] + (h[i] + h[i−1])*Δη/2
end
end
```

After the affine transformation, the final ordinary differential equation will be:

$$\varphi''' + 2\varphi\varphi'' - \varphi'^2 + 1 = 0, \tag{132}$$

where the boundary conditions of the equation is:

$$\varphi'(0) = 0 \tag{133}$$
$$\varphi(0) = 0 \tag{134}$$
$$\varphi'(\infty) = 1 \tag{135}$$

If one compares Hiemenz and Homann similarity solutions (see Equations (98) and (132)) the only difference is the 2 in the second term. In the same manner, the computational process will be the same except for two lines of code.

2.2.4. Numerical Solution of Homann Flow Problem

Since equations of Hiemenz and Homann similarity solutions are the same except one coefficient, procedures for the solution are also the same, except for two lines of code. If it is assumed that $\varphi' = h$ and $\varphi = p$ in Equation (132), the system of equations will be:

$$h'' + 2ph' - h^2 + 1 = 0 \tag{136}$$
$$p' - h = 0, \tag{137}$$

where the boundary conditions of the system are:

$$p(0) = 0 \tag{138}$$
$$h(0) = 0 \tag{139}$$
$$h(\infty) = 1 \tag{140}$$

If the finite difference scheme is applied to the h variable, the final system of equations will be:

$$A_n h_{n+1} + B_n h_n + C_n h_{n-1} + D_n = 0, \tag{141}$$

where the A_n, B_n, C_n, and D_n are:

$$A_n = \frac{1}{\Delta\eta^2} + \frac{p_n}{\Delta\eta} \tag{142}$$

$$B_n = \frac{-2}{\Delta\eta^2} - h_n \tag{143}$$

$$C_n = \frac{1}{\Delta\eta^2} - \frac{p_n}{\Delta\eta} \tag{144}$$

$$D_n = 1. \tag{145}$$

The final G_n and H_n coefficients are the same as Hiemenz flow as well and they are:

$$G_n = \frac{-C_n G_{n-1} + D_n}{B_n + C_n H_{n-1}} \tag{146}$$

$$H_n = \frac{-A_n}{B_n + C_n H_{n-1}}. \tag{147}$$

The final code can be seen in Listing 5. If the Hiemenz code (4) and Homann code (5) are compared, the only difference is in the A and C and it is because of the finite difference approach for the h'.

Listing 5. Implementation of Thomas algorithm for Homann profile.

```
h = [(i-1)/(N-1)  for i=1:N]

for i = 2:N
p[i] = p[i-1] + (h[i] + h[i-1])*Δη/2
end

while 1e-8<=errorProfile

A = [ 1/Δη^2 + p[i]/Δη for i=1:N]
B = [-2/Δη^2 - h[i]    for i=1:N]
C = [ 1/Δη^2 - p[i]/Δη for i=1:N]
D = [ 1 for i=1:N]

for i=2:N-1
G[i] = - ( C[i]*G[i-1] + D[i] )/(B[i] + C[i] * H[i-1])
H[i] = -                 A[i]  /(B[i] + C[i] * H[i-1])
end

hp = copy(h)

for i=N-1:-1:2
h[i] = G[i] + H[i] * h[i+1]
end

errorProfile = maximum(abs.(hp-h))

for i = 2:N
p[i] = p[i-1] + (h[i] + h[i-1])*Δη/2
end
end
```

The final solution profile can be seen in Figure 8. The results are also validated with White [44]. One can use these results to validate their own codes. Reference data will be shared on GitHub as well.

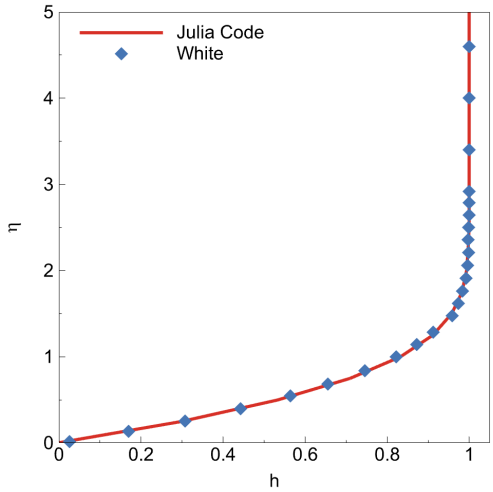

Figure 8. The velocity distribution of Homann similarity solution obtained by Julia code and digitized from White [44].

3. Discussion

Fast computational fluid dynamics solvers are crucial for engineers because the design process requires a lot of simulations to reach the final and optimized design. One of the most important factors that make a solver fast is the language itself. Writing almost the same script in different languages may give different solution central processing unit (CPU) times. For instance, a script that uses long and complex for loops in both Julia and Python environments will result in different solution times because Python is slower with for loops in general. The reasons that cause this slow behavior will not be covered in this paper since it is out of the scope of this paper. However, it is important to state these differences in order to decide which coding language is proper for the simulation that is planned.

Sometimes, some other criteria, such as a user-friendly environment, might be the critical condition. The easy matrix–matrix multiplication and backslash linear system solver can be some of the user-friendly examples. In CFD, the usage of vectors and matrices is so common. Most of the time, it is required to multiply or add vectors or matrices with one another. Fortran, which is one of the fastest languages and also one of the fundamental languages in the CFD industry, does not have a built-in element-wise vector or matrix multiplication feature. This requires the use of for loops for each element-wise matrix and vector operation. As a result of this, every time, one needs to use a for loop to do these operations. This ends up with hard-to-read codes and also excessive usage of indices is another source of possible mistakes that will cause trouble during the debugging. However, in the Julia environment, it can be done in one line without any trouble. Listing 6 is showing the two for loop usage. Both of them use Julia syntax in order to prevent confusion; however, it is required to state that Fortran has a different syntax than this but, logically, the for loops are the same with Fortran logic.

Another user-friendly feature of Julia over Fortran is to plot a vector in the code with built-in functions. However, in Fortran, it is not possible to do it with built-in functions so one needs to extract the solution vector or matrix to an external file to visualize it. As it is stated before, coding language selection can be an important topic. In this section, the strong sides of the Julia environment will be stated. One of the user-friendly features of Julia is compact for loop syntax. Listing 7 shows the traditional way and compact way to use a for loop in the Julia environment.

Listing 6. Comparison of the element-wise and compact matrix-matrix multiplication.

```
for l=1:m
for k=1:nz
for j=1:ny
for i=1:nx
A[i,j,k,l] = B[i,j,k,l]*C[i,j,k,l]
end
end
end
end

A .= B.*C
```

Listing 7. Comparison of the traditional and compact way of for loop usage.

```
for i=1:N
h[i] = (i-1)/(N-1)
end

h = [(i-1)/(N-1)  for i=1:N]
```

The readability of the code is also extremely important in order to explain the code to others or provide it as open-source code. Julia is so strong in this topic because it allows the usage of LaTeX language within the code. It is one of the unique abilities of this language. If you are writing a formula that is full of Greek letters and regular letters, it is inevitable to use complex variable names. Comparison of the usage of Greek letters and regular formula writing can be seen in Listing 8.

Listing 8. Comparison of the Greek letter usage and conventional formula writing in Julia.

```
alpha = a*beta-c/(gamma-1)*(d_i-1/(eta^2))

α = a*β-c/(γ-1)*(d_i-1/(η^2))
```

The last advantage of Julia stated in this present paper is the ability to do both dynamic and static RAM allocations. It may be important to manually allocate the variable sizes and types for optimized code. However, sometimes it is easier to use just variables without any initializing. Initializing differences in the Julia environment can be seen in Listing 9.

Listing 9. Different initializing methods in Julia environment.

```
nx = 10
ny = 15.2

q = Array{Float64}(undef, nx)
for i=1:nx
q[i] = i
end

a = Array{Float64}(undef, nx)
c = zeros(nx)
a = q
b = q
c = q
```

As it is seen from the example scripts, Julia is a user-friendly, fast, open-source, and free language which can increase productivity drastically [36]. It is a great choice both for those who are new to coding and coding experts. Julia also can call the C, Fortran, and Python libraries so it is great for experienced engineers who think their previous code in other coding languages will be useless.

4. Conclusions

In the computational fluid dynamics industry, it is crucial to have some predictions about the flow that will be simulated. It helps to spot the location at which finer mesh is required. Fundamental knowledge about canonical flows is crucial in this point because most

of the complex flow consists of a combination of a couple of canonical flows. For example, an airfoil CFD simulation can consist of boundary-layer flow over the smooth part of it, mixing layer flow where the tail ends, and blunt body flow in the wake region. In other words, one airfoil simulation consists of three different canonical flows. A full understanding on canonical flows is extremely important in order to simulate similar flows accurately and cheaply. In this paper, boundary-layer theory is introduced and boundary-layer flows are derived from scratch. The Blasius flow, Hiemenz flow, Homann flow, Falkner–Skan flow are the focus of this paper. Once the derivations of them are completed, derived forms are implemented in the Julia environment. In order to model the equations, a finite difference scheme for space discretization is used and the Thomas algorithm is used for the linear system solution. It has to be noted that other methods such as Runge-Kutta, Runge-Kutta-Fehlberg, compact finite difference, high-order finite difference can also be used to solve the given ODEs; however, in this present paper, authors preferred the finite difference method and Thomas algorithm. The authors further discussed the advantages of the Julia language over some other coding languages available in the literature. It is shown that Julia syntax is so straightforward, easy, and user-friendly. Strong sides of the Julia environment are stated with the given comparisons as well. The popularity of Julia may drastically increase in the near future because of its potential.

Author Contributions: code generation, F.O. and K.K.; validation, F.O. and K.K.; writing—original draft preparation, F.O.; writing—review and editing, F.O. and K.K.; visualization, F.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All the data used and generated in this study is available in the GitHub link provided in Appendix A.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Julia setup files can be downloaded from their website (https://julialang.org/downloads/ (accessed on 1 June 2021)). The website also includes instructions on how to install Julia on Windows, Linux, and mac operating systems. Some of the useful resources for learning Julia are listed below:

- https://docs.julialang.org/en/v1/ (accessed on 1 June 2021)
- https://www.coursera.org/learn/julia-programming (accessed on 1 June 2021)
- https://www.youtube.com/user/JuliaLanguage/featured (accessed on 1 June 2021)
- https://www.youtube.com/user/Parallel Computing and Scientific Machine Learning (accessed on 1 June 2021)
- https://discourse.julialang.org/ (accessed on 1 June 2021)

It is common to use external packages for Julia. In order to do that, Pkg, which is Julia's built-in package manager, can be used. Once Julia is opened, Pkg can be activated with the "]" button in Windows. In Linux, calling "julia" in the terminal will open it. After that "Pkg.add("Pluto")" will trigger the setup process for that package. In here, we used Pluto as an example because, in GitHub, our codes are developed in the Pluto environment. After Pluto is installed. Pluto can be run with "Pluto.run()". This command will open a new tab in the browser which you can run your Julia codes. After that, the "using Pluto" line must be placed to the top of the file. For "Plots" package, the commands will be "Pkg.add("Plots")" and "using Plots". Since the Plots package does not have a GUI, there is not a command called "Plots.run()".

Other than Pluto, JuliaPro which includes Julia and the Juno IDE (https://juliacomputing.com/products/juliapro/ (accessed on 1 June 2021)) can be used as an editor and compiler.

This software contains a set of packages for plotting, optimization, machine learning, database, and much more. Pluto is appropriate for small scripts while JuliaPro is better for more complex codes. The GitHub link of the codes used in this paper is:

- https://github.com/frkanz/A-CFD-Tutorial-in-Julia (accessed on 1 June 2021)

References

1. Prandtl, L. *Über Flüssigkeitsbewegung bei sehr Kleiner Reibung, Verh 3 int*; English Translation; Math-Kongr: Heidelberg, Germanty, 1904. Available online: http://homepage.ntu.edu.tw/~wttsai/Adv_Fluid/NACA_TM-452.pdf (accessed on 1 June 2021).
2. Blasius, H. Grenzschichten in Flüssigkeiten mit kleiner Reibung. *Z. Math. Phys.* **1908**, *60*, 397–398.
3. Hager, W.H. Blasius: A life in research and education. *Exp. Fluids* **2003**, *34*, 566–571. [CrossRef]
4. Cousteix, T.; Cebeci, J. *Modeling and Computation of Boundary-Layer Flows*; Springer: Berlin/Heidelberg, Germany, 2005.
5. Hiemenz, K. Die Grenzschicht an einem in den gleichformigen Flussigkeitsstrom eingetauchten geraden Kreiszylinder. *Dinglers Polytech. J.* **1911**, *326*, 321–324.
6. Schlichting, H.; Gersten, K. *Boundary-Layer Theory*; Springer: Berlin/Heidelberg, Germany, 2016.
7. Howarth, L. *On the Calculation of Steady Flow in the Boundary Layer Near the Surface of a Cylinder in a Stream*; Technical Report; Aeronautical Research Council London: London, UK, 1934.
8. Homann, F. Der Einfluss grosser Zähigkeit bei der Strömung um den Zylinder und um die Kugel. *ZAMM-J. Appl. Math. Mech. Für Angew. Math. Und Mech.* **1936**, *16*, 153–164. [CrossRef]
9. Brennan, G.; Gajjar, J.; Hewitt, R. Tollmien–Schlichting wave cancellation via localised heating elements in boundary layers. *J. Fluid Mech.* **2021**, *909*, A16-1. [CrossRef]
10. Brennan, G.S.; Gajjar, J.S.; Hewitt, R.E. Cancellation of Tollmien–Schlichting waves with surface heating. *J. Eng. Math.* **2021**, *128*, 1–23. [CrossRef]
11. Corelli Grappadelli, M.; Sattler, S.; Scholz, P.; Radespiel, R.; Badrya, C. Experimental investigations of boundary layer transition on a flat plate with suction. In Proceedings of the AIAA Scitech 2021 Forum, Virtual Event, 11–15 and 19–21 January 2021; p. 1452.
12. Rigas, G.; Sipp, D.; Colonius, T. Nonlinear input/output analysis: Application to boundary layer transition. *J. Fluid Mech.* **2021**, *911*, A15. [CrossRef]
13. Haley, C.; Zhong, X. Supersonic mode in a low-enthalpy hypersonic flow over a cone and wave packet interference. *Phys. Fluids* **2021**, *33*, 054104. [CrossRef]
14. Malik, M.R. Numerical methods for hypersonic boundary layer stability. *J. Comput. Phys.* **1990**, *86*, 376–413. [CrossRef]
15. Fedorov, A. Transition and stability of high-speed boundary layers. *Annu. Rev. Fluid Mech.* **2011**, *43*, 79–95. [CrossRef]
16. Long, T.; Dong, Y.; Zhao, R.; Wen, C. Mechanism of stabilization of porous coatings on unstable supersonic mode in hypersonic boundary layers. *Phys. Fluids* **2021**, *33*, 054105. [CrossRef]
17. Fong, K.D.; Wang, X.; Zhong, X. Numerical simulation of roughness effect on the stability of a hypersonic boundary layer. *Comput. Fluids* **2014**, *96*, 350–367. [CrossRef]
18. Kara, K.; Balakumar, P.; Kandil, O. Receptivity of hypersonic boundary layers due to acoustic disturbances over blunt cone. In Proceedings of the 45th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, 8–11 January 2007; p. 945.
19. Kara, K.; Balakumar, P.; Kandil, O. Effects of wall cooling on hypersonic boundary layer receptivity over a cone. In Proceedings of the 38th Fluid Dynamics Conference and Exhibit, Seattle, WA, USA, 23–26 June 2008; p. 3734.
20. Kara, K.; Balakumar, P.; Kandil, O.A. Effects of nose bluntness on hypersonic boundary-layer receptivity and stability over cones. *AIAA J.* **2011**, *49*, 2593–2606. [CrossRef]
21. Oz, F.; Kara, K. Effects of Local Cooling on Hypersonic Boundary-Layer Stability. In Proceedings of the AIAA Scitech 2021 Forum, Virtual Event, 11–15 and 19–21 January 2021; p. 0940.
22. Drozdz, A.; Niegodajew, P.; Romanczyk, M.; Sokolenko, V.; Elsner, W. Effective use of the streamwise waviness in the control of turbulent separation. *Exp. Therm. Fluid Sci.* **2021**, *121*, 110291. [CrossRef]
23. Iyer, P.S.; Malik, M.R. Wall-modeled LES of flow over a Gaussian bump. In Proceedings of the AIAA Scitech 2021 Forum, Virtual Event, 11–15 and 19–21 January 2021; p. 1438.
24. Mohammed-Taifour, A.; Weiss, J. Periodic forcing of a large turbulent separation bubble. *J. Fluid Mech.* **2021**, *915*, A24. [CrossRef]
25. Hady, F.; Ibrahim, F.; Abdel-Gaied, S.; Eid, M. Effect of heat generation/absorption on natural convective boundary-layer flow from a vertical cone embedded in a porous medium filled with a non-Newtonian nanofluid. *Int. Commun. Heat Mass Transf.* **2011**, *38*, 1414–1420. [CrossRef]
26. Hady, F.M.; Ibrahim, F.S.; Abdel-Gaied, S.M.; Eid, M.R. Radiation effect on viscous flow of a nanofluid and heat transfer over a nonlinearly stretching sheet. *Nanoscale Res. Lett.* **2012**, *7*, 1–13. [CrossRef]
27. Hady, F.; Ibrahim, F.; Abdel-Gaied, S.; Eid, M.R. Boundary-layer non-Newtonian flow over vertical plate in porous medium saturated with nanofluid. *Appl. Math. Mech.* **2011**, *32*, 1577–1586. [CrossRef]
28. Hady, F.; Ibrahim, F.; Abdel-Gaied, S.; Eid, M. Boundary-layer flow in a porous medium of a nanofluid past a vertical cone. In *An Overview of Heat Transfer Phenomena*; Kazi, S.N., Ed.; IntechOpen: London, UK, 2012; pp. 91–104.
29. Sohail, M.; Naz, R.; Abdelsalam, S.I. Application of non-Fourier double diffusions theories to the boundary-layer flow of a yield stress exhibiting fluid model. *Phys. A Stat. Mech. Appl.* **2020**, *537*, 122753. [CrossRef]

30. Bhatti, M.; Alamri, S.Z.; Ellahi, R.; Abdelsalam, S.I. Intra-uterine particle–fluid motion through a compliant asymmetric tapered channel with heat transfer. *J. Therm. Anal. Calorim.* **2020**, *144*, 2259–2267. [CrossRef]
31. Barba, L.; Forsyth, G. CFD Python: The 12 steps to Navier-Stokes equations. *J. Open Source Educ.* **2018**, *2*, 21. [CrossRef]
32. Oliphant, T.E. *A Guide to NumPy*; Trelgol Publishing: Austin, TX, USA, 2006; Volume 1.
33. Ketcheson, D.I. Teaching numerical methods with IPython notebooks and inquiry-based learning. In Proceedings of the 13th Python in Science Conference, Austin, TX, USA, 6–12 July 2014; pp. 19–24.
34. Ketcheson, D.I.; Mandli, K.; Ahmadia, A.J.; Alghamdi, A.; de Luna, M.Q.; Parsani, M.; Knepley, M.G.; Emmett, M. PyClaw: Accessible, extensible, scalable tools for wave propagation problems. *SIAM J. Sci. Comput.* **2012**, *34*, 210–231. [CrossRef]
35. Pawar, S.; San, O. CFD Julia: A learning module structuring an introductory course on computational fluid dynamics. *Fluids* **2019**, *4*, 159. [CrossRef]
36. Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V.B. Julia: A fresh approach to numerical computing. *SIAM Rev.* **2017**, *59*, 65–98. [CrossRef]
37. Hosseini, M.; Vinuesa, R.; Hanifi, A.; Henningson, D.; Schlatter, P. Turbulent flow around a wing profile, a direct numerical simulation. In Proceedings of the 68th Annual Meeting of the APS Division of Fluid Dynamics, Boston, MA, USA, 22–24 November 2015.
38. Tannehill, J.C.; Pletcher, R.H.; Anderson, D.A. *Computational Fluid Mechanics and Heat Transfer*; Taylor & Francis: Bristol, PA, USA, 1997.
39. Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P. *Numerical Recipes in Fortran 90*; Cambridge University Press: New York, NY, USA, 1996.
40. Moin, P. *Fundamentals of Engineering Numerical Analysis*; Cambridge University Press: New York, NY, USA, 2010.
41. Atkinson, K.E. *An Introduction to Numerical Analysis*; John Wiley & Sons: New York, USA, 1989.
42. Munson, B.R.; Young, D.F.; Okiishi, T.H. Fundamentals of fluid mechanics. *Oceanogr. Lit. Rev.* **1995**, *10*, 831.
43. Rogers, D.F.; Adams, J.A. *Mathematical Elements for Computer Graphics*; McGraw-Hill: New York, NY, USA 1989.
44. White, F.M.; Corfield, I. *Viscous Fluid Flow*; McGraw-Hill: New York, NY, USA, 2006; Volume 3.