Group 3 PHYS 6124 Project 2 Report: Fluid Behavior in a Laminar Boundary Layer

Hamza, Omar, and Sidney
(Dated: November 8, 2021)

A report on Group 3's work on modeling air flowing over a flat plate. The boundary layer equations are first derived using the boundary layer assumptions. These equations are then discretized with variable grid steps, and placed into a Thomas Algorithm solver. In this method, at each x location, the finite difference matrix is iterated until the difference between each iteration satisfies some tolerance, this is referred to as a "fixed point condition".The generated data from this process is the steady state for velocity in the x and y direction. These results were compared to the skin friction coefficient as calculated from the Blasius solution of the boundary layer. The agreement between the two methods is serviceable considering the usage of Python instead of the more standard FLUENT solver, or FORTRAN.

I. INTRODUCTION

Computational fluid dynamics (CFD) is at the forefront of a majority of engineering projects. It is useful in everything from building airplanes and cars, to modelling the atmosphere and making weather predictions [1]. Unfortunately, due to the extreme complexity of the Navier Stokes equations, numerical studies of fluids can be extremely taxing both mentally, and computationally [2]. In fact, a large portion of engineers rely on "black box" style applications such as Ansys Fluent to perform CFD over a thin plate, without necessarily understanding the machinery behind it [3]. For this project, the goal is to gain a level of understanding for how the flow profiles we are all familiar with are actually calculated.

In order to solve the Navier-Stokes for our problem, we started by looking at a slightly simpler systems: Prandtl's Boundary Layer Equations[4]. This system of equations uses scale analysis to reduce the complexity of the Navier Stokes, and reduces the number of momentum equations to the continuity, and the velocity in the streamwise (x) direction. The increased tractability of the Boundary Layer Equations has itself blossomed into its own theory, which has been well recorded in Schlichting's classic textbook [5], where turbulence has been explored and variable temperatures added. The numerical analysis of Prandtl's equations has also been very well developed and has been verified using experimental methods[6].
In this exploration, a fixed point iteration method was utilized along with an implicit Crank-Nicolson finite difference scheme to numerically solve the coupled nonlinear partial differential equations describing the temperature, and velocities in the x and y directions[7]. This method was developed in 1963 by Paskonov in the USSR, unfortunately, due to the time period and location, it is rather difficult to obtain a readable copy of his work, and the nearest work easily obtainable utilizes an explicit scheme, or an implicit scheme with "lagged" coefficients. It is also relatively quick to implement, and matches well with the well-tested Blasius solution [5].

The beauty of the Boundary Layer Equations is that, although they are much simplified from the Navier Stokes, they do not represent an over-simplified ideal system. CFD performed on this system is useful in every engineering field from Mechanical to Nuclear. Therefore, the work in the following report is an effective tool to understand how fluids can be modelled, and more importantly: worked with and understood.

II. MATHEMATICAL BACKGROUND

The governing equation for flow over a thin flat plate is the well-known Navier Stokes equation [2]. We will start this exploration by deriving the boundary layer equations from the Navier Stokes equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{U}) = 0$$
$$\rho \frac{\partial \vec{U}}{\partial t} + \rho \left(\vec{U} \cdot \nabla \right) \vec{U} = -\nabla p + \nabla \left(\mu \left(\nabla \vec{U} + (\nabla \vec{U})^T \right) \right) \tag{1}$$

A. Assumptions

The assumptions are as follows:
The $(\zeta - \frac{2}{3}\mu)\nabla \cdot \vec{U}$ contribution to the stress tensor has been ignored due to the fact that we are going to be simulating flows at less than the mach number. Also, an air boundary layer was considered around the thin plate and the gravitational effects have been neglected. In spirit, the derivation of the Boundary Layer Equations is a "large Reynold's number" estimate, but it can also be expressed by assuming that there is negligible variance in the z-direction, and thus neglecting this term entirely from the Navier Stokes equations. We also assume that $U >> V$ (the velocity in the x direction is much larger than the velocity in the y direction) and $L >> \delta$ (the x length scale is much larger than the y length scale, i.e. the assumption of "thin" boundary layer). [8], [5]

B. Derivation

In flow over a plane, the above assumptions turn eq.(1) into

$$\begin{aligned} \frac{\partial(\rho U)}{\partial x} + \frac{\partial(\rho V)}{\partial y} &= 0 \\ \rho U \frac{\partial U}{\partial x} + \rho V \frac{\partial U}{\partial y} &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial y}\left(\mu \frac{\partial U}{\partial y}\right) \\ \frac{\partial p}{\partial y} &= 0 \end{aligned} \tag{2}$$

In order to understand fluid simulations, one needs to verify their accuracy. The easiest way to do this, is to compare with the Blasius solution, which has be well-verified experimentally [5]. However, the classical Blasius solution is only valid for the most simple of flows. Specifically, it neglects both acceleration, and variable fluid properties. With that in mind, eq.(3) becomes

$$\begin{aligned} \frac{\partial(U)}{\partial x} + \frac{\partial(V)}{\partial y} &= 0 \\ U \frac{\partial U}{\partial x} + V \frac{\partial U}{\partial y} &= \nu \frac{\partial^2 U}{\partial y^2} \\ \frac{\partial p}{\partial y} &= 0 \end{aligned} \tag{3}$$

With the governing equations derived, we can move on to their analysis via the method of similarity solution, and explore the classical Blasius solution [9].

C. Blasius Boundary Layer Solution

The case of the boundary-layer for a flow along a thin plate can also be treated using the Blasius equation [9]. For our problem, we will start the plate at $x = 0$, extend parallel to the $x - axis$ and is considered to be of semi-infinite length. We will treat the steady flow parallel to the x axis with free stream velocity U_∞.
Our main assumption is that in this case the velocity of the potential flow is constant. This is a reasonable assumption for a flow over a flat plate with zero pressure gradient. The derivation of Blasius equation from the partial differential equation can be derived as follows using the reference [5].

$$\partial u/\partial x + \partial v/\partial y = 0 \tag{4}$$

From conservation of mass, we get the following conditions on a *stream function*.

$$\begin{aligned} u &= \partial\psi/\partial y \\ v &= -\partial\psi/\partial x \end{aligned} \tag{5}$$

Where ψ is the stream friction. We can introduce a similarity parameter (e.g. a scale that effects the velocity profile in such a way that it looks the same at each x position). We can do this because the velocity properties are very similar along the thin plate.

$$\begin{aligned} l &= y\sqrt{v_\infty/v_x} \\ f(l) &= \frac{\psi}{v_\infty \sqrt{\frac{v_x}{v_\infty}}} \end{aligned} \tag{6}$$

Thus, the velocity along x and y are given by,

$$u = \frac{\partial\psi}{\partial y} = \frac{\partial\psi}{\partial l}\frac{\partial l}{\partial y} = v_\infty f'(l) \tag{7}$$

$$\begin{aligned} v = -\frac{\partial\psi}{\partial y} &= -\left[v_\infty \sqrt{\frac{v_x}{U_\infty}}\frac{\partial f}{\partial x} + \frac{v_\infty}{2}\sqrt{\frac{\nu}{v_\infty x}} f\right] \\ &= \frac{1}{2}\sqrt{\frac{\nu v_\infty}{x}}\left(2\frac{\partial f}{\partial l} - f\right) \end{aligned} \tag{8}$$

And the first derivatives, and the second y of u is given by

$$\frac{\partial u}{\partial x} = U_\infty f''\frac{\partial l}{\partial x} = -\frac{1}{2x}U_\infty l f'' \tag{9}$$

$$\begin{aligned} \frac{\partial u}{\partial y} &= U_\infty f''\frac{\partial l}{\partial y} = U_\infty\sqrt{\frac{v_\infty}{\nu x}} f'' \\ \frac{\partial^2 u}{\partial y^2} &= \frac{U_\infty^2}{\nu x} f''' \end{aligned} \tag{10}$$

Plugging those into the momentum equation.

$$2f''' + f''f = 0 \tag{11}$$

$$\begin{aligned} f &= \frac{\psi}{U_\infty}\sqrt{\frac{\nu x}{U_\infty}} \\ f' &= \frac{u}{U_\infty} \\ f'' &= \frac{1}{U_\infty}\sqrt{\frac{\nu x}{U_\infty}}\frac{\partial u}{\partial y} \end{aligned} \tag{12}$$

Taking physical constraints into account, our boundary conditions are:

$$\begin{aligned} f(0) &= 0 \ (v(x,0) = 0) \\ f'(0) &= 0 \ (u(x,0) = 0) \\ f'(\infty) &= 1 \ (u(\infty) = U_\infty) \end{aligned} \tag{13}$$

Then,

$$\tau_\omega = \mu\left(\frac{\partial u}{\partial y}\right)_{y=0} = \mu U_\infty\sqrt{\frac{U_\infty}{\nu x}}\ f''(0) \tag{14}$$

Note: $f''(0)$ needs to be solved for numerically. By definition

$$\begin{aligned} \frac{c_f}{2} = \frac{\tau_\omega}{\rho U_\infty^2} &= \frac{\mu}{\rho}\frac{U_\infty\sqrt{U_\infty/\nu x} f''(0)}{U_\infty^2} \\ &= \frac{\nu f''(0)}{\sqrt{\nu x U_\infty}} = \frac{f''(0)}{\sqrt{Rex}} \end{aligned} \tag{15}$$

We can solve for $f''(0)$ via the shooting method. The process starts by guessing $f''(0)$, and solving the ODE normally, all the while checking the asymptote of $u(x,y)$, the guess of $f''(0)$ is refined until freestream velocity is reached. This has been done, and the answer is ≈ 0.332. With a pseudo-analytical equation to check against, we can go onto solving eq.(3) through finite differences.

III. NUMERICAL METHODS

In short, we are discretizing eqns. (3) and constructing a tri-diagonal matrix out of the resulting algebraic equations which can then be solved via the Thomas Algorithm. To better match with eq.(15) non-uniform grid spacing ($\Delta y_- \neq \Delta y_+$) is utilized, this changes the form of the finite discretization. The magnitude of difference between the two steps is determined by the equation

$$dy_1 = \frac{\delta}{\sum_{i=0}^{k-2} M^i}$$

Where M determines the spacing, and δ is the boundary layer thickness $\frac{5pL}{\sqrt{\frac{U_0 L}{\nu}}}$ where U_0 is the free stream velocity prior to acceleration, L is the x length of the system, p is a safety factor empirically matched to experiment (3 in our case), $\nu = \frac{\mu}{\rho}$, and k is the number of y nodes. From here, we can determine the i^th y location via

$$y_i = \sum_{j=2}^{i} M^{j-2} * dy_1$$

We can then determine the step sizes by directly taking differences within the loop construction the tridiagonal matrix. A similar formulation is used for the grid step in the x-direction, but the step size dx is calculated at the beginning of each step in x. This method is implicit, and thus we only work with $dx_- = x[i] - x[i-1]$. We then discretize the streamwise velocity U into the following 3-point recurrence relation:

$$a_{i,j}\theta_{i,j-1} + b_{i,j}\theta_{i,j} + c_{i,j}\theta_{i,j+1} = d_{i,j} \tag{16}$$

This gives us a $k \times k$ tri-diagonal matrix at every x node, which can be solved via the Thomas Algorithm. The following is the step-by-step method of constructing eq.(16) for the momentum equation.

$$U(x,y) = U(x_0,y_0) + (x - x_0)\frac{\partial U}{\partial x} \tag{17}$$

$$\begin{aligned} U(x_0,y) &= U(x_0,y_0) \\ U(x_0 - \Delta x, y) &= U(x_0,y_0) - \Delta x \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial x} &= \frac{U(x_0,y) - U(x_0 - \Delta x, y)}{\Delta x} \end{aligned} \tag{18}$$

To keep the approximation second order accurate I shall use a polynomial fit for the first derivative with respect to y. This fit was found using the derivative of a Lagrange polynomial expanded to three terms:

$$\begin{aligned} &\text{Using the interpolation nodes} \\ &(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2)) \\ f'(x_j) \approx\ & f(x_0)\left[\frac{2x_j - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right] + \\ & f(x_1)\left[\frac{2x_j - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right] + \\ & f(x_2)\left[\frac{2x_j - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right] \end{aligned} \tag{19}$$

If we define $\Delta y_- = (x_1 - x_0)$ and $\Delta y_+ = (x_2 - x_1)$, we produce the following from eq.(19)

$$\begin{aligned} \frac{\partial U}{\partial y} &= \frac{-\Delta y_+ U(x, y_0 - \Delta y_-)}{\Delta y_-(\Delta y_- + \Delta y_+)} \\ &+ \frac{\Delta y_+ - \Delta y_-}{\Delta y_- \Delta y_+} U(x,y_0) + \frac{\Delta y_- U(x, y_0 + \Delta y_+)}{\Delta y_+(\Delta y_- + \Delta y_+)} \end{aligned} \tag{20}$$

Also, assuming that ν is a function of x and y only (thin plate).

$$\frac{\partial}{\partial y}\left(\nu \frac{\partial U}{\partial y}\right) \rightarrow \frac{\partial \phi}{\partial y} \tag{21}$$

Now expanding the equation around $U(x,y_0)$

$$\begin{aligned} \phi\left(x, y_0 + \frac{\Delta y_+}{2}\right) &= \phi(x,y_0) + \frac{\Delta y_+}{2}\frac{\partial \phi}{\partial y} \\ \phi\left(x, y_0 - \frac{\Delta y_-}{2}\right) &= \phi(x,y_0) - \frac{\Delta y_-}{2}\frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial y} &= \frac{2\phi\left(x, y_0 + \frac{\Delta y_+}{2}\right) - 2\phi\left(x, y_0 - \frac{\Delta y_-}{2}\right)}{\Delta y_- + \Delta y_+} \end{aligned} \tag{22}$$

$$\begin{aligned} \left.\frac{\partial U}{\partial y}\right|_{i-1/2} &= \frac{U(x,y_0) - U(x, y_0 - \Delta y_-)}{\Delta y_-} \\ \left.\frac{\partial U}{\partial y}\right|_{i+1/2} &= \frac{U(x, y_0 + \Delta y_+) - U(x,y_0)}{\Delta y_+} \end{aligned} \tag{23}$$

Plugging (23) into (22) and performing some minor algebra we obtain:

$$\begin{aligned} \frac{\partial}{\partial y}\left(\nu \frac{\partial U}{\partial y}\right) &= \frac{2}{\Delta y_- + \Delta y_+} \\ &\left(\frac{\nu\left(x, y_0 + \frac{\Delta y_+}{2}\right)(U(x, y_0 + \Delta y_+) - U(x,y_0))}{\Delta y_+}\right) \\ &- \frac{2}{\Delta y_- + \Delta y_+} \\ &\left(\frac{\nu\left(x, y_0 - \frac{\Delta y_-}{2}\right)(U(x,y_0) - U(x, y_0 - \Delta y_-))}{\Delta y_-}\right) \end{aligned} \tag{24}$$

We can then take into account that ν is constant, and plug all of this into eq.(3). By writing the above in index notation, and rearranging we can construct the coefficients for eq.(16).

$$a = -\frac{V_{i,j}\Delta y_+}{\Delta y_+(\Delta y_+ + \Delta y_-)} - \frac{2\nu}{\Delta y_-(\Delta y_+ + \Delta y_-)} \tag{25}$$

$$b = \frac{U_{i,j}}{\Delta x} + \frac{V_{i,j}(\Delta y_+ - \Delta y_-)}{\Delta y_+ \Delta y_-} + \frac{2\nu}{\Delta y_+ \Delta y_-} \tag{26}$$

$$c = V_{i,j}\frac{\Delta y_-}{\Delta y_-(\Delta y_+ + \Delta y_-)} - \nu\frac{2}{\Delta y_+(\Delta y_+ + \Delta y_-)} \tag{27}$$

$$d = U_{i,j}\frac{U_{i-1,j}}{\Delta y_+} \tag{28}$$

For all j, these coefficients form the matrix equation $A\vec{U} = \vec{d}$. Unfortunately, this is not an explicit scheme as we do not know any values on the i^{th} node. As this is the case, we provide the algorithm with an initial guess of the boundary conditions at $\theta_{i,1}$ and $\theta_{i,k}$ and zeros for the rest. The resulting matrix is then inverted to give us the fixed point iteration scheme

$$U^{(m+1)} = A^{-1}U^{(m)}$$

For the values chosen in the current code, only about 8 iterations are needed before $|U^{(m+1)} - U^{(m)}| \leq 10^{-3}$. This method was settled upon as the method for solving the boundary layer equations in the mid-sixties in the USSR, unfortunately, that means the details of convergence for the fixed point iteration have been effectively lost to antiquity, and strained political relations. However, we can get a good idea of of the convergence by finding the eigenvalue at the initial guess of the boundary conditions and zero for all other nodes. The maximum eigenvalue for A^{-1} is 1, so it is marginally stable, this eigenvalue was calculated using numpy's linalg package. Now we can look at the continuity equation, since we are working with an x derivative, we can only keep it first order, discretizing we get

$$V_{i,j} = V_{i,j-1} - \frac{\Delta y_-}{\Delta x} * (U_{i,j} - U_{i-1,j})$$

This equation is also included in the iterative scheme. For the data and results below, we used the acceleration parameter set to 0 and a constant density to minimize troubleshooting time.

Finally, we have to determine the form of the boundary conditions. We can do this very easily while sticking to the tri-diagonal formulation: $a_{i,k/1}U_{i,k/1-1} + b_{i,k/1}U_{i,k/1} + c_{i,k/1+1} = d_{i,k/1}$. If we have Dirichlet boundary conditions (say $u(0) = 1$ or $u(L) = 1$) we notice that $a = c = 0$ and $d = 1$ and $b = 1$. Then with Neumann boundary conditions, we use only a first order approximation for the derivatives because that way we have the node that it's evaluated at in between the first and second node, versus a second order approximation which evaluates the derivative at the second node, which is not appropriate for a boundary condition. In this scheme, we only dealt with Dirichlet.

With our discretization in hand, we can code up the method in Python. Broadly, the loop structure is first, a global loop starting at the second x-node (this is because the profiles at $x = 0$ are set when the parameters. such as number of nodes, are set) contains an iteration loop which goes through the motions of applying the recursion relations derived above, until a fixed point is reached. When that happens, the simulation moves to the next x-node.

IV. RESULTS

The major result of a Boundary Layer computation is the streamwise velocity and temperature profile. The shape of these profiles is very recognizable, and graces the pages of many engineering textbooks. Our Python code was only able to produce velocity profiles, as adding temperature variation made the Python integrator to diverge as shown in fig.3.

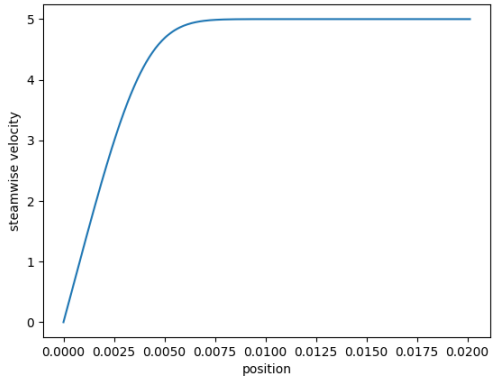

FIG. 1. Streamwise velocity profile calculated with no temperature gradient in Python

Of course, a "visual proof" is not particularly compelling evidence. To better test our method, we attempted to match the skin friction coefficient calculated in the Python via the finite difference scheme: $cf_i/2 = \frac{\nu}{U^2_{i,k-1}}\frac{U_{i,1}-U_{i,0}}{y_1-y_0}$ to eq.(15). By setting the number of x grid points $n = 2500$, number of y grid points $k = 300$, y spacing parameter $M = 1.008$, x spacing parameter $b = 1.0025$, we were able to generate fig.4

Python is not ideal as an algorithm solver for solving PDEs which involve small values. It was apparent that the error progression is much more prevalent in the Python code and is clearly shown in the velocity profile

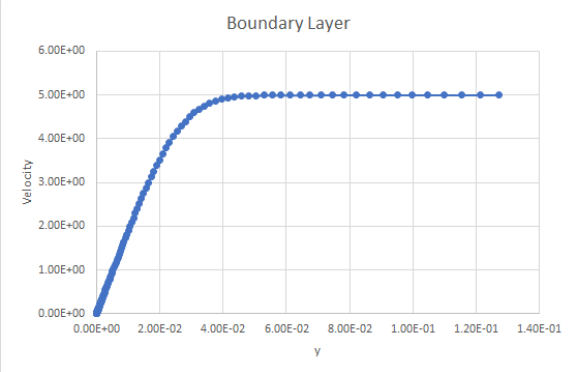

FIG. 2. Steady State Streamwise Velocity Profile using FORTRAN

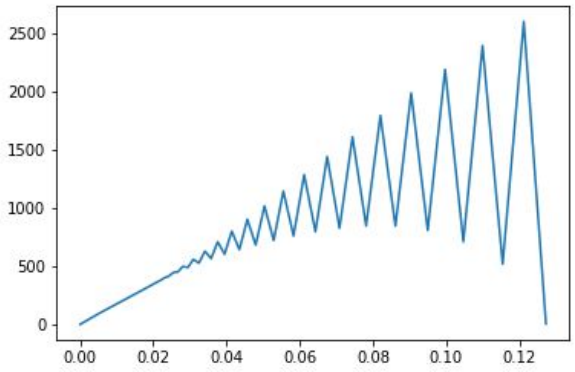

FIG. 3. Python Integrator Diverging when Temperature Variation is Added

result. The velocity profile is more difficult to obtain because of the more parameters and complexity involve in the discretized equation. However, both FORTRAN and MATLAB gives you more control over the error progression. The skin friction coefficent gives an excellent comparison between the Thomas Algorithm and the Blasius equation. As expected the absolute skin friction coefficient increases with increase in Reynolds number. The degree to which Python fails can be made clear by doing the same comparison using the same algorithm in FORTRAN, which is depicted in fig.5. This figure was generated in LabPlot2 which allowed for the functions to have their logarithms taken without changing the range of the plot.

The physical parameters used in Python were $L = 0.5$, $U_\infty = 5$, $\nu = 1.8 * 10^{-5}$.

V. CONCLUSION

The comparison between eq.(15) and the code were quite favorable, implying not only that this method valid, but that we implemented it correctly. This project achieved what it set out to do: familiarize Group 3 with numerical analysis of fluid flow. Of course, we dealt with

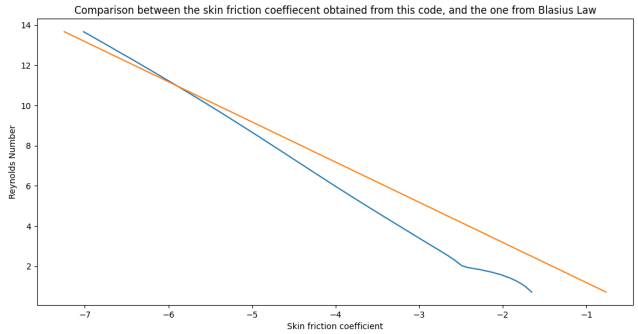

FIG. 4. Logarithm of the Skin Friction Coefficient calculated using Finite Differences in FORTRAN (Blue Line) Compared to the Logarithm Predicted Blasius Skin Friction (Orange Line)

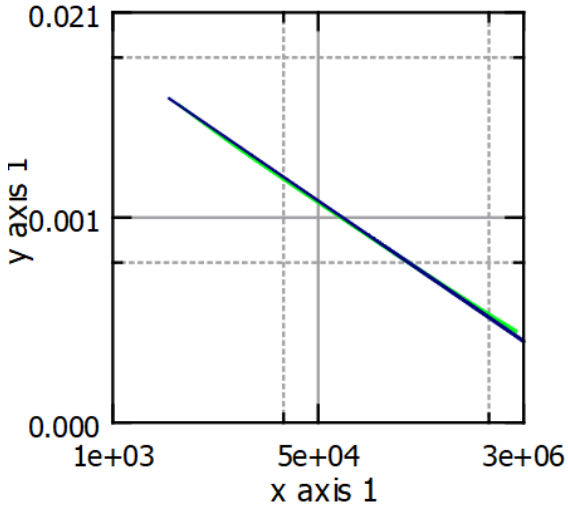

FIG. 5. Skin Friction Redone in FORTRAN, Old Data so Parameters are Unknown

an extraordinarily simple system. One in which there was no temperature, or pressure gradient, and where the fluid properties are constant throughout the flow. Despite this relative simplicity, the numerical ideas faced were extremely involved, such as introducing a fixed point iteration with convergence properties that seemingly never made it out of the USSR.

Finally, this project also revealed the utility of seemingly antiquated coding languages. By using this method in FORTRAN, it was possible (easy even) to add temperature calculation, variable density, and even a turbulence model, but in Python the Algorithm was brought to its knees with just temperature. This is likely due to some strange interaction that is unique to Python, but it nonetheless makes it hard to justify performing CFD in Python. Beyond the convergence difficulties, Python is also much slower, taking over a minute to do a calcula-

tion that took FORTRAN less than 5 seconds. Though we were expecting to find the antiquity of FORTRAN unwieldy, and unjustifiable, we found that FORTRAN is still the go-to for this line of work.

ACKNOWLEDGEMENT

A very enthusiastic thank you to Alexey Sakhnov of Novosibirsk State University for extensive help in the writing process of the code, and many helpful explanations of the numerical methods.

As well, Group 3 divided the work in the following fashion

- Sidney: Wrote the final code, wrote the rough draft of the report, finalized the report, derived Blasius solution via pen and paper
- Omar: Attempted to transfer initial code from FORTRAN to Python, matched Python code to Blasius solution, Troubleshot the code
- Hamza: Troubleshot the code, checked numerical solution of Blasius equation $2f''' + ff'' = 0$

[1] J. Wendt, J. Anderson, V. K. I. for Fluid Dynamics, G. Degrez, E. Dick, and R. Grundmann, *Computational Fluid Dynamics: An Introduction*, A Von Karman Institute book (Springer Berlin Heidelberg, 1996).
[2] G. Łukaszewicz and P. Kalita, *Navier–Stokes Equations: An Introduction with Applications*, Advances in Mechanics and Mathematics (Springer International Publishing, 2016).
[3] M. Avcar, Free vibration analysis of beams considering different geometric characteristics and boundary conditions, system **4**, 2 (2014).
[4] N. Curle, *The Laminar Boundary Layer Equations*, Dover Books on Physics (Dover Publications, 2017).
[5] H. Schlichting and K. Gersten, *Boundary-Layer Theory* (Springer Berlin Heidelberg, 2016).
[6] J. Miller, A. Musgrave, and G. Shishkin, A reynolds-uniform numerical method for the prandtl solution and its derivatives for stagnation line flow, International Journal for Numerical Methods in Fluids **43**, 881 (2003).
[7] J. Crank and P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, Mathematical Proceedings of the Cambridge Philosophical Society **43**, 50–67 (1947).
[8] M. Schobeiri, *Fluid Mechanics for Engineers: A Graduate Textbook* (Springer Berlin Heidelberg, 2010).
[9] R. Cortell, Numerical solutions of the classical blasius flat-plate problem, Applied Mathematics and Computation **170**, 706 (2005).