UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

# TREBALL DE FI DE GRAU

**TFG TITLE: Numerical solution of the Boundary Layer Equations**

**DEGREE: Grau en Enginyeria de Sistemes Aeroespacials**

**AUTHOR: Olga Catalan Aragall**

**ADVISOR: Fernando Pablo Mellibovsky**

**DATE: July 17, 2022**

**Títol:** Solució numèrica de les equacions de la capa límit
**Autor:** Olga Catalan Aragall

**Director:** Fernando Pablo Mellibovsky

**Data:** 17 de juliol de 2022

**Resum**

L'objectiu d'aquest projecte és desenvolupar un codi que sigui capaç de resoldre numèricament les equacions de Navier-Stokes parabolitzades que regeixen la dinàmica de flux dins la capa límit bidimensional. Utilitzant una escala d'auto-similitud a la formulació de la funció de corrent i donades les condicions de límit exterior del flux aigües amunt i no viscoses adequades, el codi resol la capa límit i calcula les seves propietats característiques.

Per començar, les equacions bidimensionals de la capa límit s'han expressat amb la formulació de la funció de corrent i s'ha aplicat un canvi de coordenades de tipus Falkner-Skan per expressar-les en variables de similitud. A continuació, l'equació de tercer ordre resultant s'ha reduït al primer ordre seguint un enfocament estàndard i el sistema s'ha discretitzat a l'espai mitjançant diferències finites.

El codi s'ha provat amb solucions de referència per a la validació. La solució de Blasius, que es desenvolupa sobre una placa plana amb incidència zero, i la solució de capa límit laminar del punt de recés s'han reproduït satisfactòriament. Alguns problemes resolts prèviament amb el mètode integral aproximat s'han revisat amb el codi per tal de comprovar la precisió del primer.

El codi s'ha adaptat per acceptar les condicions del límit del flux exterior en forma tant d'expressions matemàtiques com de mostres discretes de corrent de la distribució de velocitats exterior invisícida. També s'ha codificat un model de turbulència simple per resoldre les capes límit turbulentes i laminars i s'ha implementat un criteri de transició natural. Es preveu degudament el comportament típic de les capes límit turbulentes, com ara la seva tendència a resistir la separació millor que les capes límit laminars.

Finalment, s'han introduït solucions de flux invísid passats per a perfils alars amb el programari Xfoil al codi de la capa límit per tal de calcular l'arrossegament de fricció i detectar la separació. Els resultats coincideixen bé amb la literatura, que valida encara més la precisió del codi de la capa límit.

**Title :** Numerical solution of the Bounadry Layer Equations
**Author:** Olga Catalan Aragall

**Advisor:** Fernando Pablo Mellibovsky

**Date:** July 17, 2022

**Overview**

The aim of this project is to develop a code that is capable of solving numerically the parabolised Navier-Stokes equations that govern the flow dynamics within two-dimensional boundary layers. Using a self-similarity scaling on the streamfunction formulation and given appropriate upstream and inviscid outer flow boundary conditions, the code solves the boundary layer and computes its characteristic properties.

To begin with, the two-dimensional boundary layer equations have been cast in the streamfunction formulation and a Falkner-Skan-type coordinate change has been applied to express them in similarity variables. Next, the resulting third order equation has been reduced to first order following a standard approach, and the system is discretized in space using finite differences.

The code has been tested against benchmark solutions for validation. The Blasius solution, which develops on a flat plate at zero incidence, and the stagnation point laminar boundary layer solution have been satisfactorily reproduced. Some problems previously solved with the approximate integral method have been revisited using the code to check the accuracy of the former.

The code has also been adapted to accept outer flow boundary conditions in the form of both closed-form mathematical expressions or discrete streamwise samplings of the inviscid outer streamwise velocity distribution. A simple turbulence model has also been coded to resolve turbulent as well as laminar boundary layers and a criterion for natural transition has also been implemented. Typical behavior of turbulent boundary layers, such as their tendency to resist separation better than laminar boundary layers, is duly predicted.

Finally, inviscid flow solutions past airfoils obtained with the software Xfoil have been fed into the boundary layer code to compute friction drag and detect separation. Results agree well with the literature, which further validates the accuracy of the boundary layer code.

To my family and friends, for encouraging me and always showing their support.
To everyone who has helped me during the process and, in particular, to my thesis advisor, thank you for your guidance and help.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# INTRODUCTION

When discussing high-speed fluid mechanics and aerodynamics one must always eventually resort to the boundary layer concept. The boundary layer determines the aerodynamic drag of an object and, because of that several methods and studies focus on controlling its development. For instance, when the boundary layer separates a wake is formed. If the separation point could be delayed, the wake would reduce, and thus the drag would also decrease.

Studying the properties of the boundary layer on different surfaces, such as airfoils, can help in the process of optimizing the airfoil as it is desired to have a high lift-to-drag ratio and minimize the drag, which impacts the fuel consumption of the aircraft and reduces CO2 emissions. This is an important point to consider because of the increase in air traffic and the climatic emergency.

But what exactly is the boundary layer? The boundary layer is described to be the closest region of a body subjected to shear forces. These shear forces are related to viscosity, which denotes the resistance of a fluid.

Notwithstanding that viscosity values can be very small, in the boundary layer region must be considered as they can have a big impact on the fluid behavior.

The objective of this project is to develop a code capable of solving the Navier-Stokes equations in the boundary layer region. Navier-Stokes equations are the governing equations of flow movements and they rarely admit analytic closed-expression solutions, consequently, numerical methods are applied to solve them successfully.

To achieve this, [6], [7] and [8] have been followed to solve the equations for this particular case. Additionally, some approximations are considered. For simplicity, it is assumed that the flow is two-dimensional, steady, and incompressible, leading to the simplified version of the x-momentum and the continuity equations hereafter.

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{dp}{dx}+\nu\frac{\partial}{\partial y}(b\frac{\partial u}{\partial y}) \tag{1}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{2}$$

These are the local boundary layer equations, which result from applying the boundary layer hypothesis to the Navier-Stokes equations and represent the starting point of the project. To these equations, several methods and transformations are applied to pose a proper expression for programming its resolution.

The project is divided into three parts. The first chapter is dedicated to giving the theoretical background knowledge: the basic concepts, some important considerations, and some examples.

The second chapter is exclusively devoted to explaining code development. First, the equations are developed, then the grid to solve is proposed to solve them, to end with the analysis of features of the code.

Finally, the third and final chapter is focused on the validation of the code. In this chapter, different cases are tested and it is analyzed whether the results are correct or not, comparing the results with other software or already tabulated solutions.

# CHAPTER 1. BOUNDARY LAYER CONCEPTS

According to the Encyclopedia Britannica [12] the boundary layer is:

*"A thin layer of a flowing gas or liquid in contact with a surface such as that of an airplane wing or of the inside of a pipe. The fluid in the boundary layer is subjected to shearing forces. A range of velocities exists across the boundary layer from maximum to zero, provided the fluid is in contact with the surface. "* (Britannica, T. Editors of Encyclopaedia, March 6 2020)

In other words, the boundary layer is the closest region to a solid wall or surface where viscosity or shear forces must be considered. In general, when having a flow with a high value of the Reynolds number (a non-dimensional parameter weighing the ratio of inertial to viscous forces), a distinction can be drawn between a region where inviscid flow is assumed and a region where viscosity must be considered (the boundary layer region).

This chapter will present the basic concepts and examples to get a general idea of the boundary layer concept and related terms. It will explain, first, the viscosity concept, secondly, the Reynolds number, and last but not least, laminar and turbulent flows and the separation point.

Several references are consulted to properly define the terms, empathizing in sources [8] [11] [18].

## 1.1. Viscosity

A flow is considered ideal if it is incompressible and inviscid (zero viscosity) whereas in a real flow viscous forces between layers, or shear stresses, are taken into account.
Because of that, when dealing with a real flow that encounters a solid body or surface, the non-slip condition applies; meaning that the fluid adheres to (can not slide on) the surface. If dealing with an ideal flow, the fluid would just slip away.

Ideal flows can be used as suitable approximations for some specific problems with very low viscosity with the major advantage that, as considered incompressible and inviscid, the resolution is much easier. However, in some cases, even though the viscous forces are very small, the results can be significantly different and viscosity has to be considered.

Additionally, when talking of viscosity, Newtonian and non-Newtonian fluids can be defined. Newtonian fluids have constant viscosity, meaning that there is a linear behavior between shear stresses and the velocity gradient. On the other hand, fluids that do not have constant viscosity are known as non-Newtonian. For instance, water or gasoline are considered Newtonian fluids, and butter or quicksand are non-Newtonian.

Finally, a commonly used parameter related to viscosity is the kinematic viscosity. Kinematic viscosity [$\nu$] is the ratio between the dynamic viscosity [$\mu$] and the density [$\rho$] of the given fluid.

$$\nu = \frac{\mu}{\rho} \tag{1.1}$$

### 1.1.1. Couette flow

By way of example, the Couette flow can be used for introducing the viscosity in a clear case.
The Couette flow is defined by considering two parallel long plates, being the lower plate at rest and the top plate moving at a constant velocity. Between the plates, a laminar viscous flow flows and thus, the non-slip condition applies; causing the adherence of the fluid to the plate. This results in zero velocity when it is in contact with the lower plate and constant velocity (U) in the top plate.

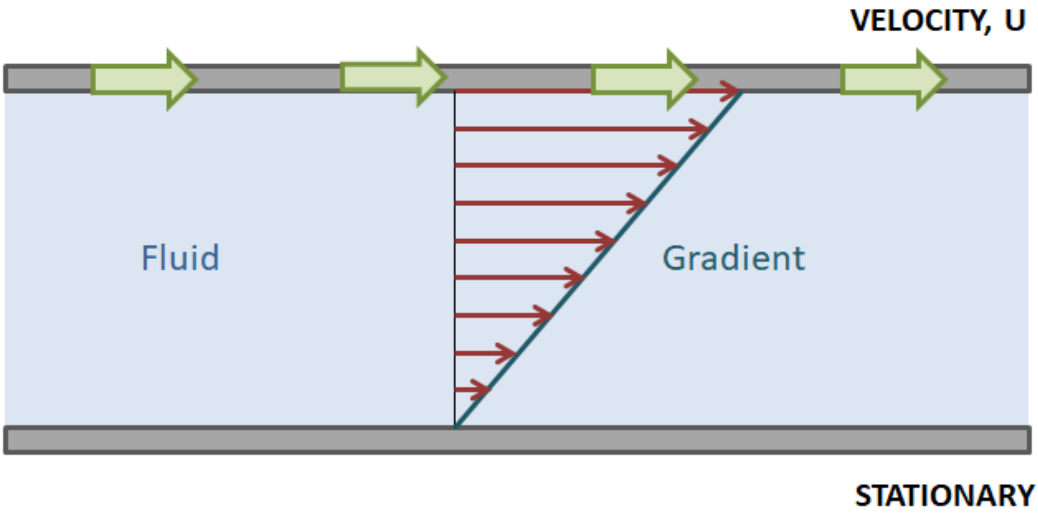



Figure 1.1: Couette Flow diagram.

The acting force causing these changes of velocity along the fluid is the shear stress. This force is going to be dependent on the velocity, the distance between the plates, and the type of fluid (air, water, etc). These dependencies can be observed in Newton's law, valid for Newtonian fluids, as the name implies.
Newton's law of friction states that shear stress ($\tau$) can be computed as:

$$\tau = \mu \frac{du}{dy} \tag{1.2}$$

Being $\frac{du}{dy}$ the wall-normal gradient of streamwise velocity. When changing the gradient, the slope of the blue line represented in Figure 1.1 is modified. If the velocity decreases, the slope gets closer to a vertical straight line. Conversely, if the velocity increases the velocity, the slope gets flatter.

## 1.2. Reynolds Number

Reynolds number is a dimensionless number resulting from the ratio between the inertial forces and the friction forces, without considering gravitational and elastic forces.

If the gravitational force is considered, Froude number is obtained $Fr = \frac{V}{\sqrt{gd}}$ and if considered elastic forces, for instance, because of the high speeds inducing the compressibility of the fluid, Mach number $Ma = \frac{V}{c}$ is obtained. Denoting $V$ the velocity in $m/s$, $g$ the acceleration due to gravity in $m/s^2$, $d$ the depth in $m$, and $c$ the speed of sound.

The Reynolds number is computed as follows, either expressed with the kinematic viscosity or the dynamic viscosity.

$$Re = \frac{Vd}{\mu} = \frac{Vd}{\nu} \tag{1.3}$$

Being $V$ the velocity of the fluid and $d$ the characteristic dimension, for instance, the length that has traveled the fluid.

Reynolds number is a greatly used parameter since if two flows have the same Reynolds it is said that they have a dynamic and geometric similarity. Because of this property, it is possible to study, for instance, scaled models of airfoils in wind tunnels. Working with scaled models enables testing different conditions and observing the behavior of that model in a more controlled space, safer, and in cheaper conditions.

Finally, a lower Reynolds number would be associated with laminar flows, and high numbers of Reynolds with turbulent flows. The next section will present these two types of flows and some of their characteristic features are going to be pointed out.

## 1.3. Laminar and turbulent flows

Two types of flows can be distinguished: laminar flow and turbulent flow. These concepts can be easily identified with the O. Reynolds experiment (1883) [11] which consisted in introducing colored liquid into a pipe of flowing water. Because of the dye, it was possible to observe the behavior of the flowing water. At lower velocities, a colored flow showed up, parallel to the longitudinal axis of the pipe: a laminar flow. However, when increasing enough the velocity, the colored flow started to move irregularly and randomly, obtaining a turbulent flow.

Figures 1.2 and 1.3 shows a representation of Reynolds experiment. In the first image, a laminar flow can be observed, whereas the second image displays the irregularities caused by the increased velocity and getting a turbulent flow.

Therefore, a critical number of Reynolds can be defined. Below the critical Reynolds, the fluid behaves in a laminar way, whereas above them it is turbulent. For instance, the critical Reynolds for a flat plate with zero angles of incidence is approximately 5E5, yet this value is not unique and other factors may influence it. For example, the critical Reynolds can be delayed to 3E6 keeping the free stream quiet or polishing the wall. [9]

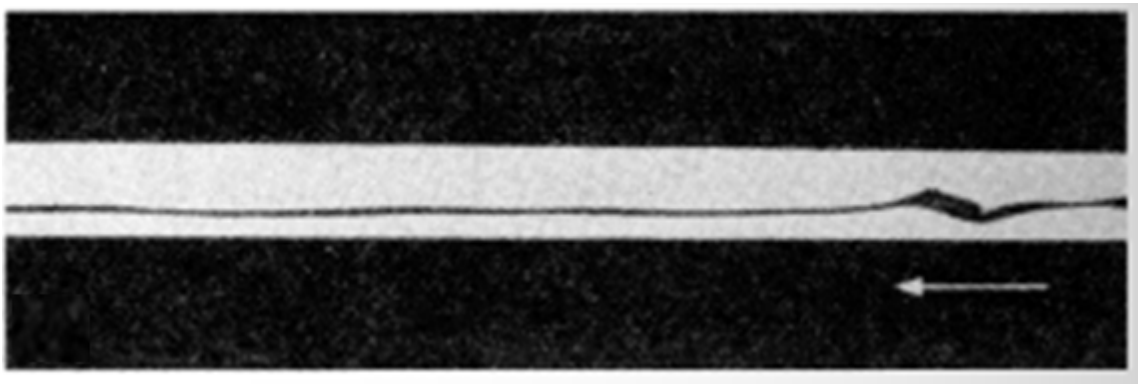


Figure 1.2: Reynolds experiment showing laminar flow. [11]

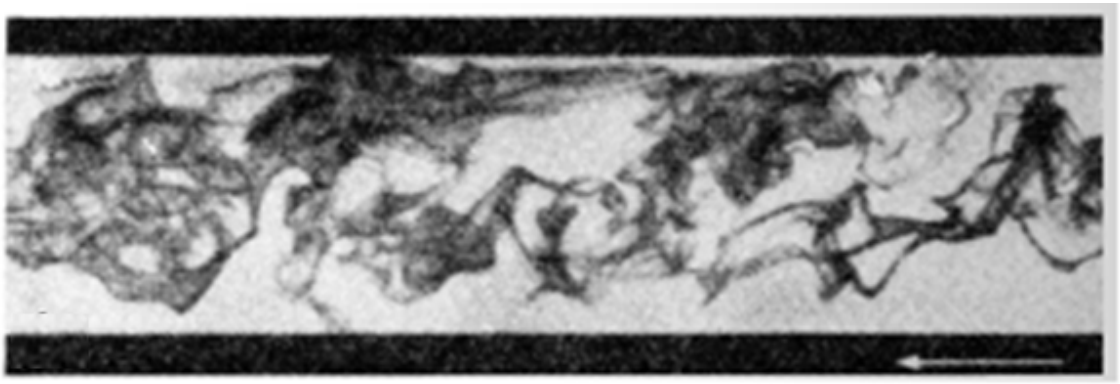


Figure 1.3: Reynolds experiment showing turbulent flow. [11]

Additionally, because of these random and irregular properties of turbulent flows, it is not possible to obtain a theory as developed as the laminar flow theory. The theories of turbulent flows are based on models with empirical knowledge.

### 1.3.1. Characteristic Properties

Some interesting parameters that give useful information about the fluid and that can be used for checking if a given solution is reasonable are the skin friction coefficient, displacement thickness, and momentum thickness. [18]

The wall shear stresses can be computed by applying Newton's law at the surface and it indicates the stresses at the wall as a consequence of the viscosity.

$$\tau_w = \mu(\frac{\partial u}{\partial y})_w \tag{1.4}$$

Laminar shear stresses at the surface are smaller than turbulent shear stresses.

Related with $\tau_w$, the skin friction coefficient can be defined as:

$$C_f = \frac{\tau_w}{\frac{1}{2}\rho_e u_e^2} \tag{1.5}$$

The skin friction coefficient is the fraction between shear stress and the pressure of an inviscid streamline just outside of the region where viscosity must be considered, resulting in a dimensionless parameter. Additionally, by computing the integral of the skin friction coefficient on the surface of the body, the skin friction drag is obtained. The skin friction drag is the drag force produced by viscous shear at the wall.

The skin friction coefficient decreases with Reynolds while there is no transition to turbulence.

Other characteristic properties are the displacement thickness and momentum thickness, computed as follows, respectively.

$$\delta_1 = \int_0^\delta (1 - \frac{\rho u}{\rho_e u_e}) dy \tag{1.6}$$

$$\theta = \int_0^\delta \frac{\rho u}{\rho_e u_e} (1 - \frac{u}{u_e}) dy \tag{1.7}$$

Both thicknesses quantify the viscous blockage effect by comparing a real flow with an ideal flow, which would be equivalent (a viscous flow and an inviscid) by determining the distance that should be displaced the surface. One through the mass flux effects, the other through the momentum flux effects.

According to the Encyclopedia of Physical Science and Technology [13]:

*"The displacement thickness represents the distance by which the body should be displaced in order to represent the boundary layer effects in the equivalent inviscid flow . [...] The momentum thickness is the thickness which is added to the displacement thickness in order to have the same flux of momentum in the real flow and in the fictitious flow."* (Encyclopedia of Physical Science and Technology, Third Edition, 2003)

Finally, the shape factor is the ratio between the displacement thickness and the momentum thickness.

$$H = \frac{\delta_1}{\theta} \tag{1.8}$$

This value can be used to identify whether the flow is laminar or turbulent. For instance, for a flat plate at zero angle of incidence, the shape factor remains constant along the development of the plate, being H=2.591 provided that the flow is laminar. If the flow turns turbulent this value decreases up to around 1.5, which reflects the change from laminar to turbulent flow.

As referred to just now, this transition point from laminar to turbulent is of great interest, as it generates changes in the behavior of the fluid and the characteristic properties. The following subsection will analyze this transition point and how it can be predicted.

### 1.3.2. Transition from laminar to turbulent flow

The transition from laminar flow to turbulent flow can be generated by several reasons or a combination of factors. For instance, it might be caused by the increase of Reynolds number, some external perturbations, the roughness of the surface, pressure gradient, etc.

Some used transition criteria are Michel's, Granville, or $e^n$ methods. Notwithstanding the great use of these methods, an empirical correlation provided in Chapter 5 of Modeling and Computation of Boundary-Layer Flows [7] will be implemented.

$$Re_{\theta_{tr}} = 1.174(1 + \frac{22400}{Re_{x_{tr}}})Re_{x_{tr}}^{0.46} \tag{1.9}$$

This model offers a criterion to determine the transition from laminar to turbulent flow in a simple equation form. Moreover, it is a free transition criterion, meaning that only if it is reached a point included in equation 1.9, a transition would occur. However, it can also be imposed a transition point or the Reynolds number at which turbulence is wished to start. Both methods can be used together in combination.

Turning now to methods for simulating turbulence, the proposed method in Modeling and Computation of Boundary-Layer Flows [7] is going to be used, specifically the Cebeci-Smith Turbulence model. This model simulates turbulence by computing eddy viscosity terms of the inner and outer regions of the boundary layer from modeled experimental data. This eddy viscosity term, represented as $\varepsilon_m$, is included in the momentum equation modifying the wall-normal diffusion term.

The model distinguishes between the inner and the outer region. The inner region is the closest region to the wall or surface where the fluid is directly influenced by the surface. On the other hand, the outer region is governed by the effects of free-stream turbulence. Thus, the overlap layer is the transitioning zone between inner and outer regions. However, to simplify the turbulence model an instantaneous change of the region is to be considered. The border point between the inner and the outer regions is computed as the 10% - 15% of the boundary layer thickness, according to references [16] and [17].

The operating and extended development of this model is going to be explained more deeply in the next chapter when dealing with the functions of the code.

## 1.4. Separation of the boundary layer

Once a general idea of the boundary layer concept and its characteristics are exposed, it may happen that when developing the boundary layer it separates from the surface and a wake starts developing. This is what happens when the boundary layer detaches.

When the fluid intercepts a body or surface, it adheres to the surface following its shape yet sometimes the flow is not able to follow the curvature of the wall and it is separated.

For instance when there is a flow around a cylinder, the fluid accelerates, having a negative pressure gradient (pressure has been converted into kinetic energy) but once reaches the maximum velocity, around the mid-point, the flow starts to slow down and the pressure gradient turns positive (kinetic energy has transformed into pressure). In this second region where the fluid is decelerating and losing kinematic energy, it reaches a point where it can not overcome the pressure distribution and the fluid is pushed backward making the boundary layer lose all the momentum it has available in the near-wall region and causing it to separate. [19]

Figure 1.4 depicts the explained situation.

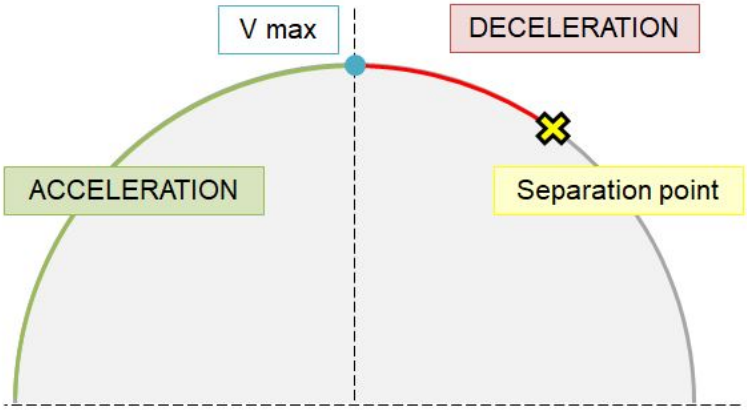



Figure 1.4: Separation point diagram of a cylinder.

This scenario can be extrapolated to other solid walls or surfaces, causing points where the boundary layer detaches and the streamlines leave the surface.

Laminar flows have small momentum and are not able to support strong adverse pressure gradient, inducing an earlier separation. However, turbulent flows have higher momentum and are better able to resist the adverse pressure gradient; meaning that the separation point is delayed.

This is the main reason why golf balls have dimples. The dimples cause turbulence and are thus better resistant to adverse pressure gradient since the separation point is retarded, causing smaller wakes. Adding dimples reduces the drag and makes the ball reach further distances.

The position where this separation occurs is when the shear stresses at the wall turn 0.

$$\tau_w = \mu(\frac{\partial u}{\partial y})_w = 0 \tag{1.10}$$

# CHAPTER 2. FORMULATION AND NUMERICAL APPROACH

In this second chapter, every aspect of the boundary layer program will be explained.

The main objective of the program is to develop a functioning code able to solve the Navier Stokes equations for the boundary layer and, at the same time, obtain some characteristic parameters and graphics that enable the analysis of it. Furthermore, the code should be robust, and at the same time that it should allow variation in the inputs and the variables.

For the development of this code, MATLAB will be used given its wide community and functionalities. Besides, the option of running code in live scripts eases the validation by sections of the code point by point and organizes the sections more visually. Nevertheless, a structure based on classes is simulated using cell arrays. In this way, if it is needed to change the programming language, for instance to a C sharp application, the structure should be compatible and no big reformulations would be required.

The operating of the code is based on defining a group of equations to solve the boundary layer scenario. Because there is no analytical solution, numerical methods are applied; meaning that a grid is defined (of N horizontal divisions by J vertical divisions). Each position of the grid would have its properties as the x-coordinate or the y-coordinate, but also each position would have associated important boundary layer field variables, such as the dimensionless velocity at that position of the grid. These parameters are the unknown values referred to when talking about the solution of the boundary layer, and the code is constantly sweeping along the diverse positions of the grid trying to obtain more accurate and precise results of these values; following a set of equations. The main followed guidelines for the resolution of the equations are drawn from [6] [7] [8].

In the following sections the governing equations and their development, the grid structure, and the code (including the main script and the most important functions) will be dealt with.

## 2.1. Boundary Layer equations

For obtaining the boundary layer solution, the momentum and continuity equations have been solved using numerical methods. Detailed procedures and further explanations step-by-step of how the final equations are reached can be found in Appendix A. However, the main steps used for obtaining the desired equation will be detailed below.

Starting with the momentum and continuity equations:

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}=-\frac{1}{\rho}\frac{dp}{dx}+\nu\frac{\partial}{\partial y}(b\frac{\partial u}{\partial y}) \tag{2.1}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{2.2}$$

Where $b$, computed as $b=1+\varepsilon_m/\nu$, denotes if the fluid is laminar or turbulent. For laminar flows b=1 and $\varepsilon_m=0$, being $\varepsilon_m$ the eddy viscosity defined with Cebeci-Smith Turbulence model [7]. The concrete definition of this parameter is going to be explained in the turbulence function section.

These equations can not be directly solved; numerical methods and transformations must be applied to obtain a valid and easier computation equation. To start this transformation process, the definition of stream function ($\psi$) is used to rewrite the equation. Using this concept enables working with a single variable of a higher order, rather than the two components of velocity, being $u = \frac{\partial \psi}{\partial y}$ and $v = -\frac{\partial \psi}{\partial x}$. Besides, the stream function formulation satisfies the continuity equation, hence the continuity equation (2.2) is no longer required. When applying this concept, the following expression is generated, where the prime denotes differentiation with respect to y.

$$\nu(b\psi'')' + u_e \frac{du_e}{dx} = \psi' \frac{\partial \psi'}{\partial x} - \psi'' \frac{\partial \psi}{\partial x} \tag{2.3}$$

Secondly, Falkner-Skan transformation is applied using the variables:

$$\eta = y\sqrt{\frac{u_e}{\nu x}} \tag{2.4}$$

$$\psi(x,y) = \sqrt{u_e \nu x} f(x,\eta) \tag{2.5}$$

Applying this change of variable enables controlling the growing magnitude of the boundary layer, as boundary layer thickness increases when moving to higher x directions. Because of that, obtaining an accurate solution would require lots of steps. To reduce the number of steps without deteriorating the accuracy, a transformation must be applied. For instance, with the Falkner-Skan transformation, the increase of the boundary layer is reduced and thus wider steps can be considered.

Moreover, $\xi = x/L$, but as in general it is working with a unitary value of x, $\xi = x$.

Another important parameter is m. It refers to the dimensionless pressure gradient. This value is going to change depending on the outer streamwise velocity distribution and it is computed as expressed hereunder.

$$m = \frac{\xi}{u_e} \frac{du_e}{d\xi} \tag{2.6}$$

Applying all these changes and now denoting prime as differentiation with respect to $\eta$, the momentum equation can be rewritten as:

$$(bf'')' + \frac{m+1}{2} f f'' + m(1-(f')^2) = \xi(f' \frac{\partial f'}{\partial \xi} - f'' \frac{\partial f}{\partial \xi}) \tag{2.7}$$

With boundary conditions: $\eta = 0 \rightarrow f' = 0$, following the no-slip condition, $f(\xi,0) = f_w(\xi)$ and at $\eta = \eta_e \rightarrow f' = 1$, as the fluid re-encounters the outer velocity.

Hitherto, a simplified equation has been obtained, but it can not be directly solved yet. A numerical formulation must be posed to solve the equation. For instance, Keller's box method.

This method consists of an initial change of variable to reduce the order of the system to a first-order one: renaming $f' = u$ and $u' = v$. After that, a grid implementation must be applied (the characteristics of this grid are going to be further explained in the next section).

Applying the central difference derivatives expressions at the grid, using midpoint centering and denoting $j$ the iterations in the vertical direction ($\eta$) and $n$ in the horizontal (x or $\xi$ ), it can be expressed the following equation.

$$\frac{f_j^n - f_{j-1}^n}{h_j} = \frac{u_j^n + u_{j-1}^n}{2} = u_{j-1/2}^n \tag{2.8}$$

$$\frac{u_j^n - u_{j-1}^n}{h_j} = \frac{v_j^n + v_{j-1}^n}{2} = v_{j-1/2}^n \tag{2.9}$$

Being $h_j$ the increments in the $\eta$ direction.

Finally, equation 2.7 is rewritten using the concept of centered-difference derivatives, writing the equation at $(x^{n-1/2}, \eta_{j-1/2})$, but an initial centering must be done at $(x^{n-1/2}, \eta)$.

Denoting $L$ the left part of the equation 2.7 and taking common factors (renaming them as $\alpha$):

$$((bv)')^n + \alpha_1 (fv)^n + m^n - \alpha_2 (u^n)^2 = -L^{n-1} + \alpha^n(-(u^{n-1})^2 + v^n f^{n-1} - f^n v^{n-1} + (fv)^{n-1}) \tag{2.10}$$

With:

$$\alpha^n = \frac{x^{n-1/2}}{k_n}, \qquad \alpha_1 = \frac{m^n + 1}{2} + \alpha_n, \qquad \alpha_2 = m^n + \alpha_n \tag{2.11}$$

Being $k_n$ the grid increments in x-direction.

Lastly, grouping the following terms:

$$R^{n-1} = -L^{n-1} + \alpha_n((fv)^{n-1} - (u^{n-1})^2) - m^n \tag{2.12}$$

The equation centered at $(x^{n-1/2}, \eta)$ is expressed as:

$$((bv)')^n + \alpha_1 (fv)^n - \alpha_2 (u^n)^2 + \alpha^n (v^{n-1} f^n - f^{n-1} v^n) = R^{n-1} \tag{2.13}$$

Finally, applying the centering in the $\eta$ coordinate to obtain the equation at $(x^{n-1/2}, \eta_{j-1/2})$:

$$\frac{1}{h_j}(b_j^n v_j^n - b_{j-1}^n v_{j-1}^n) + \alpha_1 (fv)_{j-1/2}^n - \alpha_2 (u^2)_{j-1/2}^n + \alpha^n (v_{j-1/2}^{n-1} f_{j-1/2}^n - f_{j-1/2}^{n-1} v_{j-1/2}^n) = R_{j-1/2}^{n-1} \tag{2.14}$$

With:

$$\begin{aligned} R_{j-1/2}^{n-1} &= -L_{j-1/2}^{n-1} + \alpha^n((fv)_{j-1/2}^{n-1} - (u^2)_{j-1/2}^{n-1}) - m^n \\ L_{j-1/2}^{n-1} &= \left[\frac{1}{h_j}(b_j v_j - b_{j-1} v_{j-1}) + \frac{m+1}{2}(fv)_{j-1/2} + m(1 - (u^2)_{j-1/2})\right]^{n-1} \end{aligned} \tag{2.15}$$

However, this procedure leads to a nonlinear system. To linearise it Newton's method is used, introducing iterates of the $f, u$, and $v$ variables expressed as follows. Being the next approximate value, the value of that variable at the previous iteration plus an increment. In the following expression, $t$ denotes this iteration.

The $n$ term denoting the column will be suppressed, as the code obtains the solution column by column, meaning that by default $n$ is going to be the already known working column. Except for those which will be specifically indicated, as $n-1$.

$$\begin{aligned} f_j^{t+1} &= f_j^t + \delta f_j^t \\ u_j^{t+1} &= u_j^t + \delta u_j^t \\ v_j^{t+1} &= v_j^t + \delta v_j^t \end{aligned} \tag{2.16}$$

Now, using this concept and substituting in the numerical equation the increments, and dropping the squared terms to linearise the expression; the final equation that is going to be coded is obtained, where the unknown values are the increments. These increments are going to be repeatedly computed until a good enough solution is found.

Lastly, for coding the equation more efficiently, it has been expressed in matrices form, taking the common factor of the increments.

$$A\vec{\delta} = \vec{r} \tag{2.17}$$

Being A a matrix composed of matrices, where each matrix represents the coefficients values of the increments. Each row of the matrices denotes the equations 2.8, 2.14 and 2.9.

$\vec{\delta}$ is the vector with the unknown values of the increments and $\vec{r}$ is the vector with the remaining terms or residual part.

Expressed extensively:

$$\begin{vmatrix} A_0 & C_0 & & & & & & & \\ B_1 & A_1 & C_1 & & & & & & \\ & \vdots & \vdots & \vdots & & & & & \\ & & & B_j & A_j & C_j & & & \\ & & & & \vdots & \vdots & \vdots & & \\ & & & & & B_{J-1} & A_{J-1} & C_{J-1} & \\ & & & & & & B_J & A_J \end{vmatrix} \begin{vmatrix} \vec{\delta}_0 \\ \vec{\delta}_1 \\ \vdots \\ \vec{\delta}_j \\ \vdots \\ \vec{\delta}_{j-1} \\ \vec{\delta}_J \end{vmatrix} = \begin{vmatrix} \vec{r}_0 \\ \vec{r}_1 \\ \vdots \\ \vec{r}_j \\ \vdots \\ \vec{r}_{j-1} \\ \vec{r}_J \end{vmatrix} \tag{2.18}$$

Being A, B and C matrices:

$$A_0 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & -h_1/2 \end{vmatrix} \quad A_j = \begin{vmatrix} 1 & -h_j/2 & 0 \\ (s_3)_j & (s_5)_j & (s_1)_j \\ 0 & -1 & -h_{j+1}/2 \end{vmatrix} \quad A_J = \begin{vmatrix} 1 & -h_J/2 & 0 \\ (s_3)_J & (s_5)_J & (s_1)_J \\ 0 & 1 & 0 \end{vmatrix}$$

$$B_j = \begin{vmatrix} -1 & -h_j/2 & 0 \\ (s_4)_j & (s_6)_j & (s_2)_j \\ 0 & 0 & 0 \end{vmatrix} \quad C_j = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & -h_{j+1}/2 \end{vmatrix} \tag{2.19}$$

$$(s_1)_j = h_j^{-1} b_j + \frac{\alpha_1}{2} f_j - \frac{\alpha^n}{2} f_{j-1/2}^{n-1} \tag{2.20a}$$

$$(s_2)_j = -h_j^{-1} b_{j-1} + \frac{\alpha_1}{2} f_{j-1} - \frac{\alpha^n}{2} f_{j-1/2}^{n-1} \tag{2.20b}$$

$$(s_3)_j = \frac{\alpha_1}{2} v_j + \frac{\alpha^n}{2} v_{j-1/2}^{n-1} \tag{2.20c}$$

$$(s_4)_j = \frac{\alpha_1}{2} v_{j-1} + \frac{\alpha^n}{2} v_{j-1/2}^{n-1} \tag{2.20d}$$

$$(s_5)_j = -\alpha_2 u_j \tag{2.20e}$$

$$(s_6)_j = -\alpha_2 u_{j-1} \tag{2.20f}$$

And each element of the residual terms vector:

$$(r_1)_j = f_{j-1} - f_j + h_j u_{j-1/2} \tag{2.21a}$$

$$(r_3)_{j-1} = u_{j-1} - u_j + h_j v_{j-1/2} \tag{2.21b}$$

$$\begin{aligned}(r_2)_j = R_{j-1/2}^{n-1} - [&\frac{1}{h_j}(b_j v_j - b_{j-1} v_{j-1}) + \alpha_1 (fv)_{j-1/2} \\ &- \alpha_2 (u^2)_{j-1/2} + \alpha^n (v_{j-1/2}^{n-1} f_{j-1/2} - f_{j-1/2}^{n-1} v_{j-1/2}))]\end{aligned} \tag{2.21c}$$

Taking into account that:

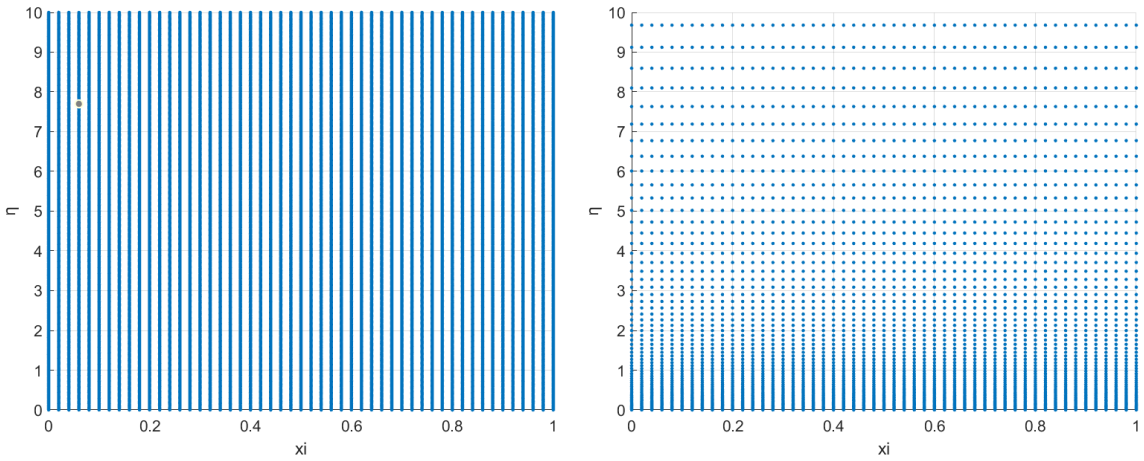
$$r_0 = \begin{vmatrix} 0 \\ 0 \\ (r_3)_0 \end{vmatrix} \quad r_J = \begin{vmatrix} (r_1)_J \\ (r_2)_J \\ 0 \end{vmatrix} \tag{2.22}$$

## 2.2. Grid representation

As had been seen in the previous section, a numerical solution has been implemented to solve numerically the equations. The axis of the grid are the $\xi$ and the $\eta$, being the increments in $\eta$ direction denoted by $h_j$ and the increments in $\xi$ denoted by $k_n$, and the parameters J and N denoting the number of vertical and horizontal divisions, respectively.

Additionally, when dealing with a boundary layer, a high grid density would be needed to solve the given scenario. In order to not have to work with huge grids, a parameter K is defined. K indicates the homogeneity of the grid, being K=1 a homogeneous grid and, for instance, K=1.05 a non-homogeneous grid. This enables establishing a grid with more points near the surface and fewer points when reaching the outer zone of the boundary layer.

Figure 2.1 shows the variation of K value. In the left figure, it can be observed a homogeneous grid with J=1000 and N=50, whereas in the right figure a non-homogeneous grid (K=1.06) with J= 70 and N=50 values; reaching both grids $\eta_e = 10$.




(a) K=1 (b) K=1.06

Figure 2.1: Grid representations for different K values.

Having non-homogeneous grids causes variations in $h_j$. Because of that, the initial increment of the grid is set ($h_1$) and the following increments are computed as $h_j = Kh_{j-1}$.

If it is a homogeneous grid (K=1) the increments are going to be equal along the grid, yet if it is not homogeneous at the positions closer to the surface the grid is going to be denser, as had been seen in the previous figure.

Another point to note is the sizing of the different parameters J, K, $h_1$ and the external value of $\eta$ ($\eta_e$) for non-homogeneous grids. Following the equation given in Chapter 7 of Computational Fluid Dynamics for Engineers [8], it has been expressed:

$$J = \frac{ln(1+(K-1)(\eta_e/h_1))}{lnK} + 1 \tag{2.23}$$

Some typical values fulfilling equation 2.23 of $\eta_e$ is around 10 and $h_1$ 0.01. The J value is considered as an integer closer to the solution. Note that if dealing with a turbulent flow $\eta_e$ would necessarily have to be increased.

Finally, one last clarification regarding the grid resolution is due. In this numerical method, the grid is not solved point by point, it is solved column by column. In other words, it is defined the equation 2.18 for an entire column of the grid and then is solved. In Figure 2.2 a representation of the procedure can be observed, representing in red the given set of initial values at $\xi = 0$.

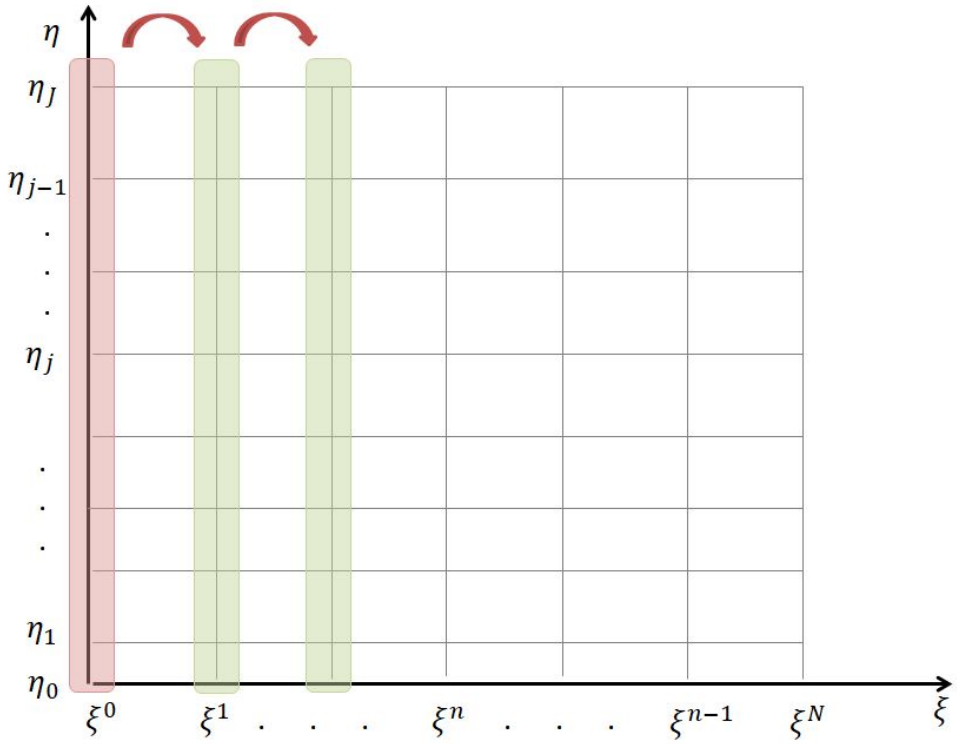



Figure 2.2: Grid representation of how the columns are solved by columns.

## 2.3. Main script

The main script is where the functions interact between them and the main loops for solving the boundary layer are done. However, before starting the computation, some initial parameters must be defined. These parameters can be grouped into: the grid defining parameters (J, N, h1, K, $\eta_e$), the flow at x=0 (which can be introduced manually or, if given some initial guesses, computed automatically), and the external velocity distribution.

After defining all the initial values and storing them in their proper site, a general loop is set from the second column of the grid to the last one. The first column is not added to the loop as represents the velocity profile at x=0.

In Figure 2.3 a general scheme of how the program works can be consulted.

The operating on the code starts once defined the incident velocity profile at $x = 0$ and saved in the first column of the values of the grid, then the loop starts. Firstly, the values from the previous column are duplicated; in this way, an initial guess for the computation is obtained. Then the corresponding functions that enable the computation of the boundary layer are called: the A and r matrices are first computed; second, the Gauss Method is applied, and third, the increments are obtained.

Once the increments are obtained it is checked whether the results are diverging or not. In order to do this verification, the square values of the second residual term are added and compared with the previously added value. If in the actual iteration the residual squared

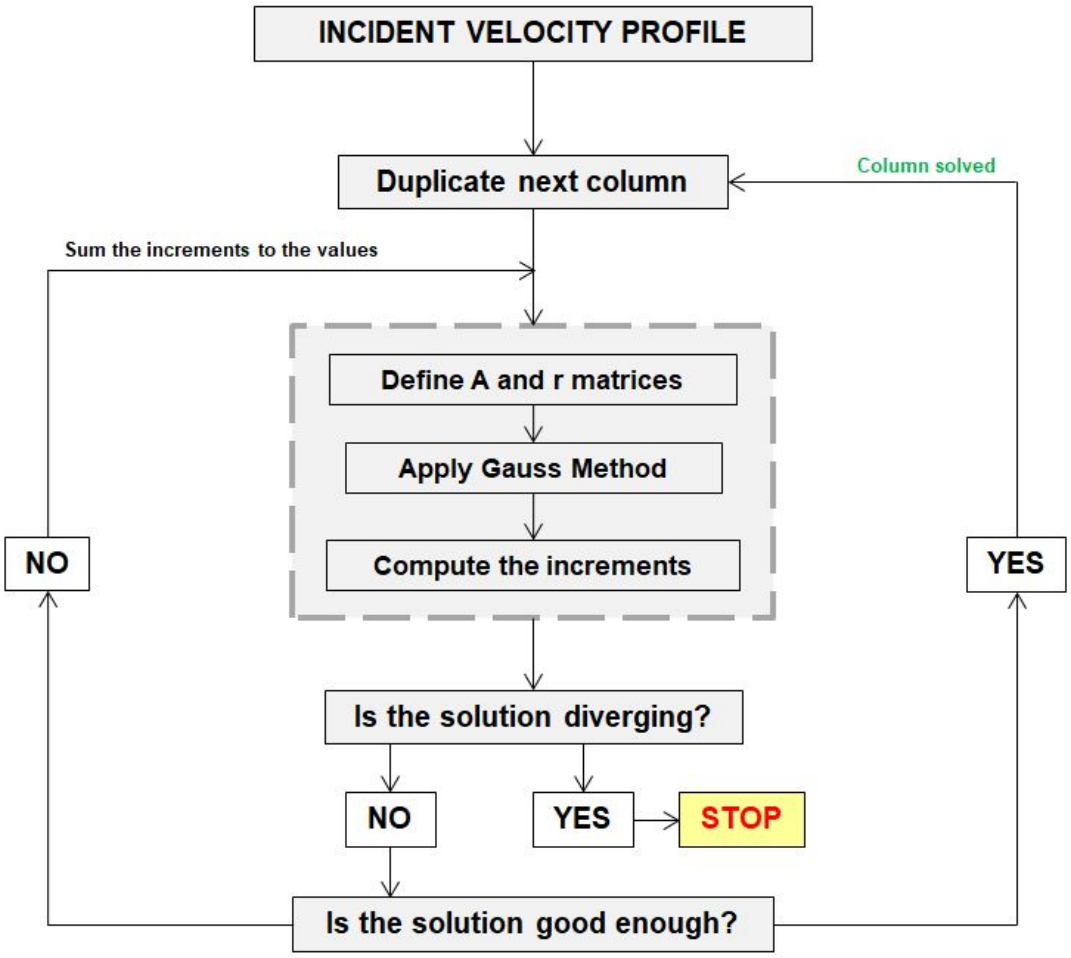



Figure 2.3: Operating diagram of the main code.

sum has increased that would mean that the code is starting to diverge, as the residual values must always get lower along the iterations. If the solution is diverging, the computation is stopped, as it is not possible to compute further points of the grid, meaning that it might reach a separation point.

If the solution is not diverging then it is checked if the solution obtained is good enough or if more iterations must be done. This verification, can be either done by establishing a tolerance value and checking the residual values or by the values of the increments. If decided that the solution is not good enough, the computed increments are added to the primitive values and the group of functions that computes the boundary layer are computed again. The procedure is repeated until the solution is valid.

Finally, when the solution is good, meaning that the column of the grid has been successfully solved, it is moved to the next column and all the steps are repeated until the entire grid is solved.

This would be a general overview of the code. However, on that basis, several functions can be added to expand the features of the program. In the next section, some of these expanding functions are explained, together with the ones mentioned in the base code.

## 2.4. Functions

To maintain the main script sorted and understandable, several functions have been defined. This approach enables the release of the main program of large loops and big blocks of code.

Some of these functions are repeatedly used along the code and others are just used in some specific points if needed. Hereafter, the most important functions of the program, albeit not all of them, will be described. Some functions are not described as they are very simple and short, and with the comments pointed out in the code it is sufficient to understand them. All the functions, together with the main script, can be consulted in Appendix C.

### 2.4.1. Create Grid function

After the initial inputs of the code have been set, this is the first function used. This function is designed so that given a set of parameters it defines all the transformed coordinates for the grid ($\eta$ and $\xi$ or x).

The inputs of the function are:

- N: denoting the number of divisions in the $\xi$ direction.
- J: denoting the number of divisions in the $\eta$ direction.
- h1: denoting the first or initial increment value in the $\eta$ direction.
- K: denoting the homogeneity of the grid, being K=1 a homogeneous grid.

First of all, the function defines an array of cells of dimensions N+1 and J+1. They are defined as +1 so that the final position can be reached. If defined instead just N, the last position would be $\xi = 1 - \frac{1}{N}$.

Then, two main loops are defined to sweep the whole cell array, defining at each position the $\xi$ and $\eta$ coordinates of that point of the grid, which are the first and second elements of that cell. Additionally, the third element of the cell stores the next increment value in $\eta$ direction, the $h_{j+1}$, as could be a parameter of interest and is going to be needed in the computation of the next grid points.

The output of the function is the entire cell array with all the positions set.

### 2.4.2. Falkner-Skan function

The next step once defined the grid is defining the velocity profile at x=0 and storing it properly. This velocity distribution can be defined as a mathematical expression, as it is going to be seen in some tests done in the following chapter. However, this velocity profile can also be computed from the self-similar Falkner-Skan solution. In this function, it will be explained how this velocity profile solution is obtained.

The required inputs are:

- J: denoting the number of divisions in the $\eta$ direction.
- h1: denoting the first increment value in the $\eta$ direction.
- K: denoting the homogeneity of the grid.
- m and Cf2Re: these values correspond to the initial guesses of the dimensionless pressure gradient and the half value of the skin friction coefficient multiplied by the local Reynolds number based on the momentum thickness. These values can be set as the ones from the Falkner-Skan self-similar solutions, which are already tabulated. For instance, for a flat plate $m = 0$ and $\frac{C_f}{2} Re_\theta = 0.22052$.
- tolerance: if the result could be obtained exactly the tolerance would be 0. However, for quicker and more efficient results a tolerance is set. A typical value used for the simulations is 1E-8.

It should be noted that this function is a modified version of an exercise of the course Aerodynamics during the degree in Aerospace Systems Engineering – EETAC, adapting the code to the parameters and the possibility of a non-homogeneous grid.

Starting with the mathematical formulation, it is worked with the simplified version of the momentum equation, applying the Falkner-Skan transformation and considering the external velocity profile as $u_e = Cx^m$, being C and m constant, and considering $f$ and $f'$ independent of x ( or $\xi$). These simplifications lead to the following expression.

$$f''' + \frac{m+1}{2} f f'' + m(1 - (f')^2) = 0 \tag{2.24}$$

Being $f$ the dimensionless stream function, $f'$ the dimensionless velocity and the term $f''$ related to shear stresses.
The initial and boundary conditions are that at $f'(0) = 0$ as the no-slip condition applies. At $f'(\infty) = 1$, the velocity re-encounters the external velocity and at $f(0) = 0$, an arbitrary condition is set.

Moreover, now it is being work with an infinite domain $\eta = [0, \infty)$. In order to work with a finite domain, new variables have to be defined $\hbar = \eta/\eta_\infty$ and $h = f/\eta_\infty$; so now having $\hbar = [0, 1]$.

However, when applying these changes, the partial derivatives must be rewritten considering:

$$\frac{\partial}{\partial \eta} = \frac{1}{\eta_\infty} \frac{\partial}{\partial \hbar} \tag{2.25}$$

And thus,

$$\begin{aligned} f &= h\eta_\infty \\ f' &= h' \\ f'' &= \frac{h''}{\eta_\infty} \\ f''' &= \frac{h'''}{\eta_\infty^2} \end{aligned} \tag{2.26}$$

Applying the transformation to equation 2.24:

$$h''' + \eta_\infty^2 \left(\frac{m+1}{2} h h'' + m(1 - h'^2)\right) = 0 \tag{2.27}$$

Hereafter, to express it as a first-order system, vectors h and H are defined.

$$h(\hbar) = \begin{bmatrix} h \\ h' \\ h'' \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} \tag{2.28}$$

$$h'(\hbar) = H(\hbar) = \begin{bmatrix} h' \\ h'' \\ h''' \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ -\eta_\infty \left(\frac{m+1}{2} h_0 h_2 + m(1 - h_1^2)\right) \end{bmatrix} \tag{2.29}$$

Solving numerically and setting the corresponding increments, explained further on, at the surface the $h(\hbar)$ matrix can be expressed as:

$$h_0 = \begin{bmatrix} 0 \\ 0 \\ h_{20} \end{bmatrix} \tag{2.30}$$

And on the limit point $\hbar = 1$:

$$h_k = \begin{bmatrix} h_{0k} \\ 1 \\ h_{2k} \end{bmatrix} \tag{2.31}$$

Representing k, in this case, the row position.

From the previous expressions, the unknown value is $h_{20}$. The other values can be obtained with the Taylor expansion:

$$h_{k+1} = h_k + h'_k \Delta\hbar \tag{2.32}$$

Then, the way to proceed will consist in setting an initial guess of $h_{20}$ and solving iteratively the set of values until a solution is found enabling that at $h_{1k} = 1$, with the given tolerance set as an input.

$$F = S(h_{20}) - 1 < 10^{-8} \tag{2.33}$$

The initial guess of the value $h_{20}$ can be obtained from the tabulated solution of Falkner-Skan. And the next values of $h_{20}$ can be computed using the secant method, finding the zero value from the secant line computed with the previous values.

$$h_{20} = \frac{F^n h_{20}^{n-1} - F^{n-1} h_{20}^n}{F^n - F^{n-1}} \tag{2.34}$$

Last but not least, the increments spacing must be adapted to match the increments of the main code.

Following all the procedures previously explained, when the function is executed, it starts creating the vector of increments, taking into account the K factor. Then the boundary conditions are defined and the main loop is started, and the equations are solved until a proper solution is found.

Once the solution is found, the corresponding scale is undone, multiplying the $f$ values by $\eta_e$ and dividing $f''$ by $\eta_e$.

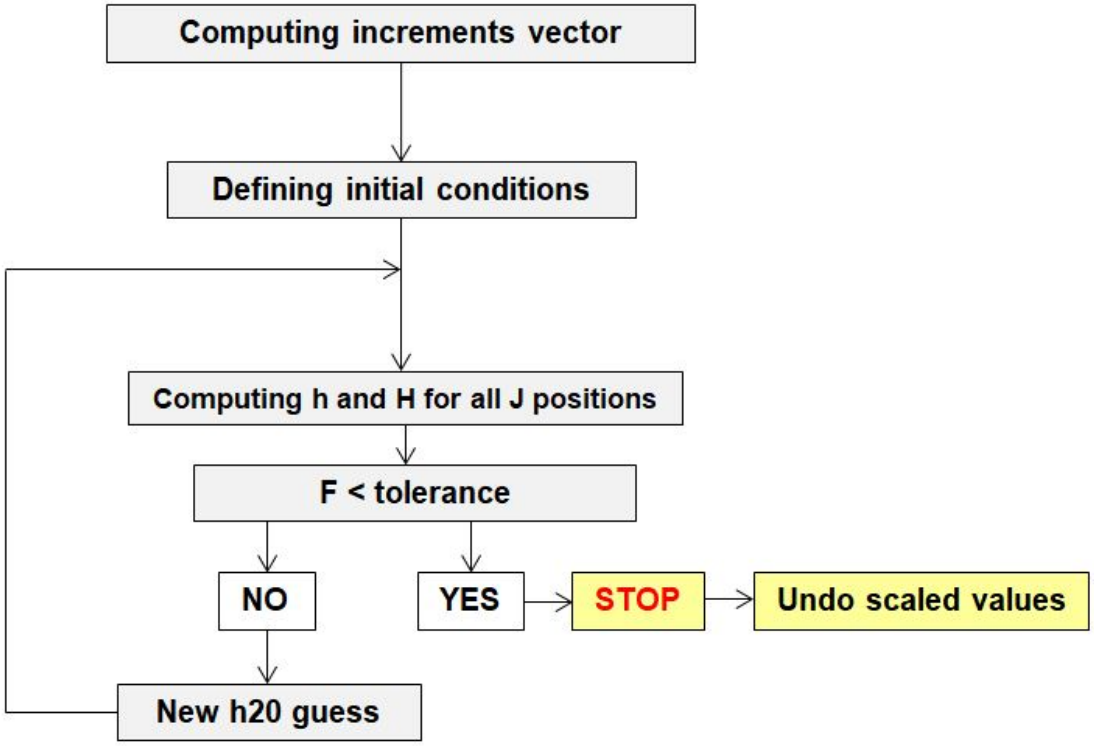



Figure 2.4: Operating diagram of the Falkner-Skan function.

The output of the function is a matrix of J+1 rows and 3 columns. For each $j$ position the $f$, $f'$ and $f''$ values are at the first, second and third column, respectively. These values are going to be the ones considered as the incident flow, stored as the first column (at position x=0) of a new cell array that would contain the values of the whole grid.

### 2.4.3. Filling A and r function

As the name indicates, this function is dedicated to filling the cell arrays A and r from the expression 2.17, and the corresponding expanded equations. This function is the first one from the solving block seen in Figure 2.3. Therefore, the outputs of the function are going to be the cell arrays A and r.

On the other hand, the inputs are:

- J: number of divisions in the $\eta$ direction.
- h1: the first increment value in the $\eta$ direction.
- n: the current horizontal iteration. It denotes the actual working column of the grid.
- Values: the cell array where all the values until the given iteration are stored. There are as many cells as grid points have been solved. In each cell position, it is specified the $f$, $f'$, $f''$ and the b term of that corresponding point.

- positions: the cell array with the ξ, η and $h_j$ values. This cell array is the output of the function Create Grid.
- $m_n$: the dimensionless pressure gradient of the actual working column (n).
- $m_{n1}$: the dimensionless pressure gradient of the previous column (n-1).

With all these parameters, the matrices that constitute the A and r matrix are computed and stored in the corresponding row and column.

First the A and r cell arrays are created. The A cell array has J+1 rows and for each row 3 cells are defined, being the $B_j, A_j$, and $C_j$ matrices of 2.19, in this respective order. The cell array $r$ would have J+1 rows and just a cell array in each row, containing the vector of residual terms $r_1, r_2$, and $r_3$.

For computing all the matrices and storing them in their proper position, a general loop sweeping all the J positions has been implemented, separating the boundary positions.

Finally, it should be underlined that for all the computations done it is used the $f$, $f'$ and $f''$ actual values in combination with other parameters of the grid. However, the third residual term has to be computed during the next $j$ iteration. Because of that, the third residual values are set as 0 at the beginning, as are unknown, and in the following iteration are overwritten with the corresponding value. This concept is explained further in the Appendix A.

### 2.4.4. Gauss Method function

The next function from the solving block is the Gauss Method function. This function consists in operating between the different equations of the system (that are expressed in a matrix form) to obtain in the last equation an expression that can be solved directly to obtain the last increment matrix. Once the last increment matrix is computed it is possible to generate all the previous increments, starting computing from the bottom to the top.

The required inputs are:

- J: number of divisions in the η direction.
- A: cell array of J+1 rows of cells and 3 columns. Each row stores $B_j, A_j$, and $C_j$ matrices. These matrices are stored in the first, second, and third columns of the A cell array, respectively.
- r: cell array of J+1 rows and a single column. Each position of the row stores a matrix with a column vector with the residual values $r_1, r_2$, and $r_3$ of each position.

This function only computes the different resulting matrices affected by operations when introducing this 0 value in the last equation. The increments values are going to be computed in another function.

The Gauss Method function, first of all, defines the new cell arrays that are going to be used: $A_{gauss}$, with J rows and 2 column dimensions, and $r_{gauss}$, with J rows and 1 column dimension.

Then a general loop is set to sweep over all the $j$ positions. In the first $j$ value, at the position in contact with the wall or surface, the first $A_{gauss}$ term is computed by multiplying each equation by $-B_j$ and by the inverse matrix of $A_{j-1}$. However, in the following iterations, instead of the inverse matrix of $A_{j-1}$, the inverse of the resulting matrix obtained in the previous row that multiplies the increment term $\delta_{j-1}$ is going to be considered. For instance, for the first row it means multiplying the expression by $-B_1 A_0^{-1}$ yet in the second row the $A_{gauss}$ term would be, in fact, $A_1 - B_1 A_0^{-1} C_0$.

Figure 2.5 depicts the procedure followed using an example of the first iterations.

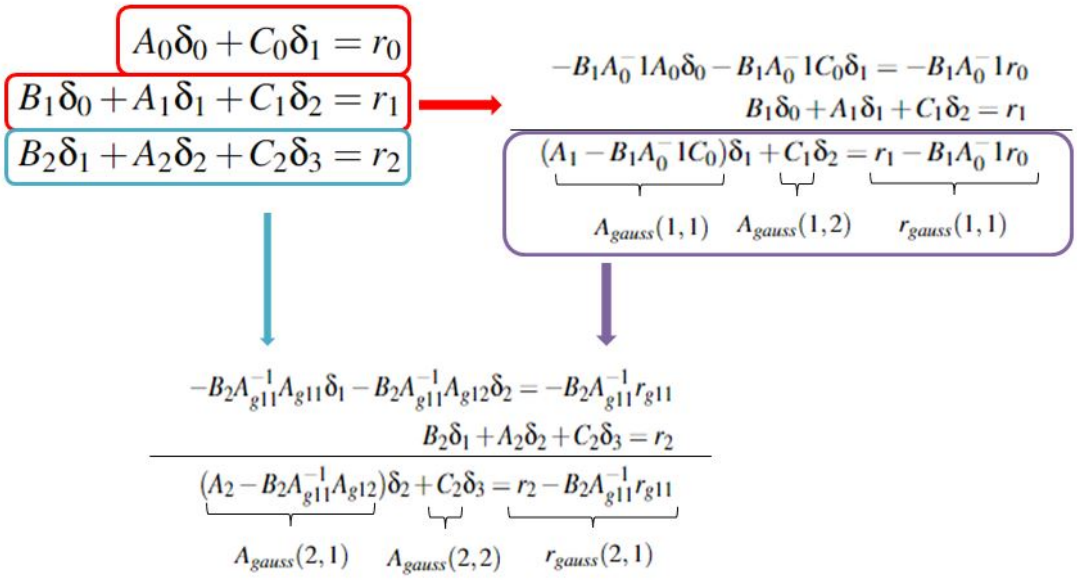



Figure 2.5: Example of how Gauss Method would apply in the first two iterations.

The output of the function are the two new cell arrays: $A_{gauss}$ and $r_{gauss}$.

### 2.4.5. Computing increments function

This function corresponds to the last one of the group for obtaining the boundary layer solution. The required inputs are:

- A gauss: cell array containing J rows and 2 columns, having at each position the resulting matrices that multiply the unknown increments. This cell array is the first output obtained from the Gauss Method function.
- r gauss: cell array with J rows and 1 column. Each position stores a matrix with the residual values of that position after applying Gauss Method. This cell array is the second output obtained from the Gauss Method function.
- A: cell array of J+1 rows of cells and 3 columns. Each position stores $B_j, A_j$, and $C_j$ matrices of the corresponding row and column grid position.

- r: cell array of J+1 rows and a single column. Each position of the row stores a matrix with a column vector with the residual values $r_1, r_2$, and $r_3$ of each position.

As expressed during the Gauss Method function, the increments have to be solved from the bottom to the top, that is why the new cell array containing the increments starts the filling from the last position, as it can be directly computed by isolating the increment matrix.

Then the above increments can be solved as they depend on the next increment (the j+1), which is already computed.

The output of the function is the new cell array named increments. Increments have in each position a matrix with the computed values of the increments of $f$, $f\prime$ and $f\prime\prime$.

### 2.4.6. Import m vector

This function is used to define the external outer streamwise velocity distribution. This velocity distribution can be set manually, by a mathematical expression, or imported from a text file. Specifically, this function imports the data from a text file to obtain the needed values, having as a reference Xfoil structure saved files.

The input of the function is just one:

- File name: name of the text file where has been stored the Xfoil data. This file has to be saved in the same folder where is the code and the structure of this document must be similar to the one shown in Figure 2.6.

```
#    s        x        y      Ue/Vinf    Dstar     Theta      Cf        H
   0.00000  1.00000  0.00126  1.02706  0.022513  0.003203 -0.000231     7.029
   0.00191  0.99811  0.00153  1.02718  0.022512  0.003200 -0.000345     7.036
   0.00507  0.99498  0.00196  1.02736  0.022433  0.003195 -0.000346     7.021
   0.00866  0.99142  0.00246  1.02756  0.022329  0.003190 -0.000347     7.000
   0.01279  0.98733  0.00302  1.02779  0.022199  0.003184 -0.000348     6.971
   0.01762  0.98254  0.00368  1.02806  0.022036  0.003178 -0.000349     6.934
   0.02338  0.97684  0.00446  1.02837  0.021828  0.003170 -0.000351     6.886
   0.03037  0.96991  0.00540  1.02876  0.021563  0.003161 -0.000353     6.822
   0.03900  0.96136  0.00656  1.02925  0.021218  0.003149 -0.000356     6.738
   0.04974  0.95072  0.00797  1.02987  0.020773  0.003134 -0.000360     6.628
   0.06291  0.93765  0.00968  1.03066  0.020208  0.003116 -0.000366     6.485
   0.07845  0.92224  0.01167  1.03164  0.019523  0.003094 -0.000372     6.310
   0.09578  0.90505  0.01385  1.03281  0.018750  0.003068 -0.000381     6.110
   0.11418  0.88679  0.01611  1.03415  0.017929  0.003040 -0.000390     5.897
   0.13313  0.86798  0.01839  1.03565  0.017088  0.003009 -0.000400     5.678
   0.15232  0.84892  0.02065  1.03733  0.016255  0.002976 -0.000407     5.462
   0.17162  0.82975  0.02288  1.03918  0.015436  0.002941 -0.000411     5.248
   0.19097  0.81052  0.02506  1.04121  0.014641  0.002904 -0.000411     5.042
   0.21034  0.79127  0.02719  1.04345  0.013877  0.002865 -0.000404     4.844
   0.22972  0.77201  0.02928  1.04591  0.013143  0.002823 -0.000384     4.656
   0.24910  0.75273  0.03132  1.04859  0.012443  0.002779 -0.000351     4.477
   0.26848  0.73346  0.03331  1.05149  0.011778  0.002733 -0.000300     4.309
   0.28785  0.71418  0.03525  1.05458  0.011149  0.002686 -0.000231     4.151
   0.30721  0.69491  0.03713  1.05788  0.010557  0.002637 -0.000142     4.003
   0.32657  0.67564  0.03897  1.06141  0.010002  0.002586 -0.000034     3.867
   0.34592  0.65638  0.04075  1.06517  0.009479  0.002534  0.000094     3.741
   0.36525  0.63712  0.04247  1.06916  0.008995  0.002479  0.000237     3.629
   0.38457  0.61787  0.04414  1.07336  0.008543  0.002423  0.000395     3.526
```

Figure 2.6: Example of a fragment Xfoil file.

The function begins with importing the data from the specified file and storing the columns

of interest: the first column, representing the arclength along the airfoil surface, and the fourth column, representing the external velocity distribution.

The first computation done is finding the stagnation point, at which the velocity would be 0 or as close to 0 as possible. This point is going to be used as the separation point between the upper surface and the lower surface regions.

The upper surface velocities would be the flipped vector going from 1 to the found value representing the stagnation point. The arclength of the airfoil positions would then be the flipped vector of the result of subtracting the arclength value from the arclength value at the stagnation point. Finally, the dimensionless pressure gradient can be computed with backward differences knowing the external velocity distribution and the corresponding positions.

For the lower surface the vectors and computations are done as in the upper surface case yet the working region is from the stagnation point to the end, excluding the wake values.

The outputs of the function are all the vectors defined (the dimensionless pressure gradient, the positions of arclength of the airfoil, and the external velocity) for both the lower surface and the upper surface.

### 2.4.7. Turbulence function

This function computes the eddy viscosity terms ($\varepsilon_m^+$) that must be added to $b$, following the turbulence model.

The required inputs of the function are:

- viscosity: viscosity of the fluid.
- Values: cell array where all the values until the given iteration are stored; having in each position $f$, $f'$, $f''$ and the $b$ term of that corresponding point (set by default as 1).
- positions: cell array with the $\xi$, $\eta$ and $h_j$ values.
- n: current column of the grid that is being solved.
- m: current dimensionless pressure gradient.
- Xtr: x position where it is expected to occur the transition from laminar to turbulent flow.
- Ue: free stream velocity of that column.
- Ue at Xtr: free stream velocity at the transition point.
- Momentum: the computed value of the momentum thickness until the current column.
- alpha and u0: some additional parameters that may be needed as a part of an integral computation of the free stream velocity.

Once defined all the required inputs, a general loop is defined sweeping all the $j$ positions and defining additional needed values and parameters. And once the eddy viscosity term is obtained, it is added to the term $b$ and the cell array of Values is updated, which is the output of the function.

Hereunder, the inner and outer layer formulation procedure will be presented. As commented before, the eddy-viscosity formulation of Cebeci and Smith [7] distinguishes between the inner and outer regions, modeling each of them differently.

#### *2.4.7.1. Inner region*

For the inner region the eddy viscosity ($\varepsilon_m$) is computed as follows.

$$\varepsilon_m = l^2 \left| \frac{\partial u}{\partial y} \right| \gamma_{tr} \tag{2.35}$$

Being l the mixing length, k=0.4 and A the damping-length constant.

$$l = ky(1 - exp(-\frac{y}{A})), \quad A = 26\frac{\nu}{N}u_\tau^{-1}, \quad N = (1 - 11.8p^+)^{1/2}, \quad p^+ = \frac{\nu u_e}{u_\tau^3}\frac{du_e}{dx} \tag{2.36}$$

However, the Falkner-Skan transformation needs to be apllyed to work with the proper transformed coordinates. Taking into account the following expressions:

$$\eta = y\sqrt{\frac{u_e}{\nu x}} \qquad \frac{\partial u}{\partial y} = \frac{1}{g}\frac{\partial}{\partial \eta}(u_e f') = \frac{1}{g}u_e f'' \tag{2.37}$$

Then, the inner region expression can be rewritten as:

$$\varepsilon_m^+ = \frac{\varepsilon_m}{\nu} = \frac{0.16}{\nu}\eta^2\frac{\nu x}{u_e}(1 - exp(-y/A))^2\sqrt{\frac{u_e}{\nu x}}f'' u_e \gamma_{tr} \tag{2.38}$$

Or equivalently, using the Reynolds number:

$$\varepsilon_m^+ = 0.16 Re_x^{1/2}\eta^2[1 - exp(-y/A)]^2 f''\gamma_{tr} \tag{2.39}$$

Where the y/A ratio can be computed as:

$$y = \eta\sqrt{\frac{\nu x}{u_e}}, \quad A = 26\frac{\nu}{N}u_\tau^{-1} \quad Being \quad \frac{y}{A} = \eta\sqrt{\frac{\nu x}{u_e}}\frac{N}{26\nu}u_\tau$$

$$With \quad u_\tau = \sqrt{\frac{\tau_w}{\rho}} = \sqrt{\frac{\mu(\frac{\partial u}{\partial y})_w}{\rho}}, \quad (\frac{\partial u}{\partial y})_w = \frac{1}{g}\frac{\partial}{\partial \eta}(u)_w = \sqrt{\frac{u_e}{\nu x}}u_e f''_w$$

Resulting in the following expression:

$$y/A = \eta\frac{N}{26}Re_x^{1/4} f''^{1/2}_w \tag{2.40}$$

Additionally, $p^+$ is going to be also affected by this transformation of variables. For the following transformation the dimensionless pressure gradient $m = \frac{x}{u_e}\frac{du_e}{dx}$ is going to be used.

$$p^+ = \frac{\nu u_e}{u_\tau^3}\frac{du_e}{dx} = \frac{\nu u_e}{u_\tau^3}\frac{mu_e}{x} = \frac{\nu u_e^2 m}{x}\left[\sqrt{\frac{\mu}{\rho}\sqrt{\frac{u_e}{\nu x}}u_e f''_w}\right]^{-3}$$

$$p^+ = m f''_w{}^{-3/2} Re_x^{-1/4} \tag{2.41}$$

Finally, the last term $\gamma_{tr}$ (an intermittency value) will be also used for the outer region. It is expressed as:

$$\gamma_{tr} = 1 - exp\left[-G(x - x_{tr})\int_{x_{tr}}^{x}\frac{dx}{u_e}\right] \tag{2.42}$$

$$With \quad G = \frac{3}{C^2}\frac{u_e^3}{\nu^2}Rx_{tr}^{-1.34}, \quad C^2 = 213(logRx_{tr} - 4.7323) \tag{2.43}$$

Denoting $x_{tr}$ the position at which transition occurs and $R_{xtr}$ the Reynolds value associated.

#### *2.4.7.2. Outer region*

For the outer region the eddy viscosity can be computed as:

$$\varepsilon_m = \alpha u_e \delta^* \gamma_{tr} \gamma \tag{2.44}$$

Where $\delta^*$ is the displacement thickness which has already been described in previous chapters.

$$\delta^* = \frac{x}{Re_x^{1/2}}(\eta_e - f_e) \tag{2.45}$$

For simplicity, $\alpha = 0.0168$ is considered to mean that there is no strong pressure gradient effect; obtaining the final expression of the eddy viscosity.

$$\varepsilon_m^+ = 0.0168 Re^{1/2}(\eta_e - f_e)\gamma_{tr}\gamma \tag{2.46}$$

The $\gamma_{tr}$ term is computed as in the inner region yet the $\gamma$, which is an intermittency parameter, is computed as:

$$\gamma = \frac{1}{2}(1 - erf\frac{y - Y}{\sqrt{2}\sigma}) \tag{2.47}$$

Where the unknown values Y and $\sigma$ (general intermittency parameters) can be obtained from the Figure 2.7 of experimental data of H. Fiedler and M. R. Head [20], considering normal development of the boundary layer.

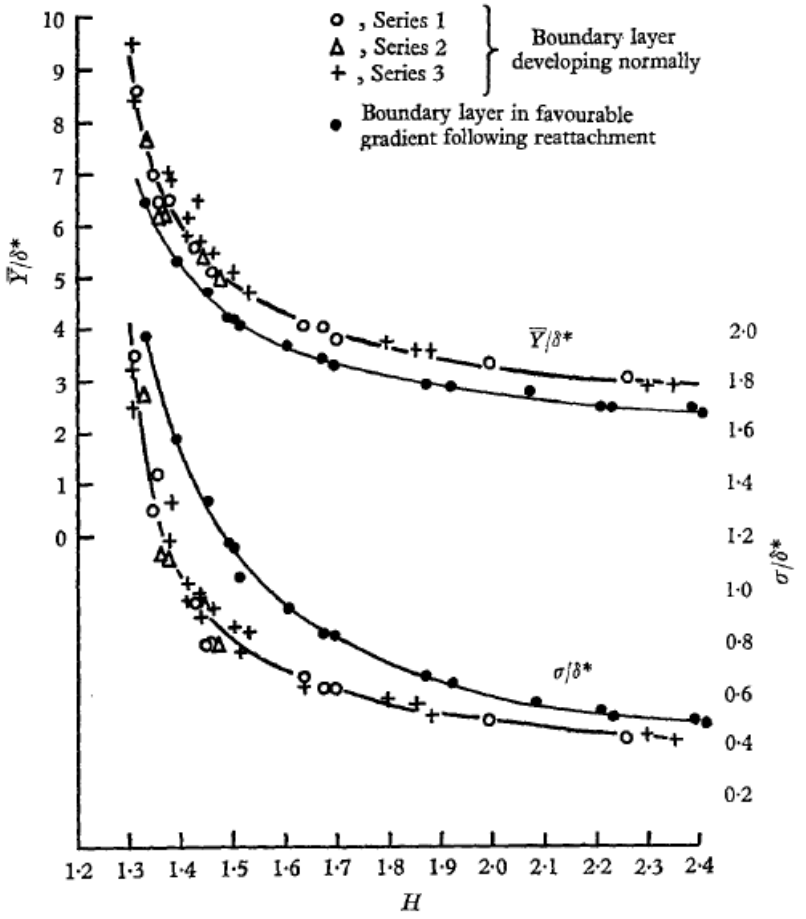



FIGURE 8. $\overline{Y}/\delta^*$ and $\sigma/\delta^*$ plotted against $H$.

Figure 2.7: Graphics of $Y/\delta^*$ and $\sigma/\delta^*$ against H. [20]

The proceeding way to determine the $\gamma$ is by computing the displacement thickness and the momentum thickness in the actual position, thus the H is obtained. Once the H is known, the Y and $\sigma$ ratios for the displacement thickness can be extracted from the graphics. As the displacement thickness is already computed the Y and $\sigma$ values can easily be obtained.

Finally, to obtain good enough values from the graphic, Engauge Digitizer has been used. This open-source software enables obtaining the points of a given graph and because of that, it facilitates the task of introducing in the code these values. From the Engauge Digitizer lists of points can be obtained for each curve. These lists can be introduced in the code and approximated by a function that when introduced an H value returns the intermittency parameters Y and $\sigma$.

### 2.4.8. Compute displacement thickness function

The displacement thickness can be computed as:

$$\delta^* = \int_0^{\delta} (1 - \frac{u}{u_e}) dy \tag{2.48}$$

Applying the transformation of variables $\eta\sqrt{\frac{\nu x}{u_e}} = y$ and thus $d\eta\sqrt{\frac{\nu x}{u_e}} = dy$ and $u/u_e = f'$ the following expression is deduced.

$$\delta^* = \int_0^{\eta_e}(1-f')d\eta\sqrt{\frac{\nu x}{u_e}} = \sqrt{\frac{\nu x}{u_e}}(\int_0^{\eta_e} d\eta - \int_0^{\eta_e} f'd\eta) = \sqrt{\frac{\nu x^2}{u_e x}}(\eta_e - f(x,\eta_e))$$

$$\delta^* = \frac{x}{\sqrt{Re_x}}(\eta_e - f(x,\eta_e)) \tag{2.49}$$

The main aim of this function is computing the equation 2.49 for the indicated range of $\eta$ values introduced. The output of the function is a vector of the local values of the displacement thickness at the different positions.

### 2.4.9. Compute momentum thickness function

The momentum thickness can be computed as:

$$\theta = \int_0^{\delta} \frac{u}{u_e}(1-\frac{u}{u_e})dy \tag{2.50}$$

Applying the transformation of variables the following expression is deduced.

$$\theta = \frac{x}{\sqrt{Re_x}}\int_0^{\eta_e} f'(1-f')d\eta = \frac{x}{\sqrt{Re_x}}(\int_0^{\eta_e} f'd\eta - \int_0^{\eta_e}(f')^2 d\eta)$$

$$\theta = \frac{x}{\sqrt{Re_x}}(f(x,\eta_e) - \int_0^{\eta_e}(f')^2 d\eta) \tag{2.51}$$

The first integral term can be solved directly as $f(x,\eta_e)$ yet the second term needs to be solved, for instance, by applying the Riemann sum (approximating the integral by a finite sum of the local values multiplied by the steps increments).

Following this equation, this function computes the momentum thickness of the indicated interval and outputs a vector with each local value of the momentum.

# CHAPTER 3. TESTING THE CODE

Thus the previous chapter presented the function of the code and the expected inputs and outputs. This third chapter deals exclusively with the testing of the functioning of the code to validate its proper operation.

The initial test is going to be executed with simple cases: maintaining a flat plate with a laminar flow, incorporating a stagnation point, comparing results with the integral method formulation, and analyzing the evolution of the separation point.

Additionally, once checked the proper running for these cases,the effects of incorporating turbulence, modifying the upstream boundary condition velocity profile, or incorporating airfoils will be tested.

## 3.1. Flat plate case

The initial validation of the results is done for the specific case of having a flat plate at zero angle of incidence, comparing it with the Blasius boundary layer solution (stationary, two-dimensional and laminar boundary layer along an infinite flat plate parallel to a constant one-way flux).

Because of setting a constant free stream velocity, the derivative of velocity is zero and thus the dimensionless pressure gradient (m=0).

The expected solution from the Blasius [18] is:

$$c_f = \frac{0.664}{Re_x^{1/2}} \qquad \frac{\delta^*}{x} = \frac{1.721}{Re_x^{1/2}} \qquad \frac{\theta}{x} = \frac{0.664}{Re_x^{1/2}} \tag{3.1}$$

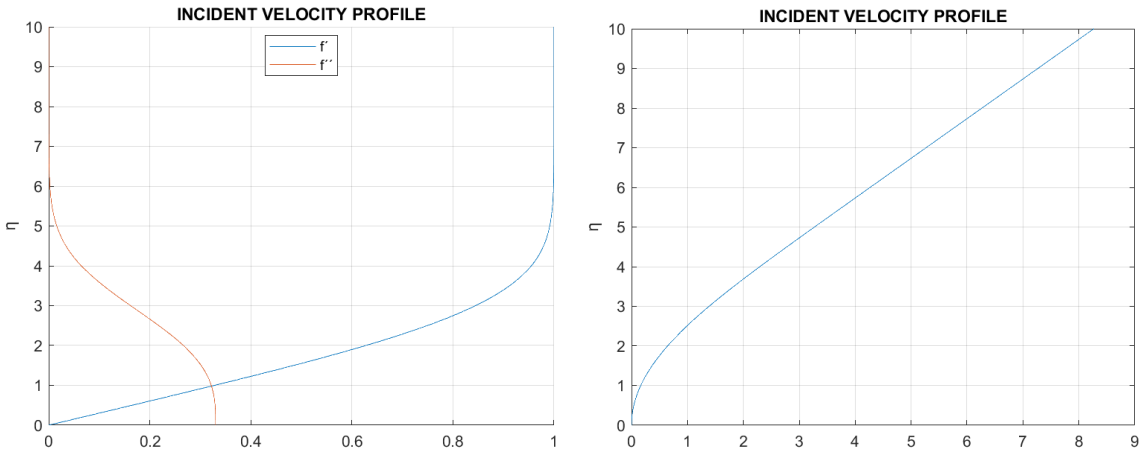
For these computations, the local Reynolds number (equation 3.2) is used being $x$ the horizontal coordinate axis set at the origin of the boundary layer and $u_e$ the outer flow velocity at that $x$ coordinate.

$$Re_x = \frac{u_e x}{\nu} \tag{3.2}$$

For simulating this scenario, the code has been configured initially with a homogenous grid (K=1) taking 1000 divisions in the $\eta$ direction and 200 in the $x$ direction (J=1000, N=200). The first grid increment $h1$ is defined as 0.01, the one-way velocity as 5 m/s and the viscosity as 1.2245E-05 kg/(m·s). However, as being a homogenous grid all the increments will have the same value. The velocity distribution is obtained with the Falkner-Skan code, setting the tolerance at 1E-8, m= 0 and Cf2Re= 0.33206. The tolerance defined inside the code to determine if the solution is enough good or not is also defined at 1E-8.

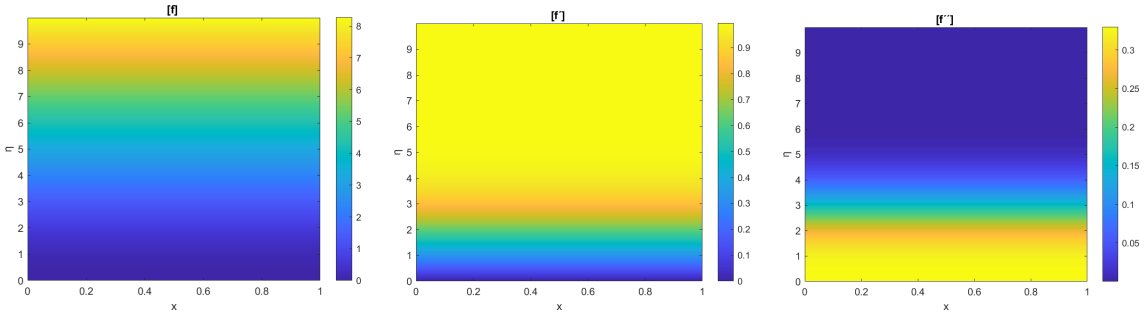
The velocity profile at $x = 0$ is the output from the Falkner-Skan function, being the initial input guesses of the function the tabulated values for a flat plate case. Because of that, the velocity profile at $x = 0$ is directly the solution, meaning that this solution should be maintained along the x-direction.

Figure 3.1 displays the velocity profile at x=0 and Figure 3.2 shows the solutions of the $f$, $f'$ and $f''$ along the x-direction. As expected, the values are maintained.




(a) $f'$ and $f''$ values (b) $f$ values

Figure 3.1: Velocity profile at x=0 of a flat plate.



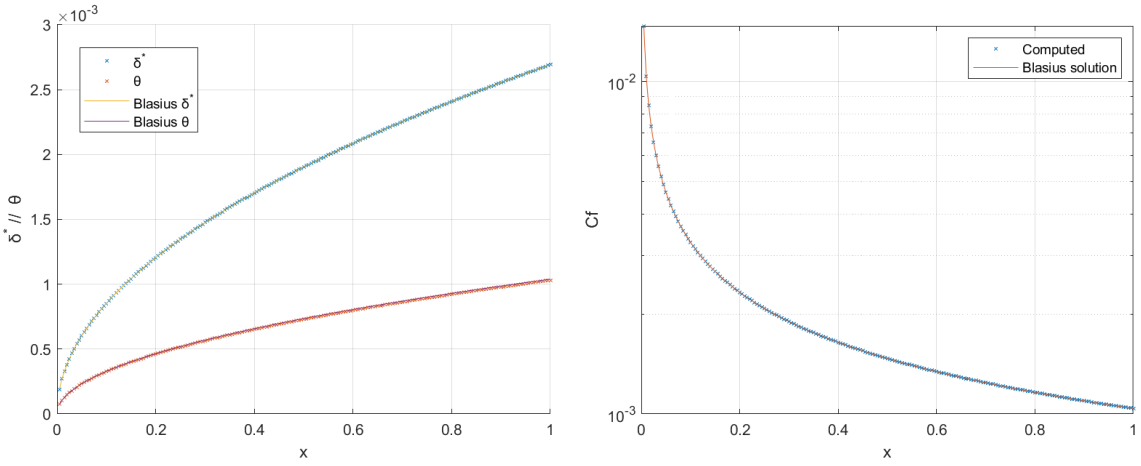

(a) Dimensionless stream function $f$ distribution along the grid positions

(b) Dimensionless velocity $f'$ distribution along the grid positions

(c) $f''$ values distribution along the grid positions

Figure 3.2: $f$, $f'$ and $f''$ distribution results for a flat plate.

If comparing with the expected tabulated results from equations 3.1, as seen in Figure 3.3, it is observed that the results fit completely the Blasius solution, in many cases even fully overlapping the plots, as in the cases of the skin friction coefficient and the displacement thickness.



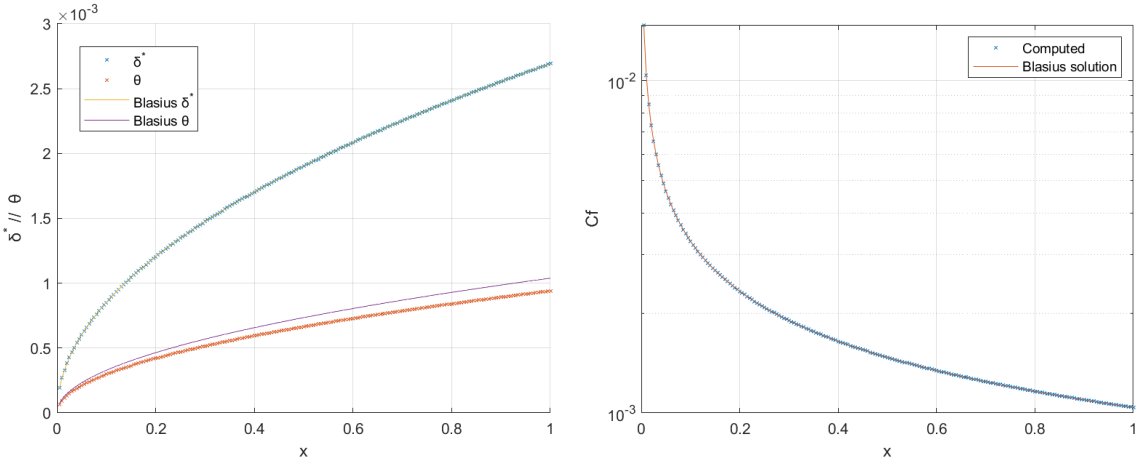

(a) Displacement and momentum thickness. (b) Skin friction coefficient.

Figure 3.3: Characteristic properties of the flat plate results in a homogeneous grid.

Besides, some simulations have also been executed with non-homogenous grids.

If establishing a non-homogeneous grid (K=1.05 and $\eta_e = 10$ ) the number of increments in the $\eta$ direction is reduced to just J=81. As can be seen in Figure 3.4, even reducing the increments from 1000 to just 81 the results continue to be accurate since a homogeneous grid is not considered.

The only parameter that diverges is the momentum thickness computation. This is because its computation includes an integral that can not be solved analytically and it is computed applying Riemann sum. The use of this procedure can lead to not having as precise results as if using a more precise method. Additionally, when computing with a non-homogenous grid, at the less dense parts of the grids, the same value for a larger step is assumed, which may also induce some deviation of the parameters.



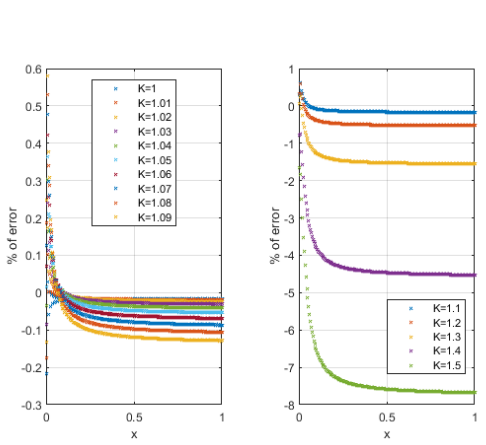

(a) Displacement and momentum thickness (b) Skin friction coefficient

Figure 3.4: Characteristic properties of the flat plate results in a non-homogeneous grid

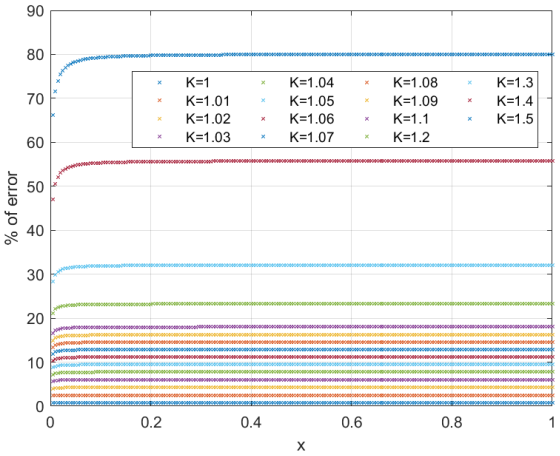
As seen, when incorporating a non-homogenous grid some parameters start deviating from the theoretical solution. Then, two cases will be studied: keeping $\eta_e$ constant and varying $K$ value and keeping the $K$ value and modifying the $\eta_e$. In this second case, the

mesh is the same, just with more points added at the upper boundary layer.

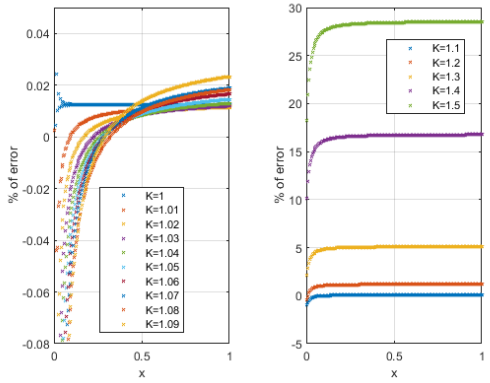
For the first case, with the same configuration of parameters as before, fixing an $\eta_e$=10 and modifying the K value from 1 to 1.5, the following percentage of errors of the characteristic properties are found:




(a) Skin friction coefficient error




(b) Momentum thickness error




(c) Displacement thickness error

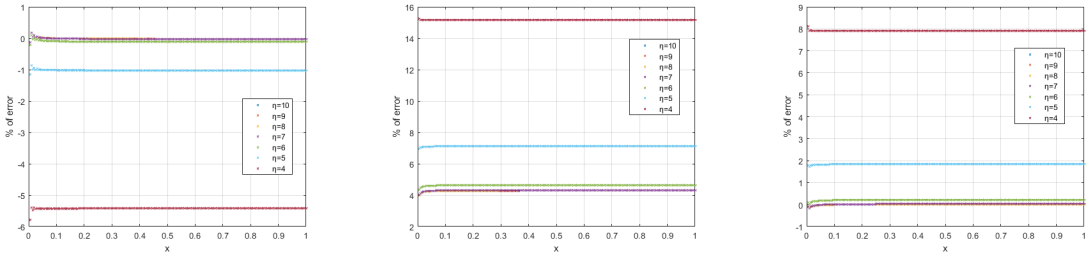
Figure 3.5: Computed errors of the characteristic properties when modifying the K value

As observed, when increasing the K parameter (meaning that fewer points are added at the upper region of the grid), the errors increase. For the skin friction term and displacement thickness, it can be seen that when moving to K higher than 1.09 the errors highly increase. If analyzing the momentum thickness, these increases are even bigger. Having for big K values excessive errors.

On the other hand, setting a constant value of $h_1$ and K, for instance, K=1.02 (which offers a pretty good accuracy) and varying the $\eta_e$ value, the results shown in the following figures can be obtained.

As can be seen, if increasing the $\eta_e$ value the solution gets slightly better, but this improvement is so small that can be even neglected. This small improvement is caused as if setting bigger $\eta_e$ values, the dense grid is going to be focus on the first $\eta$ values. For instance, when having an $\eta_e = 10$ the denser part of the grid may focus between 0-5, yet when changing to $\eta_e = 20$ the denser part is expanded to 0-10, covering more accurately

the boundary layer region (these $\eta_e$ example values are indicative just for explaining the concept, no specific ranges have been computed).




(a) Skin friction coefficient error (b) Momentum thickness error (c) Displacement thickness error

Figure 3.6: Computed errors of the characteristic properties when modifying the $\eta_e$ value

If decreasing the $\eta_e$ the accuracy is worsen, as the high-density part of the grid may not fully cover the totality of the boundary layer thickness. When $\eta_e$ is even smaller than the boundary layer thickness, the induced error is even greater.

Finally, if instead of taking as a constant K=1.02 a higher K value is considered, these results are even worse.

## 3.2. Introduction of a stagnation point

A stagnation point is a point at the body surface where the velocity of the fluid is zero. Because of that, this point is where the static pressure is going to be highest (Bernoulli equation).

For a two-dimensional stagnation point, the potential theory shows that the outer velocity grows linearly as $u_e = kx$, being k a constant value, the dimensionless pressure gradient (m) is 1 and the following results should be expected [18]:

$$\delta_1 = 0.6479\sqrt{\nu/k} \qquad \theta = 0.2923\sqrt{\nu/k} \tag{3.3}$$

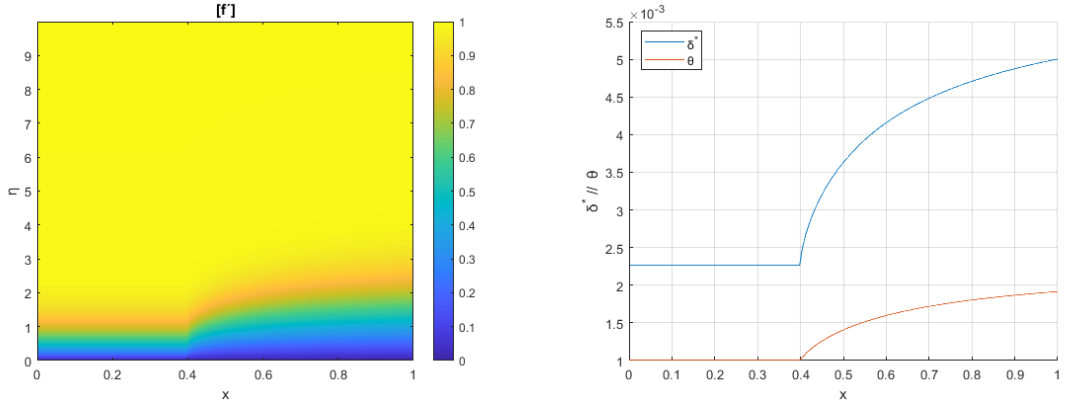
Setting the same values for the code and for equation 3.3 (k=1, viscosity 1.2245e-05 and density 1.225) and initializing the code with N=500, J=1000 and K=1.0, the results obtained are shown in the following table. The solution offers good accuracy and this can even be improved if a denser grid is defined.

Table 3.1: Dispalcement and momentum thickness results for a stagnation point

| Values | Momentum thickness | Displacement thickness |
|---|---|---|
| Tabulated values | 0.0022671777 | 0.001 |
| Code values | 0.0022671894 | 0.001 |

When combining the flat plate case with the stagnation point case, starting with stagnation points and then changing to flat plate conditions when reaching x=0.4, it should be observed that the laminar boundary layer growing in the vicinity of a stagnation point must

remain with non-null thickness, finite and constant. For doing this, a unitary dimensionless pressure gradient (m=1) is first used. When x=0.4 is reached, m=0 is forced for all the remaining x-positions.




(a) Color plot of the dimensionless velocity

(b) Displacement and momentum thickness

Figure 3.7: Evolution of a stagnation point changed to a flat plate x=0.4

In the color plot, it can be observed that the dimensionless pressure gradient changes from 1 to 0. The dimensionless velocity stops being constant and re-adapts to the new parameters. Besides, looking at the displacement and momentum thickness, it is observed that whenever a stagnation point is defined the property value remains constant and it only starts increasing when m=0 is defined at x=0.4.

## 3.3. Modifying the velocity profile

Another interesting parameter to change is the upstream boundary condition velocity profile. The code is prepared for setting the desired velocity profile, meaning that it should be able to compute a solution whenever a realistic profile is introduced. For the major of cases, the velocity profile at x=0 is set by the resolution of the Falkner-Skan equation yet in this section a hyperbolic tangent is considered.

The hyperbolic tangent is a function with values included in a range from -1 to 1. Because of this, the hyperbolic tangent is used as the $f'$ values from the velocity profile at x=0, but only working with the positive values, fixing the range from 0 to 1.

$$f' = \tanh k\eta \tag{3.4}$$

The $k$ parameter defines whether the hyperbolic tangent profile approaches a heaviside step function. This parameter is very sensitive and its ranges must be studied.

For computing the $f$ and $f''$ the backward difference is used, which for a general function $f(x)$ states that:

$$\frac{df(x)}{dx} = \frac{f(x) - f(x-h)}{h} \tag{3.5}$$

Because of that, it can be approximated that:

$$f'' = \frac{f'_j - f'_{j-1}}{h_j}$$
$$f = f_{j-1} + h_j f'_j \tag{3.6}$$

The other term that has to be computed is the k value. This parameter determines how long it takes the function to reach 1. However, these values cannot be chosen freely, as it may cause the introduction of not valid profiles.

In order to ensure that the profile is valid for computations the range of k is defined. The first factor that is checked is the different Falkner-Skan profiles for different constant values of the pressure-gradient parameter "m" (the Falkner-Skan solutions).

Figure 3.8 displays the different velocities profiles at x=0 from the one corresponding to a stagnation point (m=1) to the closest profile to the separation that can be computed (m=-0.08844).

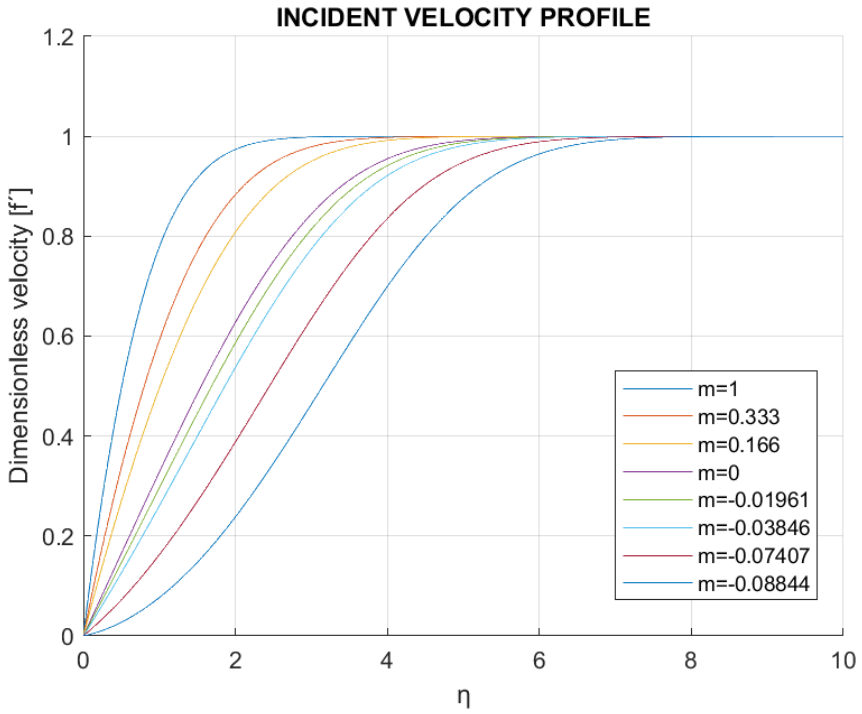
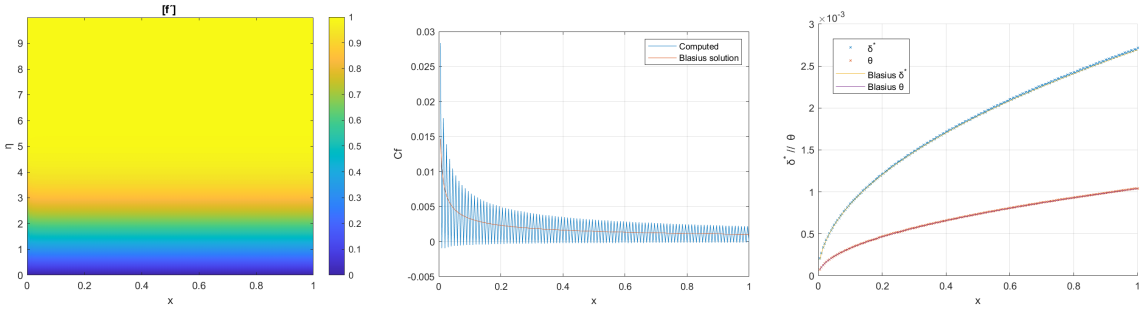



Figure 3.8: Falkner-Skan solutions of the velocity profile for different values of m, for N= 500, J= 1000, K=1, h1= 0.01 and tolerance=1E-8.

When analysed these values, the velocity profile exactly reaches $f'$=1 for a stagnation point at $\eta = 3.94$ and for the point closest to separation at $\eta = 9.07$. A flat plate configuration at a point between this range results specifically at $\eta = 7.15$.
If solving the equation 3.4 for the $\eta$ values found, it is obtained that the range of k is [0.30, 0.68], being 0.3 the point close to separation, 0.68 a stagnation point and 0.38 flat plate. These ranges could be lightly expanded by considering the 99.9% of the outer velocity.

Simulating different cases, as in the previous sections, but now with the hyperbolic tangent, the following figures are generated.

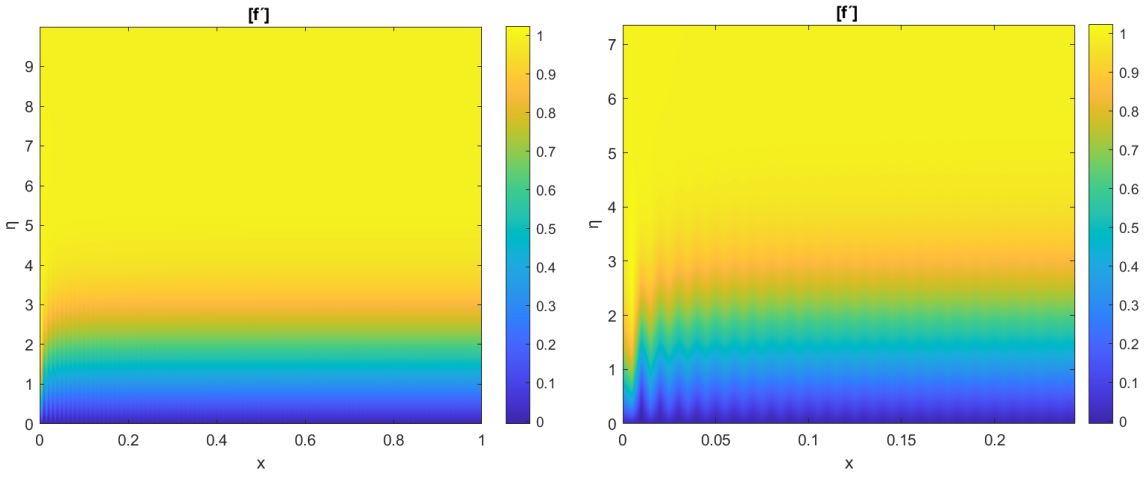



(a) Color plot of the dimensionless velocity.

(b) Computed and theoretical skin friction coefficient.

(c) Computed and theoretical displacement and momentum thickness.

Figure 3.9: Flat plate results obtained from considering an hyperbolic tangent profile as the upstream boundary condition, with K=1, N=200, J=1000 and h1=0.01.




(a) Color plot of the dimensionless velocity.

(b) Zoomed color plot of the dimensionless velocity.

Figure 3.10: Stagnation point at x=0 results obtained from considering an hyperbolic tangent profile as the upstream boundary condition, with K=1, N=200, J=1000 and h1=0.01.

In the previous graphics, an initial case of considering the upstream boundary condition of a hyperbolic tangent with a typical k value corresponding to a flat plate configuration is studied. Additionally, a flat plate development is also studied, starting with a k value corresponding to a stagnation point.

There are some variations since they are set as a mathematical expression, and may not fully fulfill a real velocity distribution.

Finally, note that the different color plots may seem perfect solutions, yet if zoomed some irregularities of the flow can be distinguished. For instance, for the flat plat case, there is a slightly bigger peak of the dimensionless velocity at the beginning, and the stagnation point case shows, if zoomed, an oscillating variation of the velocity. These oscillations are also found in the skin friction coefficient computations (for both cases).

Regarless the strong oscillations, if the solution is filtered on average, the results are generally acceptable and correlate with the theoretical expected ones.

By making these variations of the velocity profile it can be shown that the code is adapted to work independently of the upstream boundary condition velocity profile, provided that it fulfills the boundary conditions and represents a real behavior of a fluid.

## 3.4. Re-compression case

By simulating the effects of incorporating the presence of an external flow, characteristic of re-compression, the code can be adapted and compared to the simplified integral method.

From the integral method the separation point can be deduced. This subsection analyses the different separation points obtained from the code.

The resolution done with the integral method was submitted as an exercise of the course Aerodynamics during the degree in Aerospace Systems Engineering – EETAC. This resolution can be consulted in Appendix B.

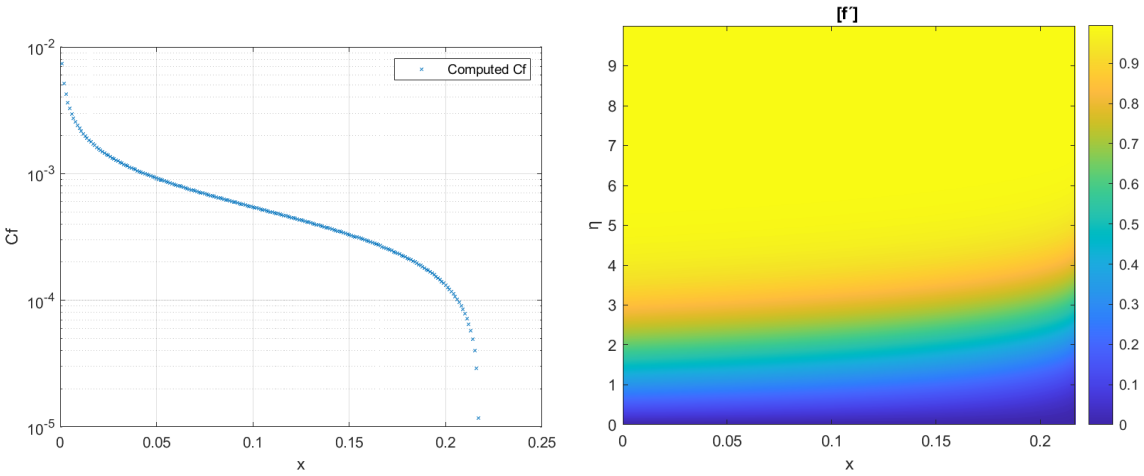
The velocity distribution representing the re-compression is:

$$u_e(x) = U_0(1 - \frac{x}{L})^{\alpha} \tag{3.7}$$

When incorporating this new profile in the code, the pressure gradient is going to be greatly affected as it is computed as the derivative of the velocity profile. As the code is not able to compute further positions from the separation, the last column computed would represent the last point before separation. When the separation point is overcome and the code continues computing, the solutions diverge, causing a never-ending loop. In order to avoid this case, the squared added values of the second residual term are analyzed iteration by iteration to ensure that the code is not diverging.

For an initial run of the code, an $\alpha$=0.5 has been considered. From the integral method, considering a laminar case with the separation condition $C_{lam} = 0.068$, the separation point is computed to be at x/L=0.169.

Moving to the code, the x-direction has been separated into 1000 equally spaced points. Meaning that, since unitary longitude is used, the increments in the x directions and thus the error range in the computation of the separation point is 0.001. From that, it can be said that the separation point must be, approximately, 0.218. For instance, if computing a bigger grid with 3000 separations, the separation point is at 0.217667, yet the computing time increases significantly. Because of that, 1000 separations are considered good enough for this case, as it is just intended to verify the correct functioning.




(a) Skin friction coefficient plot (b) Colour plot of the dimensionless velocity

Figure 3.11: Re-compression graphics for laminar flow.

The color plot here displays the variation of the $f'$ due to the effects of changing the pressure gradient. On the other hand, in the Cf plot, it is observed that the skin friction coefficient tendency is reaching 0, as the separation point is when the shear stress at the wall becomes 0 and thus the skin friction coefficient becomes 0.

Analyzing the solutions obtained for different values of alpha:

Table 3.2: Comparing results for different values of alpha for a laminar re-compression.

| **Method** | **alpha = 1/4** | **alpha = 1/2** | **alpha = 1** | **alpha = 5/4** |
|---|---|---|---|---|
| Code separation point | 0.365 | 0.217 | 0.119 | 0.097 |
| Integral method C=0.068 | 0.2933 | 0.1692 | 0.0916 | 0.0745 |
| Integral method C=0.09 | 0.3364 | 0.1982 | 0.1086 | 0.0866 |

When changing the parameter alpha the curvature of the velocity profile changes, which also produces changes in the skin friction coefficient. For small alpha values, the separation point (reaching zero skin friction coefficient) is retarded. The greater the alpha, the faster the separation occurs.

## 3.5. Turbulent re-compression case

As explained in the first chapter, there are several transition criteria such as Michel's, Granville or $e^n$ Method. For the simulations of the code an empirical correlation provided in Chapter 5 of Modeling and Computation of Boundary-Layer Flows [7] is going to be used.

$$R_{\theta_{tr}} = 1.174(1 + \frac{22400}{Re_{x_{tr}}})Re_{x_{tr}}^{0.46} \tag{3.8}$$

The code is prepared to be constantly checking if the local $Re_\theta$ and $Re_x$ is included or not in the expression given by 3.8. Specifically, to evaluate it, the local Reynolds value from the current $n$ position is taken into account and evaluated in equation 3.8, comparing it with the local $Re_\theta$. When the former overtakes the latter, transition occurs.

Additionally to this procedure that automatically computes the number of Reynolds at transition (a free transition), a forced transition can be introduced by specifying the desired Reynolds number. The code is going to consider the strictest value, the one that causes an earlier transition to turbulent boundary layer.

Applying this concept to the already studied re-compression case can be of interest as the separation point should be retarded in comparison to when having laminar flow.

For this specific case a forced transition Reynolds has been set at 5.5E5. However, with the empirical correlation an earlier transition Reynolds value is found around 5.2E5.

Applying the turbulence model for an inner and outer region and an $\alpha = 5/4$ and $u_0 = 100$ the results shown in Figure 3.12 are obtained.

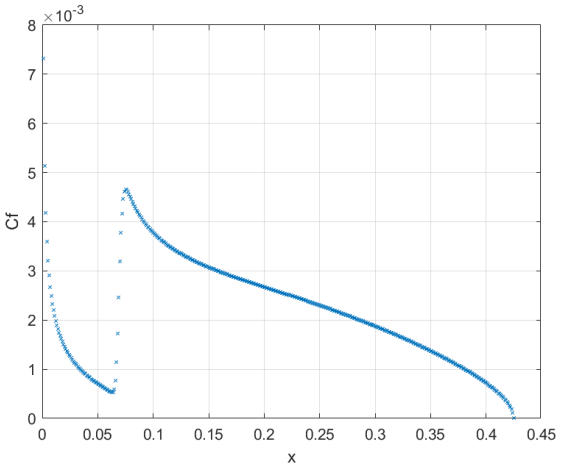
The threshold between the inner and outer layer has been considered around 10% - 15% of the boundary layer thickness [16] [17], according to some references. Additionally, this point can also be determined by the $y^+$ wall variable, which is computed as:

$$y^+ = \frac{y u_\tau}{\nu} \tag{3.9}$$

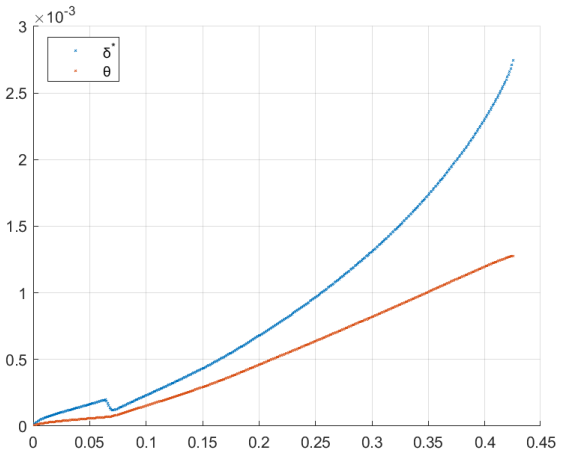
Being the shear velocity:

$$u_\tau = \sqrt{\tau_w / \rho} \tag{3.10}$$

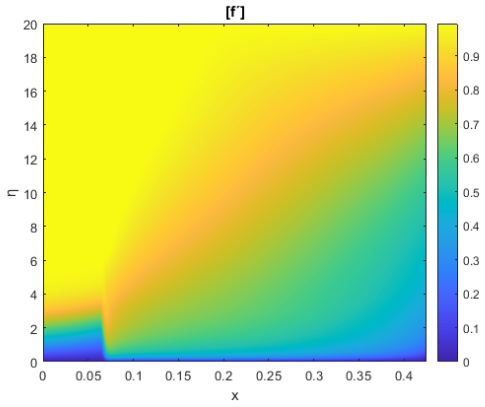
Literature can be found in establishing values for the different regions. For instance, the inner region is divided into diverse sublayers, the first one is the viscous sublayer which goes up to $y^+ = 5$. The last one is the log law region, being the upper limit depending on the Reynolds number. This limit is the transition point from the inner and outer layers.




(a) Skin friction evolution




(b) Momentum and displacement thickness evolution




(c) Color plot graphic

Figure 3.12: Results obtained from incorporating a turbulence model to the re-compression case.

These figures depict the expected behavior when making the transition to turbulence. The skin friction coefficient is proportional to the shear forces at the wall or surface. For a laminar flow, the shear forces are lower at the surface whereas turbulent flows have higher energy and so it can easily stay attached against an adverse pressure gradient. Moreover, it can be observed that the momentum thickness starts increasing faster as it is proportional to the velocity. The change of behavior of the displacement thickness is proportional to the behavior of the dimensionless stream function.

Finally, it can be observed that the separation point has been displaced further in comparison with the fully laminar case. If different alpha values are studied, it is observed as, in fact, when having a turbulent flow the separation point is delayed.

Table 3.3: Comparison of the separation point between fully laminar cases and with a turbulence model.

| **Method** | **alpha = 1/4** | **alpha = 1/2** | **alpha = 5/4** |
|---|---|---|---|
| Fully laminar [code] | 0.365 | 0.217 | 0.097 |
| Incorporating turbulence [code] | 0.89 | 0.74 | 0.43 |

## 3.6. Boundary layers on airfoils

Once the working of the code has been properly checked, several airfoils have been tested. To enable this, the Xfoil program has been used to compare the results and to obtain the free stream velocities required as inputs.

Xfoil is a program, developed by Mark Drela at MIT, that analyses subsonic airfoils.

The method used for the computation is based on a panel method. Once defined the working airfoil, the airfoil is divided into panels and is added, at the midpoint of each panel, a vortex. It is solved iteratively by the strength of these vortexes along the panels ensuring that the stagnation streamline has to follow the shape of the airfoil. For viscous computation, the integral momentum method together with kinetic energy shape parameter equations are used.

For computing the transition Xfoil uses two methods, taking the condition that implies an early transition. The first method consists in directly setting the chord location at which is desired to have the transition. If this location is not defined it is set by default at the trailing edge. The other method is the $e^N$ method. A critical value of N is specified and when it is met the transition occurs. A typical value of N for an average wind tunnel is 9. [14]

The program itself contains a database of NACA airfoils yet other airfoils can be easily loaded [15] and inviscid and viscous solutions can be obtained. Finally, the data exported from Xfoil can be imported to the developed code, using the Import M function.

### 3.6.1. NACA 0012

Starting with NACA 0012, a symmetrical airfoil with a maximum thickness of 12% of the chord, as represented in Figure 3.13.

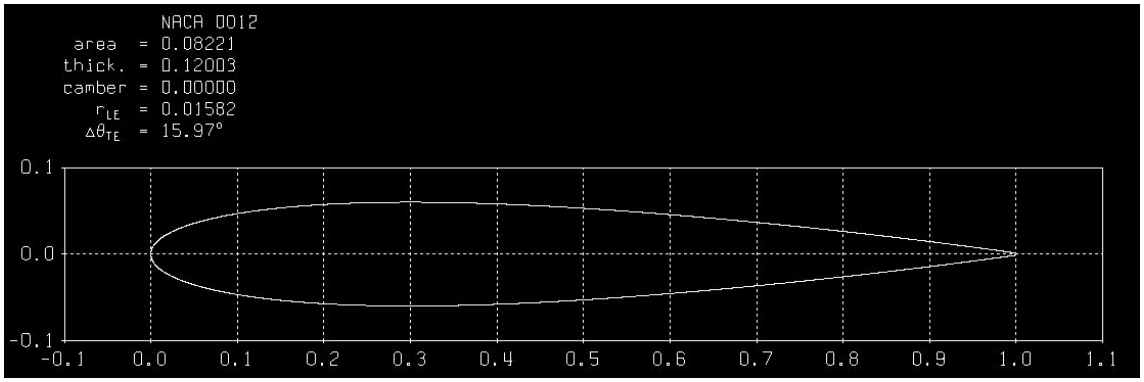



Figure 3.13: NACA 0012 profile. Source: Xfoil program

Firstly, simulating this profile with Xfoil at different angles of attack, the following results have been obtained for the lower surface and upper surface. It is expected to obtain the same results in the lower surface and upper surface when the angle of attack is 0, as it is a symmetrical airfoil, yet when changing the angle of attack the lower surface and upper surface are going to be different.

Running Xfoil for a Reynolds number of 60,000 the data from Table 3.4 is obtained.

Table 3.4: Transition points for different angles of attack at Reynolds 60,000.

| alpha | Top Xtr | Bot Xtr |
|---|---|---|
| -10 | 1 | 0.131 |
| -8 | 1 | 0.152 |
| -6 | 1 | 0.284 |
| -4 | 1 | 0.611 |
| -2 | 1 | 0.894 |
| 0 | 1 | 1 |
| 2 | 0.894 | 1 |
| 4 | 0.611 | 1 |
| 6 | 0.284 | 1 |
| 8 | 0.152 | 1 |
| 10 | 0.131 | 1 |

As can be observed in the table, for a 0 angle of attack the flow is completely laminar. However, when modifying the angle turbulence appears. This is considering a free transition and in fact, in the developed code this may not be appreciated because a separation point is reached earlier.

#### *3.6.1.1. Re= 60,000 and AoA= 0*

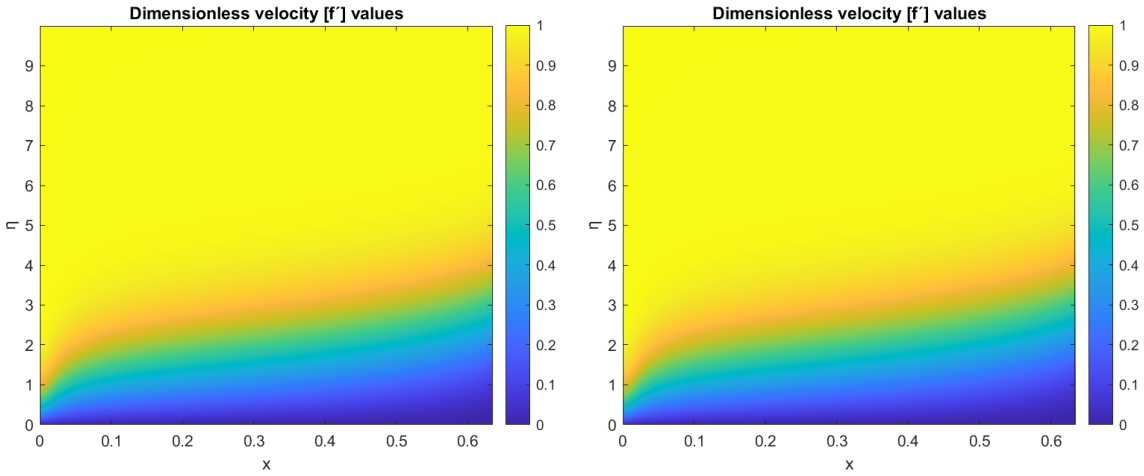
If simulating with the developed code the case with Re = 60,000 and an angle of attack of $0^{o}$, having a fully laminar flow, the results obtained are the ones shown in Figures 3.14 3.15.

The results from the lower surface and upper surface, as predicted, coincide. Additionally, it can be seen that the separation point, when the skin friction turns 0, is approximately the same position. Xfoils predict a separation point at 0.63911 and the developed code at 0.63505.

Along the test, it can be observed that Xfoil can compute further points from separation, as the system used is robust against small separations and can reattach the flow. However, in the programmed code the computations stop at the separation point.

Because of that, it can be observed that in the graphics done with the computations stop much earlier than in Xfoil yet the separation point with both methods coincide (when the skin friction coefficient turns negative in Xfoil is when the computations done in the code stops). Additionally, Xfoil computes approximately up to x=1.4, including part of the wake. In the code programmed it is computed until the end of the surface.

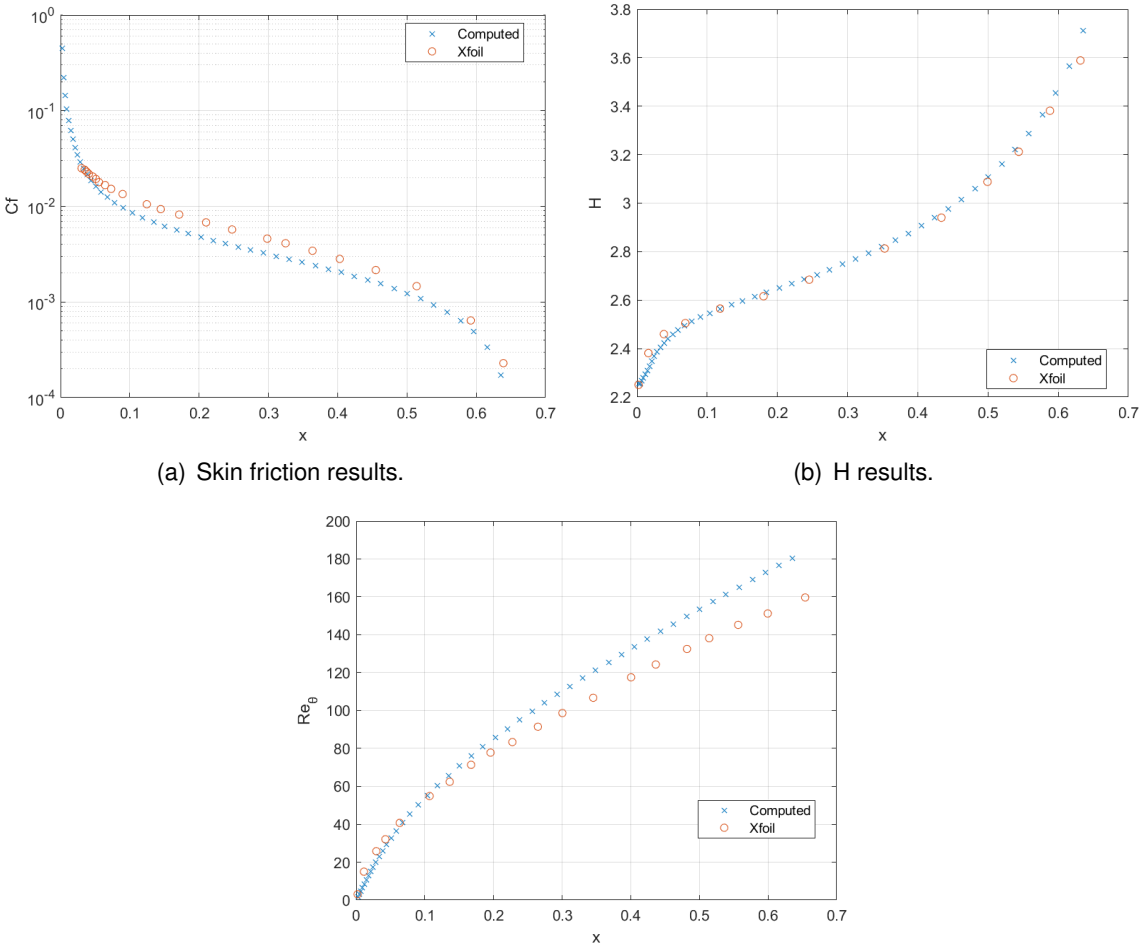
From the results obtained, shown in Figure 3.15, it can be seen that the results coincide pretty well between the predicted from Xfoil and the obtained from the developed code. The less accurate parameter is related to the momentum thickness. As had been seen previously, the momentum thickness was less accurate from the characteristic properties, and when computing $Re_{\theta}$ these mismatches are exposed.




(a) Color plot of the dimensionless velocity at upper surface

(b) Color plot of the dimensionless velocity at lower surface

Figure 3.14: Color plots of the upper surface and lower surface of a NACA 0012 with an angle of attack of 0º.




(a) Skin friction results.

(b) H results.

(c) $Re_\theta$ results.

Figure 3.15: Comparison of the results obtained from Xfoil and the code for a NACA 0012 at 0º.

#### *3.6.1.2. Re= 60,000 and AoA= 2*

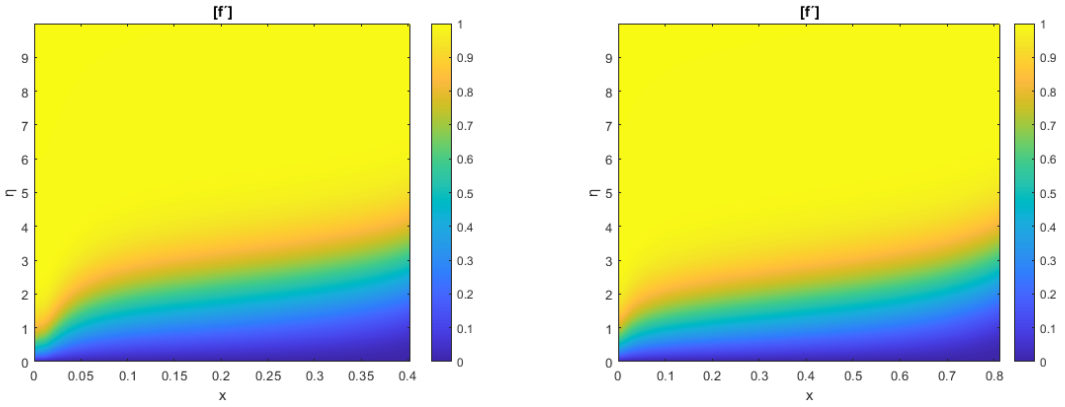
If modifying the angle of attack to 2º, different values between the lower surface and upper surface are expected. Also, a transition to turbulence around 0.849 is expected, at the top surface, yet in the code, it can not be appreciated as it separates around 0.4.

In this case, a positive alpha ($\alpha = 2$) with a number of Reynolds of 60,000 is going to be studied. However, a negative alpha would have the same solution as the positive one but with top and bottom surfaces exchanged, on account of the airfoil symmetry.

Figures 3.16 and 3.17 show the obtained results.

First of all, the different results from the lower and upper surfaces are appreciated. The separation point obtained from the developed code are 0.81335 and 0.40273, from the lower and upper, respectively. In Xfoil, very similar results are obtained: 0.82256 and 0.44004.

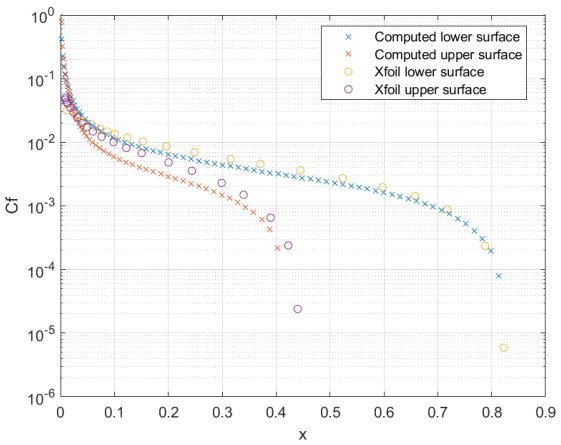
Again, if comparing the skin friction, H and $Re_\theta$ from the Xfoil and the code, it can be observed that the solutions fit quite well. The only case that mismatches is the $Re_\theta$, as explained in the previous subsection.



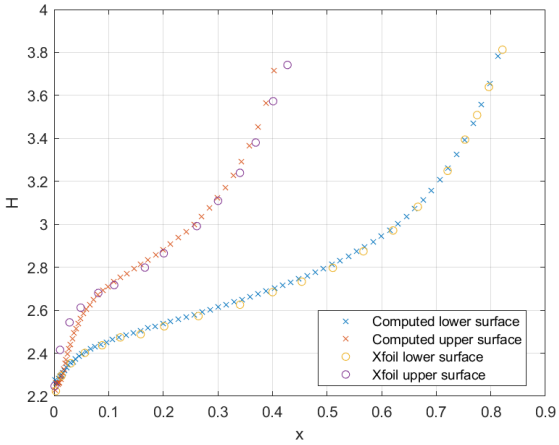

(a) Color plot of the dimensionless velocity at upper surface

(b) Color plot of the dimensionless velocity at lower surface

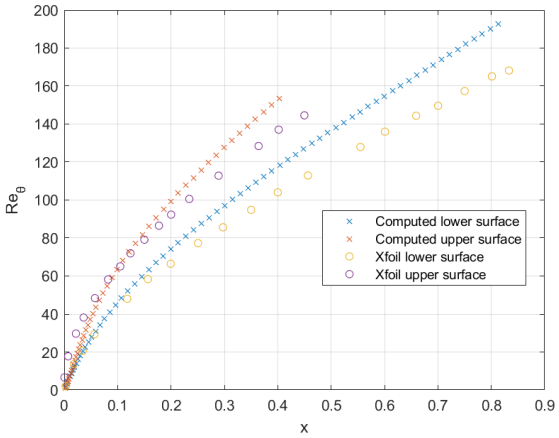
Figure 3.16: Color plots of the upper surface and lower surface of a NACA 0012 with an angle of attack of 2º.




(a) Skin friction results.




(b) H results



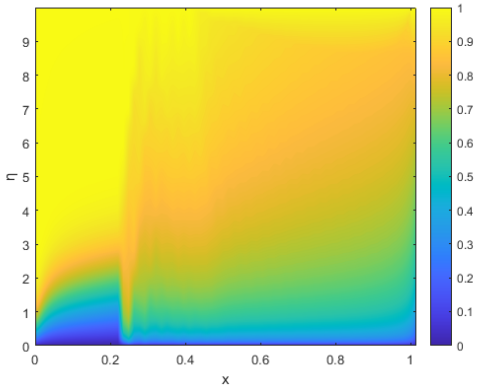

(c) $Re_\theta$ results.

Figure 3.17: Results from the lower surface and upper surface with Xfoil comparisons for Re= 60,000 and AoA= 2.

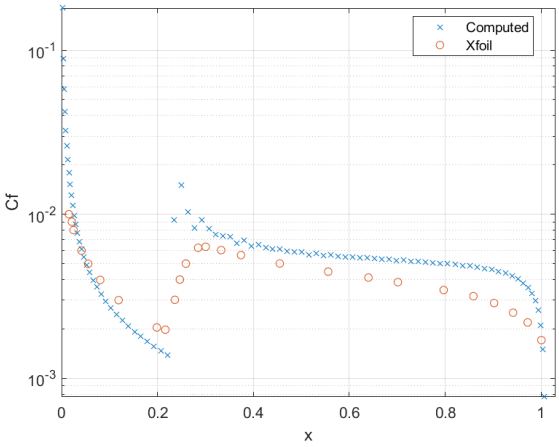
#### *3.6.1.3. Incorporating turbulence*

Now a turbulence case is simulated forcing the transition point at Re= 2.2E5 (x=0.22). The data is set from the inviscid solution from Xfoil, for a NACA 0012 at 0 degrees, and increasing the velocity 10 times its value. The results obtained are shown in Figure 3.18.

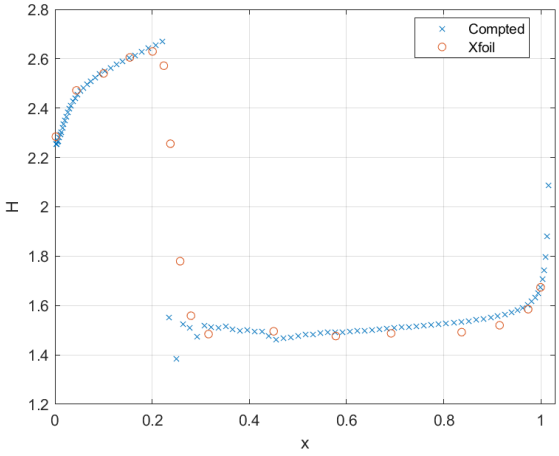
In order to be able to compare the obtained results, a similar case in Xfoil is simulated, defining $Re_x = 8E5$ (computing this value according to the obtained data from the code) and forcing the transition point at 0.22.



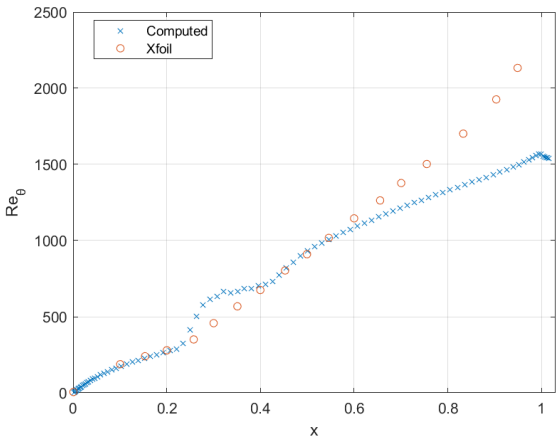

(a) Color plot of the dimensionless velocity.




(b) Skin friction coefficient results.




(c) H results.




(d) $Re_\theta$ results.

Figure 3.18: Comparison of the results obtained from Xfoil and the code for a NACA 0012 at 0º with turbulence.

The obtained results fit nearly perfectly in the laminar region and have some mismatches in the turbulent. However, these differences can be considered minor, especially in the skin friction (which is plot on a logarithmic scale) and H computation. Once again, the least accurate parameter is the $Re_\theta$.

All these differences may be due to several factors: the lack of accuracy between the simulated case from the code and the Xfoil case, the usage of different turbulence models, or some improving weak points in the developed code (as the definition between the inner and outer regions would require an extensive study).

### 3.6.2. SD7003

Finally, one last case that can be tested is changing the airfoil. For instance, SD7003 [15] is known as a high-performance airfoil for lower Reynolds numbers, because it has a high lift-to-drag ratio for low Reynolds.

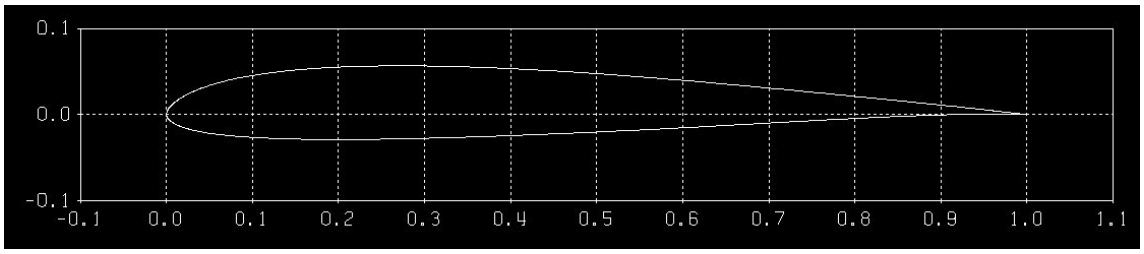
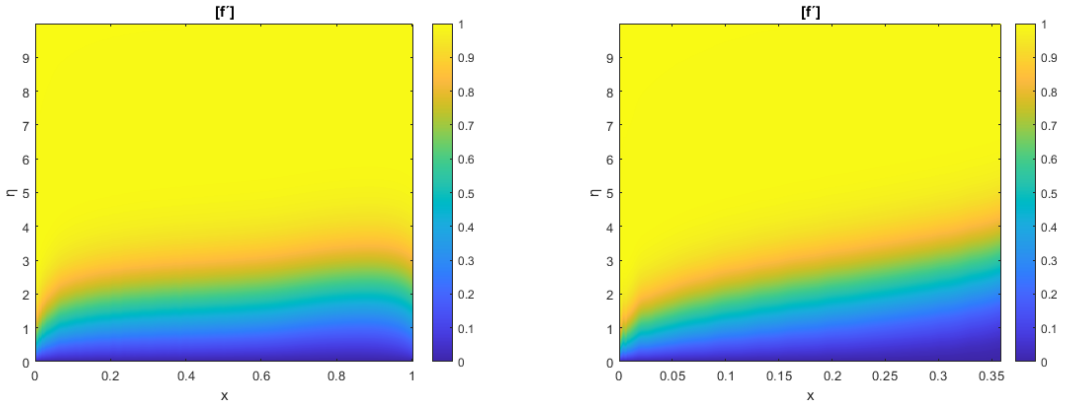



Figure 3.19: SD7003 profile. Source: Xfoil program

If simulating this profile for a number of Reynolds of 60,000 and an angle of attack of 2 degrees, the results obtained are the ones shown in Figures 3.20 and 3.21.

The first point to notice is the separation points. It is observed that the lower surface does not have a separation point, but the upper separate at x=0.35911 (according to the developed code) and at x= 0.37911 (according to Xfoil).



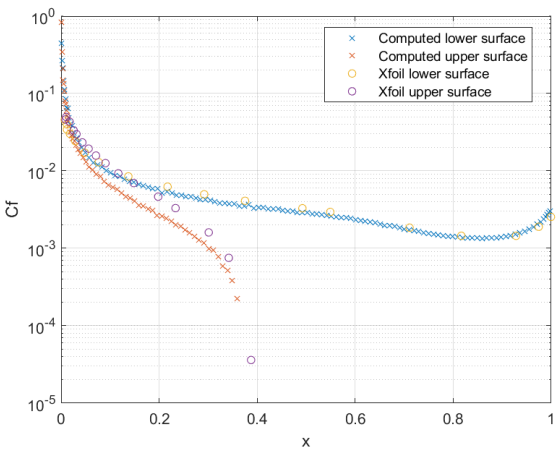

(a) Color plot of the lower surface

(b) Color plot of the upper surface

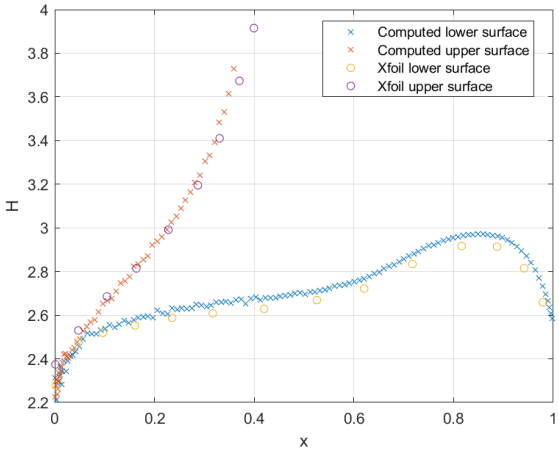
Figure 3.20: Dimensionless velocity results from the lower surface and upper surface of SD7003 for Re= 60,000 and AoA= 42.

In this last case, it can be observed how the expected results fit well with the Xfoil results, being $Re_\theta$ the one that mismatches more.

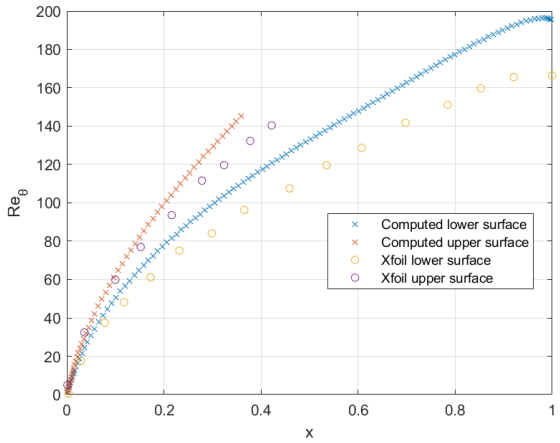
Because of this last test, the robustness of the code and its proper operating have been proved, and not only for symmetrical airfoils.




(a) Skin friction coefficient results




(b) H results




(c) $Re_\theta$ results.

Figure 3.21: Comparison of results from Xfoil and the developed code for a Re= 60,000 and AoA=2.

# CONCLUSIONS

This project aimed to develop a code able to solve the Navier-Stokes equations for the boundary layer region for a two-dimensional incompressible flow. Based on the tests executed, it is concluded that the program can successfully solve the equations for both laminar and turbulent flows.

The results indicate very precise solutions with the laminar flow yet further research is needed regarding the turbulent flow. For simulating turbulent flows a turbulence model has been used although some features would require further extensive studies. For instance, the transition point has been computed with an empirical correlation but other methods might be more accurate for the computation. Besides, the determination of the inner and outer regions of the turbulent boundary layer has been done experimentally by testing different values inside the ranges found in the literature. A specific analysis should be developed to study the behavior of this region to properly determine them accurately.

Moving on to the tests, an initial verification of the code has been done simulating the Blasius solution. For this concrete case, a nearly perfect solution has been obtained. For most of the graphics, the theoretical results and the obtained overlap. The only property that slightly diverges is the momentum thickness, as it is computed using the Riemann sum.

For a better resolution of the momentum thickness, a more accurate method could be used, for instance, Simpson's rule with variable steps. However, although it could improve, the solution with Riemann has been considered good enough. Afterwards, a stagnation point has been introduced. Again the results obtained are nearly perfect and the boundary layer thickness behaves as expected.

Once the flat plate and stagnation point cases have been successfully tested, some modifications were done regarding the upstream boundary condition velocity profile. A hyperbolic tangent has been tested as the dimensionless velocity at x=0, computing with backward differences the required values for initializing the profile. The results obtained, as expected, have been that the code can run under this new velocity profile. As it is a mathematical expression, the rate of growth of the parameters is not exactly the realistic one, and some mismatches are found when solving the equations. The general solution was correct if compared with their corresponding solutions, but some oscillating values and peaks appeared.

Finally, the last scenarios that have been tested were regarding the outer velocity distribution. An initial case of re-compression modeled by an equation has been tested for both laminar and turbulent flow. With the results obtained from the integral method, the separation points have been checked. Additionally, for the turbulent case, the analysis of the graphics has been of great interest as it can be observed how the separation point is retarded.

The other case tested of outer velocity distributions were airfoils, using inviscid flow calculations obtained with the Xfoil software. For these cases, it has been observed that laminar solutions fit perfectly the predicted by Xfoil yet with the distinction that Xfoil can compute further points after separation and even re-attachments of the flow, whereas the code developed stops at separation. One last case of turbulence has been tested with inviscid data to check that the properties behave as expected.

In conclusion, the code developed offers a nearly perfect resolution for laminar flows and a good for turbulence cases. Further points of study to continue developing the code would be focused on the turbulence model and also trying to incorporate time evolution. Additionally, another working line could consist in optimizing the running of the code and timings, including the possibility of changing the programming language.

# BIBLIOGRAPHY


[1] *Derivation of the Boundary Layer Equations*. (2016, March 23). YouTube. Retrieved 18 February 2022, from https://www.youtube.com/watch?v=h3RuIhR6TTU 57

[2] Asaithambi, A. (2021). *On Solving the Nonlinear Falkner–Skan Boundary-Value Problem*: A Review. Fluids, 6(4), 153. https://doi.org/10.3390/fluids6040153 57

[3] Gibiansky, A. (2021). *Fluid Dynamics*: The Navier-Stokes Equations. https://andrew.gibiansky.com 57

[4] Schmidt-Didlaukies, H. (2014, May). *The Navier-Stokes Equations*. Massachusetts Institute of Technology. https://math.mit.edu/classes/18.086/2014/reports/HenrikSchmidtDidlaukies.pdf 57

[5] Conservation of Mass. (n.d.). *SolidMechanicsBooks*. Retrieved 23 February 2022, from https://pkel015.connect.amazon.auckland.ac.nz/SolidMechanicsBooks 57

[6] Cebeci, T., Platzer, M., Chen, H., Chang, K., Shao, J. P. (2005). *Analysis of Low-Speed Unsteady Airfoil Flows* (2005 ed.). Springer. (pages 21-29) (pages 59-76) 1, 11, 57

[7] Cebeci, T., Cousteix, J. (2005).*Modeling and Computation of Boundary-Layer Flows: Laminar, Turbulent and Transitional Boundary Layers in Incompressible and Compressible Flows* (2nd 2005 ed.). Springer. (pages 44-52) (pages 66-73) (pages 117-118) (pages 176-179) 1, 8, 11, 27, 40, 57

[8] Cebeci, T. (2005). *Computational Fluid Dynamics for Engineers.* Horizons Pub. Incorporated. (pages 41-47) (pages 83-85) (pages 211-236) 1, 3, 11, 16, 57

[9] White, F. M. (2011). *Fluid Mechanics.* McGraw-Hill Education. (Chapter 7) 5

[10] Newton's Method. (2018, March 6). [Video]. YouTube. https://www.youtube.com/watch?v=-5e2cULI3H8 57

[11] Schlichting, H., Kestin, J. (1979). *Boundary-layer Theory.* McGraw-Hill Education. (Chapter 1 Chapter 2) xi, 3, 5, 6

[12] Britannica, T. Editors of Encyclopaedia (2020, March 6).*boundary layer. Encyclopedia Britannica.* https://www.britannica.com/science/boundary-layer 3

[13] Meyers, R. A. (2002). *Encyclopedia of Physical Science and Technology* [E-book]. Amsterdam University Press. Retrieved 27 May 2022, from https://www.sciencedirect.com/science/article/pii/B0122274105009066 7

[14] Drela, M. (n.d.). *XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils.* MIT Dept. of Aeronautics and Astronautics, Cambridge Massachusetts. 43

[15] UIUC Airfoil Data Site. (2022). *UIUC Applied Aerodynamics Group.* Retrieved 26 June 2022 43, 48

[16] Sricharan Srinath. *Modeling and prediction of near wall turbulent flows.* Fluids mechanics [physics.class-ph]. Ecole Centrale de Lille, 2017. English. 8, 41

[17] Ansys. (2020). *Turbulent Boundary Layers* [Slides]. Ansys. https://courses.ansys.com/wp-content/uploads/2020/09/Basics-of-Turbulent-Flows-Lesson-6-Handout.pdf 8, 41

[18] Mellibovsky, F. (2020, May 9). *Boundary Layer Theory - Aerodynamics* [Slides]. Escola d'Enginyeria de Telecomunicaci´o i Aeroespacial de Castelldefels Universitat Polit'ecnica de Catalunya. 3, 6, 31, 35

[19] Flow Separation - *Boundary layer separation explained.* (2021, September 21). [Video]. YouTube. https://www.youtube.com/watch?v=HfICmKv9SzE 8

[20] FIEDLERT, H., HEAD, M. R. (n.d.). Intermittency measurements in the turbulent boundary layer (J. Fluid Meeh. (1966), wol. 25, part 4, pp. 719–735). xi, 28, 29

# APPENDICES

# APPENDIX A. MATHEMATICS-PHYSICS DEVELOPMENT

## A.1. Navier-Stokes equations

Navier-Stokes equations were named after Claude-Louis Navier and George Gabriel Stokes. These equations determine the motion of Newtonian viscous fluid and state the momentum and mass conservation.

They are obtained following the mechanics and thermodynamics principles applied to a volume and combining Newton's second low, viscosity, and fluid stresses, and the pressure concept. By doing this, the integral formulation of Navier-Stokes equations is obtained; nevertheless, the differential formulation is usually more useful when implemented.

Navier-Stokes equations are non-linear partial derivatives equations and only for some particular cases of fluxes, an analytical solution is met. For obtaining a general approximation in most cases numerical methods are required; this branch of research is known as CFD (Computational Fluid Dynamics).

For developing the equations the guides from [6], [7] and [8] have been followed. And secondarily [1], [2], [3] [4], [5] and [10]

### A.1.1. Development of Navier-Stokes equations

**CONSERVATION OF MASS**

Mass can neither be created nor destroyed; it remains constant in a system. It can be expressed as follows.

$$\frac{dm}{dt} = 0 \tag{A.1}$$

Rewriting the mass conservation equations for a given control volume (being $\Omega$ the control volume and S its boundary, the control surface), it can be expressed as the flow rate of the control volume and the control surface, being the sum 0. Expressed in the integral form we have:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \cdot d\Omega + \int_{S} \rho v n \cdot dS = 0 \tag{A.2}$$

Denoting the density as $\rho$ and the normal vector to S as n.
And in the differential form:

$$\frac{\partial \rho}{\partial t} + \nabla(\rho v) = 0 \tag{A.3}$$

If we consider the fluid as in-compressible, constant density, the divergence has to be 0 ($\nabla \cdot v = 0$) since changes in density along the time would imply compression or expansion of the fluid. Resulting in the following continuity equation form.

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{A.4}$$

**CONSERVATION OF MOMENTUM**

The momentum can be defined by the mass of a given object multiplied by its velocity. This

momentum remains constant if no external forces are described.

$$\frac{dmv}{dt} = \sum f \tag{A.5}$$

Newton's Second Law states that force equals mass times acceleration. This expression can be rewritten as density multiplied by the derivative over time of the velocity, being the velocity function of time and x,y,z; which using the concept of material derivative can be written easily.

$$\vec{F} = \rho \frac{D\vec{v}}{Dt} = \rho(\frac{\partial \vec{v}}{\partial t} + \vec{v}\nabla\vec{v}) \tag{A.6}$$

It can be considered that this force is caused by stresses (using stress tensor $\sigma$) and external forces.

$$\vec{F} = \nabla\sigma + \vec{f} \tag{A.7}$$

The stress tensor for the Navier-Stokes application is usually divided into two terms: the volumetric stress tensor (pressure terms) and the stress deviator tensor (shears stresses that determine the deformation and movement).
Putting together the concepts the following equation is obtained.

$$\rho \frac{D\vec{v}}{Dt} = -\nabla p + \nabla T + \vec{f} \tag{A.8}$$

Being T the stress deviator tensor.
Moreover, as a general approximation, it is considered the fluid incompressible and Newtonian (viscosity only varies with temperature or pressure). For a Newtonian fluid:

$$\tau_{ij} = \mu(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) \tag{A.9}$$

Being $\mu$ the viscosity of the fluid.
Applying this concept and operating mathematically, the stress deviator tensor divergence is equal to Laplacian vector.

$$\nabla T = \mu\nabla^2\vec{v} \tag{A.10}$$

Adding this last term, the final Navier-Stokes equation obtained for an incompressible Newtonian fluid is obtained.

$$\rho \frac{D\vec{v}}{Dt} = -\nabla p + \mu\nabla^2\vec{v} + \vec{f} \tag{A.11}$$

### A.1.2. Turbulent flow

In order to take into account the turbulent flows, and not exclusively laminar, the instantaneous quantities (u,v,w,etc) are replaced by their mean value plus a fluctuating term.

$$\rho \frac{D\bar{u}}{Dt} = -\frac{\partial \bar{p}}{\partial x} + \mu\nabla^2\bar{u} + \rho\bar{f}_x - \rho\frac{\partial}{\partial x}(\overline{u'^2}) - \rho\frac{\partial}{\partial y}(\overline{u'v'}) - \rho\frac{\partial}{\partial z}(\overline{u'w'}) \tag{A.12}$$

$$\rho \frac{D\bar{v}}{Dt} = -\frac{\partial \bar{p}}{\partial y} + \mu\nabla^2\bar{v} + \rho\bar{f}_y - \rho\frac{\partial}{\partial x}(\overline{v'u'}) - \rho\frac{\partial}{\partial y}(\overline{v'^2}) - \rho\frac{\partial}{\partial z}(\overline{v'w'}) \tag{A.13}$$

$$\rho \frac{D\bar{w}}{Dt} = -\frac{\partial \bar{p}}{\partial z} + \mu\nabla^2\bar{w} + \rho\bar{f}_z - \rho\frac{\partial}{\partial x}(\overline{w'u'}) - \rho\frac{\partial}{\partial y}(\overline{w'v'}) - \rho\frac{\partial}{\partial z}(\overline{w'^2}) \tag{A.14}$$

### A.1.3. General approximations

First of all, two-dimensional, steady, and incompressible flows are going to be studied and from that point, several approximations and reductions of the Navier-Stokes equations are going to be done to adapt them to the boundary layer case.

Analysing the order of magnitude of the terms of the equations, additional approximations can be done. For instance, $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \sim \frac{u_e}{L^2} + \frac{u_e}{\delta^2}$ being $\frac{u_e}{L^2} << \frac{u_e}{\delta^2}$. Thus the derivative term regarding x can be neglected.

Additionally, observing the order of magnitude of the terms, the y-momentum equation can be approximated to $\frac{\partial p}{\partial y} = 0$. And thus the resulting momentum equations with turbulent modeling is just:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2 u}{\partial y^2} - \frac{\partial}{\partial y}(\overline{u'v'}) \tag{A.15}$$

Finally, applying the freestream values, the zone where Bernoulli's equations can be applied, the derivative of pressure can be defined as:

$$\frac{dp}{dx} = -\rho u_e \frac{du_e}{dx} \tag{A.16}$$

## A.2. Stream function

The stream function is defined for incompressible 2D flows and it enables working with a single variable of higher order, rather than two variables (u and v). The continuity equation can be written in terms of the stream function which allows them to solve directly the momentum equation, as there is only one variable. Streamlines are defined as lines of a constant value of $\psi$. Stream function is defined as

$$\begin{aligned} u &= \frac{\partial \psi}{\partial y} \\ v &= -\frac{\partial \psi}{\partial x} \end{aligned} \tag{A.17}$$

### A.2.1. Applying stream function concept in Navier-Stokes equations

Starting from the equation defined in A.15 and using eddy viscosity concept $\rho\overline{u'v'} = \rho v_t \frac{\partial u}{\partial y}$ it is obtained:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2 u}{\partial y^2} + \frac{\partial}{\partial y}(v_t\frac{\partial u}{\partial y}) \tag{A.18}$$

Using the stream function (A.17):

$$\frac{\partial \psi}{\partial y}\frac{\partial}{\partial x}(\frac{\partial \psi}{\partial y}) - \frac{\partial \psi}{\partial x}\frac{\partial}{\partial y}(\frac{\partial \psi}{\partial y}) = -\frac{1}{\rho}\frac{dp}{dx} + \nu\frac{\partial^2}{\partial y^2}(\frac{\partial \psi}{\partial y}) + \frac{\partial}{\partial y}(v_t\frac{\partial}{\partial y}(\frac{\partial \psi}{\partial y})) \tag{A.19}$$

And, in order to simplify the previous equation, it is used prime to denote differentiation with respect to y.

$$\psi'\frac{\partial \psi'}{\partial x} - \frac{\partial \psi}{\partial x}\psi'' = -\frac{1}{\rho}\frac{dp}{dx} + \nu\psi''' + (v_t\psi'')') \tag{A.20}$$

Rearranging the terms and using A.16 it is obtained the final form of the expression.

$$((\nu+\nu_t)\psi'')' + u_e\frac{du_e}{dx} = \psi'\frac{\partial\psi'}{\partial x} - \psi''\frac{\partial\psi}{\partial x} \tag{A.21}$$

By doing these transformations, single variable equations have been obtained, only depending on $\psi$ .

## A.3. Falkner-Skan

In order to solve the resulting equation, a transformation of coordinates is done to reduce the equation to an ordinary differential equation. For this transformation, Falkner-Skan-type coordinate is going to be used and studied with the following change of coordinates.

$$\begin{aligned} \xi &= x/L \\ \eta &= \frac{y}{g(x)} = y\sqrt{\frac{u_e}{\nu x}} \end{aligned} \tag{A.22}$$

Being $f$ the dimensionless stream function.

$$\psi(x,y) = \sqrt{u_e \nu x} f(x,\eta) \tag{A.23}$$

Additionally, the partial derivatives that must be considered for this change of variables can be expressed as:

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{A.24}$$

As had been seen $\xi$ is the x, meaning that the partial derivative of $\xi$ with respect to x is 1. Moreover, as expressed in A.22, $\eta$ = y / g(x) . Because of that it can be computed the partial derivative with respect to x.

$$\begin{aligned} \frac{\partial}{\partial x} &= \frac{\partial}{\partial \xi} - \frac{y}{g^2}\frac{dg}{dx}\frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial x} &= \frac{\partial}{\partial x} - \frac{\eta}{g}\frac{dg}{dx}\frac{\partial}{\partial \eta} \end{aligned} \tag{A.25}$$

On the other hand, the partial derivative with respect to y can be computed similarly.

$$\begin{aligned} \frac{\partial}{\partial y} &= \frac{\partial}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial}{\partial y}\frac{\partial \eta}{\partial \eta} \\ \frac{\partial}{\partial y} &= \frac{1}{g}\frac{\partial}{\partial \eta} \end{aligned} \tag{A.26}$$

Finally, in the same way the second derivative can also be expressed as followed.

$$\begin{aligned} \frac{\partial^2}{\partial y^2} &= \frac{\partial}{\partial y}\left(\frac{\partial}{\partial y}\right) = \frac{1}{g}\frac{\partial}{\partial \eta}\left(\frac{1}{g}\frac{\partial}{\partial \eta}\right) \\ \frac{\partial^2}{\partial y^2} &= \frac{1}{g^2}\frac{\partial^2}{\partial \eta^2} \end{aligned} \tag{A.27}$$

With the transformations found (A.27 A.26 A.25) and the transformation of variables and dimensionless stream function relation; it is applied the Falkner Skan transformation on the equation A.21.

In order to simplify the procedure, the transformation is going to be done step by step and working with the terms separately; naming the left part of the equation ① + ② and the right part ③ -④ .

Denoting prime differentiation with respect to y:

$$① = ((\nu+\nu_t)\psi'')' \quad ② = u_e\frac{du_e}{dx} \quad ③ = \psi'\frac{\partial\psi'}{\partial x} \quad ④ = \psi''\frac{\partial\psi}{\partial x}$$

Applying the corresponding transformations and hereon denoting prime as differentiation with respect to $\eta$.

$$① = \frac{1}{g}\frac{\partial}{\partial\eta}((\nu+\nu_t)\frac{1}{g^2}\frac{\partial^2}{\partial\eta^2}(f\sqrt{u_e\nu x})) = \frac{1}{g}\frac{\partial}{\partial\eta}(\frac{(\nu+\nu_t)}{g^2}f''\sqrt{u_e\nu x}) = \frac{1}{g^3}\sqrt{u_e\nu x}((\nu+\nu_t)f'')' =$$

$$= \sqrt{\frac{u_e\nu x u_e^3}{\nu^3x^3}}((\nu+\nu_t)f'')' = \sqrt{\frac{u_e^4}{\nu^2x^2}}((\nu+\nu_t)f'')' = \frac{u_e^2}{\nu x}((\nu+\nu_t)f'')' = \frac{u_e^2}{x}(bf'')'$$

$$Being \quad b = 1+\frac{\nu_t}{\nu}$$

$$② = m\frac{u_e^2}{x} \quad Being \quad m = \frac{x}{u_e}\frac{du_e}{dx}$$

$$③ = \frac{1}{g}\frac{\partial}{\partial\eta}(f\sqrt{u_e\nu x})(\frac{\partial}{\partial x}(\frac{1}{g}\frac{\partial}{\partial\eta}(f\sqrt{u_e\nu x})) - \frac{\eta}{g}\frac{dg}{dx}\frac{\partial}{\partial\eta}(\frac{1}{g}\frac{\partial}{\partial\eta}(f\sqrt{u_e\nu x}))) =$$

$$= \frac{\sqrt{u_e\nu x}}{g}f'(\frac{\partial}{\partial x}(\frac{\sqrt{u_e\nu x}}{g}f') - \frac{\eta}{g}\frac{dg}{dx}\frac{\partial}{\partial\eta}(\frac{\sqrt{u_e\nu x}}{g}f')) = u_ef'(\frac{\partial}{\partial x}(u_ef') - \frac{\eta}{g}\frac{dg}{dx}\frac{\partial}{\partial\eta}(u_ef\prime)) =$$

$$= u_ef'(\frac{due}{dx}f' + \frac{\partial f'}{\partial x}u_e - \frac{\eta}{g}\frac{dg}{dx}u_ef'')$$

$$④ = \frac{1}{g^2}\frac{\partial^2}{\partial\eta^2}(f\sqrt{u_e\nu x})(\frac{\partial}{\partial x}(f\sqrt{u_e\nu x}) - \frac{\eta}{g}\frac{dg}{dx}\frac{\partial}{\partial\eta}(f\sqrt{u_e\nu x})) = \frac{\sqrt{u_e\nu x}}{g^2}f''(\frac{\partial f}{\partial x}\sqrt{u_e\nu x} + \frac{\sqrt{u_e\nu}}{2\sqrt{x}}f +$$

$$+\frac{\partial\sqrt{u_e}}{\partial x}f\sqrt{x\nu} - \frac{\eta}{g}\frac{dg}{dx}f'\sqrt{u_e\nu x}) = \sqrt{\frac{u_e^3}{\nu x}}f''(\frac{\partial f}{\partial x}\sqrt{u_e\nu x} + \frac{\sqrt{u_e\nu}}{2\sqrt{x}}f + \frac{\partial\sqrt{u_e}}{\partial x}f\sqrt{x\nu} - \frac{\eta}{g}\frac{dg}{dx}f'\sqrt{u_e\nu x})$$

Gathering all the parts of the equations (① +② =③ -④) it is obtained:

$$\frac{u_e^2}{x}(bf'')' + m\frac{u_e^2}{x} = u_ef'(\frac{due}{dx}f' + \frac{\partial f'}{\partial x}u_e - \frac{\eta}{g}\frac{dg}{dx}u_ef'') - \sqrt{\frac{u_e^3}{\nu x}}f''(\frac{\partial f}{\partial x}\sqrt{u_e\nu x} + \frac{\sqrt{u_e\nu}}{2\sqrt{x}}f + \frac{\partial\sqrt{u_e}}{\partial x}f\sqrt{x\nu} - \frac{\eta}{g}\frac{dg}{dx}f'\sqrt{u_e\nu x}) \tag{A.28}$$

Working with the equations and simplifying the expressions with common factors and using the change of $m = \frac{x}{u_e}\frac{due}{dx}$:

$$\frac{u_e^2}{x}(bf'')' + m\frac{u_e^2}{x} = u_e(f')^2\frac{du_e}{dx} + u_e^2f'\frac{\partial f'}{\partial x} - \frac{\eta}{g}u_e^2\frac{dg}{dx}f''f' - (\frac{u_e^2}{2x}ff'' + f''\frac{\partial f}{\partial x}u_e^2 + \frac{\partial\sqrt{u_e}}{\partial x}ff''\sqrt{u_e^3}$$

$$-u_e^2f''f'\frac{\eta}{d}\frac{dg}{dx}) = m\frac{u_e^2}{x}(f')^2 + u_e^2(f')\frac{\partial f'}{\partial x} - \frac{u_e^2}{2x}ff'' - f''\frac{\partial f}{\partial x}u_e^2 - ff''\frac{u_e^2}{x}\frac{m}{2}$$

$$
\begin{aligned}
\textit{Rearranging} \quad & \frac{u_e^2}{x}(bf'')' + m\frac{u_e^2}{x} - m\frac{u_e^2}{x}(f')^2 + \frac{u_e^2}{2x}ff'' + ff''\frac{u_e^2}{x}\frac{m}{2} = u_e^2 f'\frac{\partial f'}{\partial x} - f''\frac{\partial f}{\partial x}u_e^2 \\
& (bf'')' + m - m(f')^2 + \frac{ff''}{2} + ff''\frac{m}{2} = x(f'\frac{\partial f'}{\partial x} - f''\frac{\partial f}{\partial x}) \\
& (bf'')' + \frac{m+1}{2}ff'' + m(1-(f')^2) = x(f'\frac{\partial f'}{\partial x} - f''\frac{\partial f}{\partial x})
\end{aligned} \tag{A.29}
$$

## A.4. Keller's Box method

Keller's Box method is an implicit method in which given a set of differential equations a first-order system is obtained. For achieving a solution with this method an initial change of variables is done to reduce the system to a first-order one. Then, by using central differences the center value of the cells of the grid can be numerically formulated. Finally, the resulting system is nonlinear and Newton's method is used for writing the equations in matrices.

### A.4.1. Reducing the order

New variables are defined in order to reduce the order of the system A.29, replacing the first and second derivatives of the dimensionless stream function.

$$
\begin{aligned}
f' &= u \\
f'' &= u' = v
\end{aligned} \tag{A.30}
$$

Making this substitution, equation A.29 can be written as:

$$
(bv)' + \frac{m+1}{2}fv + m(1-u^2) = x(u\frac{\partial u}{\partial x} - v\frac{\partial f}{\partial x}) \tag{A.31}
$$

With boundary conditions:

$$
\begin{aligned}
\eta = 0, u = 0, f = f_w(x) \\
\eta = \eta_e, u = 1
\end{aligned} \tag{A.32}
$$

### A.4.2. Centred difference derivatives

Establishing the centered difference derivatives and relating with the A.30 variables, the derivative of $f$ as $u$ and the derivative of $u$ as $v$ can be expressed. Considering the grid representation shown in Figure A.4, the central-difference derivatives are expressed as:

$$
\frac{f_j^n - f_{j-1}^n}{h_j} = \frac{u_j^n + u_{j-1}^n}{2} = u_{j-1/2}^n \tag{A.33}
$$

$$
\frac{u_j^n - u_{j-1}^n}{h_j} = \frac{v_j^n + v_{j-1}^n}{2} = v_{j-1/2}^n \tag{A.34}
$$

For the momentum equation, it is desired to find the values at the midpoint $(x^{n-1/2}, \eta_{j-1/2})$. For achieving the centre position an initial intermediate step is done to find first the position $(x^{n-1/2}, \eta)$.

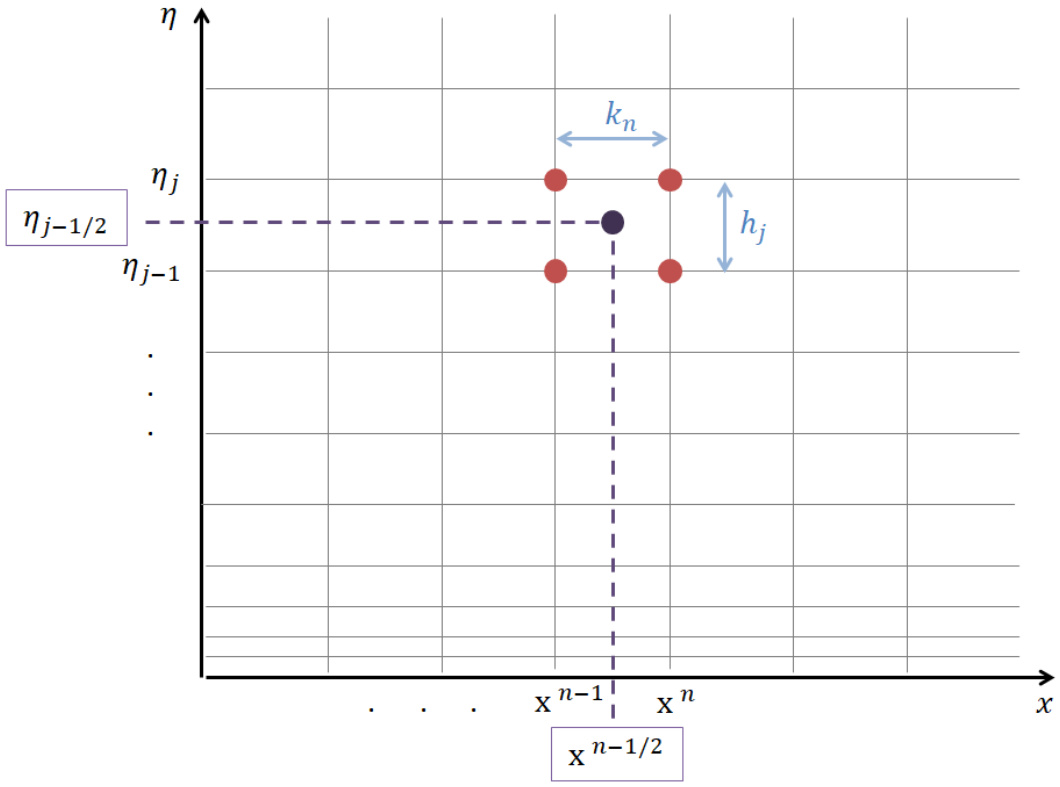



Figure A.1: Grid representation.

Denoting the left part of the equation A.31 as $L$, it can be approximated as follows.

$$\frac{1}{2}(L^n + L^{n-1}) = x^{n-1/2}(u^{n-1/2}(\frac{u^n - u^{n-1}}{k_n}) - v^{n-1/2}(\frac{f^n - f^{n-1}}{k_n})) \tag{A.35}$$

Rewriting the right part of the equation using the corresponding centered difference derivatives expressions:

$$\frac{1}{2}(L^n + L^{n-1}) = x^{n-1/2}(\frac{u^n + u^{n-1}}{2}(\frac{u^n - u^{n-1}}{k_n}) - \frac{v^n + v^{n-1}}{2}(\frac{f^n - f^{n-1}}{k_n})) \tag{A.36}$$

It can be observed as the factor ½ is simplified. Additionally, in order to rewrite and express the equation in an easier computing manner, it is renamed as $\alpha^n$ the term $\frac{x^{n-1/2}}{k_n}$. Making this substitution and operating the remaining terms it is obtained:

$$(L^n + L^{n-1}) = \alpha^n((u^n + u^{n-1})(u^n - u^{n-1}) - (v^n + v^{n-1})(f^n - f^{n-1}))$$

$$(L^n + L^{n-1}) = \alpha^n((u^n)^2 - (u^{n-1})^2 - (vf)^n + v^n f^{n-1} - v^{n-1} f^n + (vf)^{n-1})$$

$$Being \quad L^n = ((bv)')^n + \frac{m^n + 1}{2}(fv)^n + m^n(1 - (u^n)^2)$$

$$((bv)')^n + \frac{m^n + 1}{2}(fv)^n + m^n(1 - (u^n)^2) + L^{n-1} = \alpha^n((u^n)^2 - (u^{n-1})^2 - (vf)^n + \\ + v^n f^{n-1} - v^{n-1} f^n + (vf)^{n-1}) \tag{A.37}$$

Then the common factors $(fv)^n$ and $(u^2)^n$ can be grouped and their coefficients renamed to make the writing of the equation easier.

$$((bv)')^n + (\frac{m^n+1}{2} + \alpha^n)(fv)^n + m^n - (m^n+\alpha^n)(u^n)^2) = -L^{n-1} + \alpha^n(-(u^{n-1})^2 + \\ + v^n f^{n-1} - v^{n-1} f^n + (vf)^{n-1})$$

$$Being \quad \alpha_1 = \frac{m^n+1}{2} + \alpha^n \qquad \alpha_2 = m^n + \alpha^n$$

$$((bv)')^n + \alpha_1(fv)^n + m^n - \alpha_2(u^n)^2) = -L^{n-1} + \alpha^n(-(u^{n-1})^2 + v^n f^{n-1} - v^{n-1} f^n + (vf)^{n-1})$$

Rearranging the terms and grouping under the name $R^{n-1}$ the following terms:

$$R^{n-1} = -L^{n-1} + \alpha^n((vf)^{n-1} - (u^{n-1})^2) - m^n$$

The final momentum equation at $(x^{n-1}, \eta)$ is:

$$((bv)')^n + \alpha_1(fv)^n - \alpha_2(u^n)^2) + \alpha^n(v^{n-1} f^n - v^n f^{n-1}) = R^{n-1} \tag{A.38}$$

Once the equation is written at $(x^{n-1/2}, \eta)$, it is repeated the same procedure done before, but now centring the equations at $\eta_{j-1/2}$. Resulting in obtaining the final equation at the centre point $(x^{n-1/2}, \eta_{j-1/2})$. The $\alpha$ coefficients are not going to be modified as they are function of the dimensionless pressure gradient (m) which is only dependent on x (n variations).

$$\frac{(b_j^n v_j^n - b_{j-1}^n v_{j-1}^n)}{h_j} + \alpha_1(fv)_{j-1/2}^n - \alpha_2(u^2)_{j-1/2}^n + \alpha^n(v_{j-1/2}^{n-1} f_{j-1/2}^n - f_{j-1/2}^{n-1} v_{j-1/2}^n) = R_{j-1/2}^{n-1} \tag{A.39}$$

$$With \quad R_{j-1/2}^{n-1} = -L_{j-1/2}^{n-1} + \alpha^n((fv)_{j-1/2}^{n-1} - (u^2)_{j-1/2}^{n-1}) - m^n \tag{A.40}$$

$$L_{j-1/2}^{n-1} = \frac{(b_j^{n-1} v_j^{n-1} - b_{j-1}^{n-1} v_{j-1}^{n-1})}{h_j} + \frac{m^{n-1}+1}{2}(fv)_{j-1/2}^{n-1} + m^{n-1}(1 - (u^2)_{j-1/2}^{n-1}) \tag{A.41}$$

With boundary conditions:

$$f_0^n = f_w, \quad u_0^n = 0, \quad u_J^n = 1 \tag{A.42}$$

### A.4.3. Newton's Method

In A.39 A.40 A.41 it is obtained a discretized equation yet it is non linear. To linearize it and obtain a solution, Newton's Method is applied. This method consists in getting an approximation of the values following an iterative manner.

First of all, in general, Newton's Method works starting with an initial guess. Then, it is evaluated the value of the function at that point and is computed the tangent at that initial point. The relation between them returns the increment that must be added to the initial guess to obtain a second new guess value. Like this, given an initial guess, it is possible to obtain a more accurate solution for each iteration. Expressed mathematically, denoting $t$ the iteration that is being computed, it can be written as:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \tag{A.43}$$

Focusing now on the case it is being dealt with, the values that are wished to approximate are the $f$, $u$, and $v$.

For being able to proceed, the grid is solved by columns, assuming that the previous column ($n-1$ terms) are known values. Then, when starting a new column, the initial guess of the new column will be the duplicated values of n-1. With these guesses, the iterative procedure is applied until a proper solution is found. In each iteration, it is computed the increment that must be added to $f$, $u$, and $v$.

Rewriting A.43 for the boundary layer field variables ($f$, $u$ and $v$):

$$\begin{aligned} f_v^{t+1} &= f_j^t + \delta f_j^t \\ u_v^{t+1} &= u_j^t + \delta u_j^t \\ v_v^{t+1} &= v_j^t + \delta v_j^t \end{aligned} \tag{A.44}$$

Denoting $t+1$ the next value of that variable and $t$ the actual value of the iteration. $\delta$ indicates the computed increment that is going to be added to the field variable to obtain a more accurate solution.

Additionally, for writing the equations simpler the term $n$, denoting the column, is removed from the equations. As it is working column by column, the $n$ is going to be a fixed value for all the iterations done and it is only going to be changed once it is obtained an accurate enough solution. So, to write the equations easier, all the values imply the $n$ position despite it is not written. If the column is different, for instance, $n-1$, it is indicated.

To this method the centered-difference derivatives and the momentum equation are rewritten in the following subsections.

#### *A.4.3.1. Centered-difference derivatives*

Starting with A.33, it is expressed:

$$f_j - f_{j-1} = \frac{h_j}{2}(u_j + u_{j-1}) \tag{A.45}$$

$$u_j + u_{j-1} = 2u_{j-1/2} \tag{A.46}$$

Applying A.44 and replacing the $f$, $u$ and $v$ for the value at that iteration plus an increment, it is obtained the following expression.

$$f_j + \delta f_j - f_{j-1} - \delta f_{j-1} = \frac{h_j}{2}(u_j + \delta u_j + u_{j-1} + \delta u_{j-1}) \tag{A.47}$$

Grouping the increment terms.

$$\delta f_j - \delta f_{j-1} - \frac{h_j}{2}(\delta u_j + \delta u_{j-1}) = f_{j-1} - f_j + \frac{h_j}{2}(u_j + u_{j-1}) \tag{A.48}$$

Where the $u_j + u_{j-1}$ can be written as A.46. Leading to the final form of the expression.

$$\delta f_j - \delta f_{j-1} - \frac{h_j}{2}(\delta u_j + \delta u_{j-1}) = f_{j-1} - f_j + h_j(u_{j-1/2}) \tag{A.49}$$

The right part of the equation, the remainder part, is intended to be as close to 0 and it is going to be equal to those corresponding increments found that made the values variate. Once it is reached an iteration where the solution is accurate enough, the remaining part is approximately 0 as the increments are approximately 0. This means that no changes, or very small changes, have to be done to $f$, $v$, and $u$ values.

Some conditions are established related to the remaining part and the increments to determine if the solution obtained is a good enough approximation.

Similarly, equation A.34 can be expressed as:

$$\delta u_j - \delta u_{j-1} - \frac{h_j}{2}(\delta v_j + \delta v_{j-1}) = u_j - u_{j-1} - h_j(v_{j-1/2}) \tag{A.50}$$

### *A.4.3.2. Momentum equation*

As done in the previous section, an iterative process is also applied in A.39, replacing the unknown values with the iterative decomposition with the increments. Clarify that, the values from the previous column of the grid, the $n-1$ terms, as are already known values the iterative process must not be applied to them. Leading to the following expression:

$$\begin{aligned}\frac{1}{h_j}(b_j v_j + b_j\delta v_j - b_{j-1}v_{j-1} - b_{j-1}\delta v_{j-1}) + \alpha_1(\frac{f_j + f_{j-1} + \delta f_j + \delta f_{j-1}}{2})\\ (\frac{v_j + v_{j-1} + \delta v_j + \delta v_{j-1}}{2}) - \alpha_2(\frac{(u_j+\delta u_j)^2 + (u_{j-1}+\delta u_{j-1})^2 + 2(u_j+\delta u_j)(u_{j-1}+\delta u_{j-1})}{4})\\ +\alpha^n(v_{j-1/2}^{n-1}\frac{f_j + f_{j-1} + \delta f_j + \delta f_{j-1}}{2} - f_{j-1/2}^{n-1}\frac{v_j + v_{j-1} + \delta v_j + \delta v_{j-1}}{2}) = R_{j-1/2}^{n-1}\end{aligned} \tag{A.51}$$

Developing the expression:

$$\begin{aligned}\frac{1}{h_j}(b_j v_j - b_{j-1}v_{j-1}) + \frac{1}{h_j}(b_j\delta v_j - b_{j-1}\delta v_{j-1}) + \frac{\alpha_1}{4}(f_j v_j + f_j v_{j-1} + f_j\delta v_j + f_j\delta v_{j-1} +\\ f_{j-1}v_j + f_{j-1}v_{j-1} + f_{j-1}\delta v_j + f_{j-1}\delta v_{j-1} + \delta f_j v_j + \delta f_j v_{j-1} + \delta f_j\delta v_j + \delta f_j\delta v_{j-1} +\\ \delta f_{j-1}v_j + \delta f_{j-1}v_{j-1} + \delta f_{j-1}\delta v_j + \delta f_{j-1}\delta v_{j-1}) - \frac{\alpha_2}{4}(u_j^2 + \delta u_j^2 + 2u_j\delta u_j + u_{j-1}^2 +\\ \delta u_{j-1}^2 + 2u_{j-1}\delta u_{j-1} + 2(u_j u_{j-1} + u_j\delta u_{j-1} + \delta u_j u_{j-1} + \delta u_j\delta u_{j-1})) + \frac{\alpha^n}{2}(v_{j-1/2}^{n-1}(f_j + f_{j-1}) -\\ f_{j-1/2}^{n-1}(v_j + v_{j-1})) + \frac{\alpha^n}{2}(v_{j-1/2}^{n-1}\delta f_j + v_{j-1/2}^{n-1}\delta f_{j-1} - f_{j-1/2}^{n-1}\delta v_j - f_{j-1/2}^{n-1}\delta v_{j-1}) = R_{j-1/2}^{n-1}\end{aligned} \tag{A.52}$$

Grouping the terms and drooping the square terms of increments (to linearise) it is obtained the following expression:

$$\frac{1}{h_j}(b_j v_j - b_{j-1}v_{j-1}) + \frac{1}{h_j}(b_j \delta v_j - b_{j-1}\delta v_{j-1}) + \frac{\alpha_1}{4}(f_j(v_j + v_{j-1}) + f_{j-1}(v_j + v_{j-1})) +$$
$$\frac{\alpha_1}{4}(\delta v_j(f_j + f_{j-1}) + \delta v_{j-1}(f_j + f_{j-1}) + \delta f_j(v_j + v_{j-1}) + \delta f_{j-1}(v_j + v_{j-1})) - \alpha_2(u^2)_{j-1/2} -$$
$$\frac{\alpha_2}{2}(\delta u_j(u_j + u_{j-1}) + \delta u_{j-1}(u_{j-1} + u_j)) + \alpha^n(v^{n-1}_{j-1/2}(f_{j-1/2}) - f^{n-1}_{j-1/2}(v_{j-1/2})) +$$
$$\frac{\alpha^n}{2}(v^{n-1}_{j-1/2}\delta f_j + v^{n-1}_{j-1/2}\delta f_{j-1} - f^{n-1}_{j-1/2}\delta v_j - f^{n-1}_{j-1/2}\delta v_{j-1}) = R^{n-1}_{j-1/2} \tag{A.53}$$

Using that a term of the previous row is the actual term minus the increment, for instance, $f_{j-1} = f_j - \delta f_j$ and rearranging the terms.

$$\frac{1}{h_j}(b_j \delta v_j - b_{j-1}\delta v_{j-1}) + \frac{\alpha_1}{4}(\delta v_j(f_j + f_j - \delta f_j) + \delta v_{j-1}(f_{j-1} + \delta f_j + f_{j-1}) + \delta f_j(v_j + v_j -$$
$$\delta v_j) + \delta f_{j-1}(v_{j-1} + \delta v_j + v_{j-1})) - \frac{\alpha_2}{2}(\delta u_j(u_j + u_j - \delta u_j) + \delta u_{j-1}(u_{j-1} + u_{j-1} + \delta u_j)) +$$
$$\frac{\alpha^n}{2}(v^{n-1}_{j-1/2}\delta f_j + v^{n-1}_{j-1/2}\delta f_{j-1} - f^{n-1}_{j-1/2}\delta v_j - f^{n-1}_{j-1/2}\delta v_{j-1}) = R^{n-1}_{j-1/2} - [\frac{1}{h_j}(b_j v_j - b_{j-1}v_{j-1})$$
$$+\alpha_1(f v_{j-1/2}) - \alpha_2(u^2)_{j-1/2} + \alpha^n(v^{n-1}_{j-1/2}(f_{j-1/2}) - f^{n-1}_{j-1/2}(v_{j-1/2}))] \tag{A.54}$$

Finally, by simplifying the previous expression and dropping again the square increments terms the final expression is obtained.

$$\frac{1}{h_j}(b_j \delta v_j - b_{j-1}\delta v_{j-1}) + \frac{\alpha_1}{2}(\delta v_j f_j + \delta v_{j-1} f_{j-1} + \delta f_j v_j + \delta f_{j-1} v_{j-1}) - \alpha_2(\delta u_j u_j + \delta u_{j-1} u_{j-1}) +$$
$$\frac{\alpha^n}{2}(v^{n-1}_{j-1/2}\delta f_j + v^{n-1}_{j-1/2}\delta f_{j-1} - f^{n-1}_{j-1/2}\delta v_j - f^{n-1}_{j-1/2}\delta v_{j-1}) = R^{n-1}_{j-1/2} - [\frac{1}{h_j}(b_j v_j - b_{j-1}v_{j-1})$$
$$+\alpha_1(fv)_{j-1/2} - \alpha_2(u^2)_{j-1/2} + \alpha^n(v^{n-1}_{j-1/2}(f_{j-1/2}) - f^{n-1}_{j-1/2}(v_{j-1/2})] \tag{A.55}$$

Equation A.55 is the final expression it is going to work with, yet as it is going to work with the matrix form, common factors of the increments are going to be defined to simplify the matrix writing. Grouping the coefficients of the increments and naming them $s_1, s_2, s_3, s_4, s_5$ and $s_6$ the following expression is obtained:

$$(s_1)_j\delta v_j + (s_2)_j\delta v_{j-1} + (s_3)_j\delta f_j + (s_4)_j\delta f_{j-1} + (s_5)_j\delta u_j + (s_6)_j\delta u_{j-1} = R^{n-1}_{j-1/2} -$$
$$[\frac{1}{h_j}(b_j v_j - b_{j-1}v_{j-1}) + \alpha_1(fv)_{j-1/2} - \alpha_2(u^2)_{j-1/2} + \alpha^n(v^{n-1}_{j-1/2}f_{j-1/2} - f^{n-1}_{j-1/2}v_{j-1/2})] \tag{A.56}$$

With:

$$
\begin{aligned}
(s_1)_j &= \frac{b_j}{h_j} + \frac{\alpha_1}{2} f_j - \frac{\alpha^n}{2} f_{j-1/2}^{n-1} \\
(s_2)_j &= -\frac{b_{j-1}}{h_j} + \frac{\alpha_1}{2} f_{j-1} - \frac{\alpha^n}{2} f_{j-1/2}^{n-1} \\
(s_3)_j &= \frac{\alpha_1}{2} v_j + \frac{\alpha^n}{2} v_{j-1/2}^{n-1} \\
(s_4)_j &= \frac{\alpha_1}{2} v_{j-1} + \frac{\alpha^n}{2} v_{j-1/2}^{n-1} \\
(s_5)_j &= -\alpha_2 u_j \\
(s_6)_j &= -\alpha_2 u_{j-1}
\end{aligned}
\tag{A.57}
$$

#### *A.4.3.3. Matrices approach*

Once the equations are defined properly in the iterative way, it is intended to pose a general system representing, for a given column of the grid ( constant $\xi$ ), all the equations needed for computing the required increments of the values. For achieving this, a system of matrices is implemented.

The main matrices are going to be A, $\vec{\delta}$ and $\vec{r}$ and they are going to be related as next expressed.

$$
A\vec{\delta} = \vec{r} \tag{A.58}
$$

Representing $\vec{r}$ the right part of the equations A.49, A.50 and A.56, $\vec{\delta}$ the vector with all the increments and A the coefficients of the increments (left part of the equations A.49, A.50 and A.56).

**VECTOR** $\vec{\delta}$. The vector $\vec{\delta}$ is the vector of increments and has J+1 elements where each element is, at the same time, a vector of three values. These values are the $f$, $u$ and $v$ increments of that position, that should be added to the $f$, $u$ and $v$ values to obtain a more accurate solution.

$$
\vec{\delta} = \begin{vmatrix} \vec{\delta}_0 \\ \vec{\delta}_1 \\ \vdots \\ \vec{\delta}_j \\ \vdots \\ \vec{\delta}_J \end{vmatrix} \quad Being \quad \vec{\delta}_j = \begin{vmatrix} \delta f_j \\ \delta u_j \\ \delta v_j \end{vmatrix} \tag{A.59}
$$

**VECTOR** $\vec{r}$. The vector $\vec{r}$ is the vector with the values of the right part of the equations A.49, A.50 and A.56. It is a vector of J+1 elements and each element has 3 values corre-

sponding to the right side values of the 3 equations.

$$\vec{r} = \begin{vmatrix} \vec{r_0} \\ \vec{r_1} \\ \vdots \\ \vec{r_j} \\ \vdots \\ \vec{r_J} \end{vmatrix} \quad Being \quad \vec{r_j} = \begin{vmatrix} r_1 \\ r_2 \\ r_3 \end{vmatrix} \tag{A.60}$$

With boundary conditions:

$$\vec{r_0} = \begin{vmatrix} 0 \\ 0 \\ (r_3)_0 \end{vmatrix} \qquad \vec{r_J} = \begin{vmatrix} (r_1)_J \\ (r_2)_J \\ 0 \end{vmatrix} \tag{A.61}$$

Noting that the $r_3$ residual term is unknown at the first iteration and, as $r_3$ is function of $j-1$ terms, it is not possible to compute these values. Because of that the $r_3$ terms are computed in the next $j$ iteration and stored in $j-1$.

All this leads to the final expression of the residual terms:

$$(r_1)_j = f_{j-1} - f_j + h_j u_{j-1/2} \tag{A.62}$$

$$(r_3)_{j-1} = u_{j-1} - u_j + h_j v_{j-1/2} \tag{A.63}$$

$$\begin{aligned} (r_2)_j = R^{n-1}_{j-1/2} - [\frac{1}{h_j}(b_j v_j - b_{j-1} v_{j-1}) + \alpha_1 (fv)_{j-1/2} \\ - \alpha_2 (u^2)_{j-1/2} + \alpha^n (v^{n-1}_{j-1/2} f_{j-1/2} - f^{n-1}_{j-1/2} v_{j-1/2})] \end{aligned} \tag{A.64}$$

**MATRIX A**. The matrix A is composed of other matrices allocated in the diagonal. Moreover, as in each equation, it is found dependencies of $j$, $j+1$ and $j-1$ terms, three types of matrices 3x3 are going to be defined. Matrices $A_j$ are going to have the $j$ terms, $B_j$ matrices the $j-1$ and $C_j$ the $j+1$.

These matrices are going to be allocated in the proper order and when multiplying the general A matrix by the $\vec{\delta}$ the $j-1$ increment term is going to be multiplying the B matrix, the $j$ term the A and the $j+1$ the C.

Moreover, the first row of each of the matrices $A_j$,$B_j$,$C_j$ is the coefficients of the increments of the equation A.49, the second row of the equation A.56, and the last one of A.50 .

However, for the third row, equation A.50 can not be obtained for the current position; as has been explained with the $(r_3)_{j-1}$ term. Because of that, equation A.50 can not be solved in the current position, as for instance at the beginning it is not possible defining $\delta v_{j-1}$. To solve this problem, when being in the $j$ iteration it is going to be computed A.50 for $j-1$. Otherwise, another way to express that is that when being at iteration $j$ the equation is being computed as a function of $j+1$ values, as expressed hereafter.

$$\delta u_{j+1} - \delta u_j - \frac{h_{j+1}}{2}(\delta v_{j+1} + \delta v_j) = (r3)_j \tag{A.65}$$

Arranging the terms in matrices, taking common factor of the increments and representing each row of the matrix the three equations:

$$\delta f_j - \delta f_{j-1} - \frac{h_j}{2}(\delta u_j + \delta u_{j-1}) = (r1)_j$$

$$(s1)_j \delta v_j + (s2)_j \delta v_{j-1} + (s3)_j \delta f_j + (s4)_j \delta f_{j-1} + (s5)_j \delta u_j + (s6)_j \delta u_{j-1} = (r2)_j$$

$$\delta u_{j+1} - \delta u_j - \frac{h_{j+1}}{2}(\delta v_{j+1} + \delta v_j) = (r3)_j$$

$$A_j = \begin{array}{c} \begin{array}{ccc} \delta f_j & \delta u_j & \delta v_j \end{array} \\ \begin{vmatrix} 1 & -\frac{h_j}{2} & 0 \\ (s3)_j & (s5)_j & (s1)_j \\ 0 & -1 & -\frac{h_{j+1}}{2} \end{vmatrix} \end{array}$$

Figure A.2: Aj matrix terms

$$\delta f_j - \delta f_{j-1} - \frac{h_j}{2}(\delta u_j + \delta u_{j-1}) = (r1)_j$$

$$(s1)_j \delta v_j + (s2)_j \delta v_{j-1} + (s3)_j \delta f_j + (s4)_j \delta f_{j-1} + (s5)_j \delta u_j + (s6)_j \delta u_{j-1} = (r2)_j$$

$$\delta u_{j+1} - \delta u_j - \frac{h_{j+1}}{2}(\delta v_{j+1} + \delta v_j) = (r3)_j$$

$$B_j = \begin{array}{c} \begin{array}{ccc} \delta f_{j-1} & \delta u_{j-1} & \delta v_{j-1} \end{array} \\ \begin{vmatrix} -1 & -\frac{h_j}{2} & 0 \\ (s4)_j & (s6)_j & (s2)_j \\ 0 & 0 & 0 \end{vmatrix} \end{array}$$

Figure A.3: Bj matrix terms

$$\delta f_j - \delta f_{j-1} - \frac{h_j}{2}(\delta u_j + \delta u_{j-1}) = (r1)_j$$

$$(s1)_j\delta v_j + (s2)_j\delta v_{j-1} + (s3)_j\delta f_j + (s4)_j\delta f_{j-1} + (s5)_j\delta u_j + (s6)_j\delta u_{j-1} = (r2)_j$$

$$\delta u_{j+1} - \delta u_j - \frac{h_{j+1}}{2}(\delta v_{j+1} + \delta v_j) = (r3)_j$$

$$C_j = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & -\frac{h_{j+1}}{2} \end{vmatrix}$$

$\delta v_{j+1}$

Figure A.4: Cj matrix terms

Finally, expressing the matrices fully expanded:

$$A = \begin{vmatrix} A_0 & C_0 & & & & & & & \\ B_1 & A_1 & C_1 & & & & & & \\ & \vdots & \vdots & \vdots & & & & & \\ & & & B_j & A_j & C_j & & & \\ & & & & \vdots & \vdots & \vdots & & \\ & & & & & B_{J-1} & A_{J-1} & C_{J-1} & \\ & & & & & & B_J & A_J & \end{vmatrix} \tag{A.66}$$

$$Being \quad A_j = \begin{vmatrix} 1 & -\frac{h_j}{2} & 0 \\ (s_3)_j & (s_5)_j & (s_1)_j \\ 0 & -1 & -\frac{h_{j+1}}{2} \end{vmatrix} \quad B_j = \begin{vmatrix} -1 & -\frac{h_j}{2} & 0 \\ (s_4)_j & (s_6)_j & (s_2)_j \\ 0 & 0 & 0 \end{vmatrix} \quad C_j = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & -\frac{h_{j+1}}{2} \end{vmatrix} \tag{A.67}$$

Being $A_0$ and $A_J$ the boundary conditions.

$$A_0 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & -\frac{h_1}{2} \end{vmatrix} \quad A_J = \begin{vmatrix} 1 & -\frac{h_J}{2} & 0 \\ (s_3)_J & (s_5)_J & (s_1)_J \\ 0 & 1 & 0 \end{vmatrix} \tag{A.68}$$

### A.4.4. Gauss elimination method

Finally, to solve the system and obtain the values of increments, it is applied the Gauss method. In this way, it is introduced a 0 in the last equation, enables solving directly

the value of the last increment. Once the last increment is obtained it is substituted in the previous equation and the penultimate increment is known and successively with the following equations.
Expressing the proceed mathematically, for instance for a small system of just 3 divisions of $\eta$, it proceeds as follows.

$$\begin{aligned} A_0\vec{\delta_0} + C_0\vec{\delta_1} &= \vec{r_0} \\ B_1\vec{\delta_0} + A_1\vec{\delta_1} + C_1\vec{\delta_2} &= \vec{r_1} \\ B_2\vec{\delta_1} + A_2\vec{\delta_2} &= \vec{r_2} \end{aligned} \tag{A.69}$$

Multiplying the first equation by $-B_1A_0^{-1}$:

$$\begin{aligned} -B_1A_0^{-1}A_0\vec{\delta_0} - B_1A_0^{-1}C_0\vec{\delta_1} &= -B_1A_0^{-1}\vec{r_0} \\ -B_1\vec{\delta_0} - B_1A_0^{-1}C_0\vec{\delta_1} &= -B_1A_0^{-1}\vec{r_0} \end{aligned} \tag{A.70}$$

Then, summing the second equation of the system with the equation A.70:

$$\begin{aligned} -B_1\vec{\delta_0} - B_1A_0^{-1}C_0\vec{\delta_1} + B_1\vec{\delta_0} + A_1\vec{\delta_1} + C_1\vec{\delta_2} &= \vec{r_1} - B_1A_0^{-1}\vec{r_0} \\ (A_1 - B_1A_0^{-1}C_0)\vec{\delta_1} + C_1\vec{\delta_2} &= \vec{r_1} - B_1A_0^{-1}\vec{r_0} \end{aligned} \tag{A.71}$$

Finally, expressing $A_1 - B_1A_0^{-1}C_0$ as $A_{01}$ and $\vec{r_1} - B_1A_0^{-1}\vec{r_0}$ as $r_{01}$.

$$A_{01}\vec{\delta_1} + C_1\vec{\delta_2} = r_{01} \tag{A.72}$$

Now, it is repeated the process with the third equation of A.69 and the equation obtained in the previous step A.72. First, multiplying A.72 by $-B_2A_{01}^{-1}$ and then summing the resulting equation with the third equation of A.69. Doing that it is obtained:

$$(A_2 - B_2A_{01}^{-1}C_1)\vec{\delta_2} = r_3 - B_2A_{01}^{-1}r_{01} \tag{A.73}$$

As seen, a zero has been introduced in the last equation and now it is possible to isolate the increment vector from A.73. Once known $\delta_2$ it can be isolated from A.72 the $\delta_1$ and finally, from the first equation of A.69 the $\delta_0$ is obtained.

Extrapolating this procedure to a matrix of J elements it is obtained all the increments vectors that are going to be added to the $f$, $v$ and $u$ values to make the next iteration.

# APPENDIX B. RE-COMPRESSION WITH THE INTEGRAL METHOD

Applying the integral method assuming that flat plate conditions can be used as an acceptable approximation when dealing with the presence of an external flow characteristic of a re-compression. The flow can be modeled as:

$$u_e(x) = U_0(1 - \frac{x}{L})^{\alpha} \tag{B.1}$$

Or equivalently, in the dimensionless way:

$$\bar{u_e}(\bar{x}) = (1 - \bar{x})^{\alpha} \tag{B.2}$$

Where the $\alpha$ parameter determines the curvature of the profile, as observed in the following image.

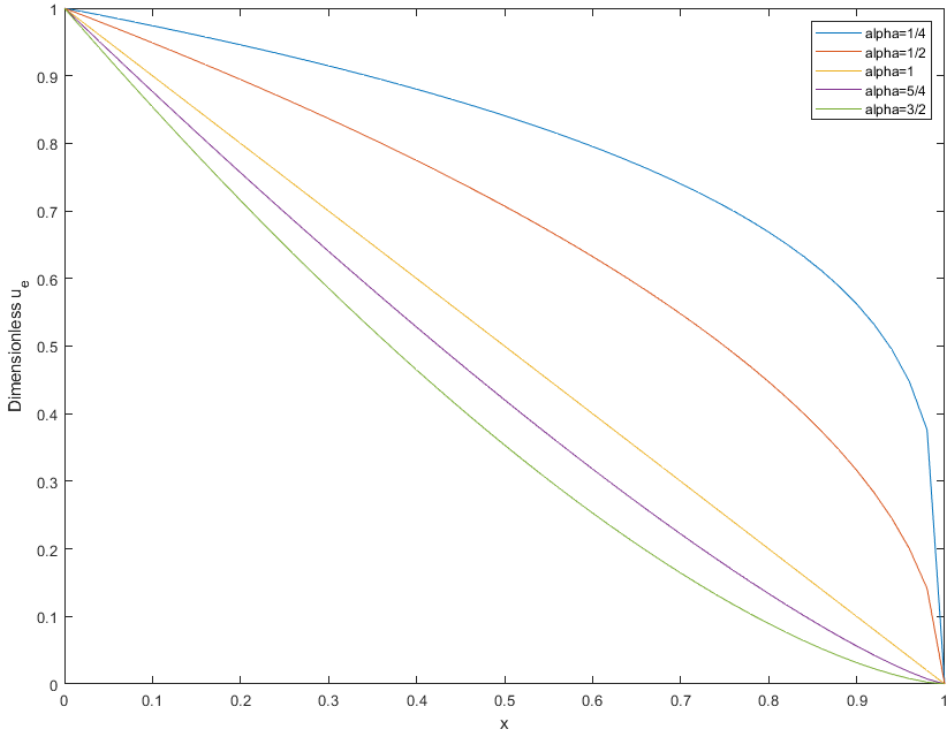



Figure B.1: Velocity profile of a re-compression case

On the other, the solution of the integral method approximation can be expressed as:

$$(\theta u_e^{H+2})_{x_1}^2 = (\theta u_e^{H+2})_{x_0}^2 + 2b \int_{x_0}^{x_1} \frac{u_e^{2(H+2)}}{u_e/\nu} dx \tag{B.3}$$

In order to compute the separation point of the flow, first, it is going to find the expression of $\bar{\theta}$ as function of $\bar{x}$. For doing this, an initial change of variable is going to be applied to B.3, defining $\frac{\beta}{\alpha} = (m_0+1)(H+2) - m_0$:

$$\theta_1^{m_0+1} u_e^{\frac{\beta}{\alpha}+m_0} = \theta_0^{m_0+1} u_0^{\frac{\beta}{\alpha}+m_0} + (m_0+1) b \nu^{m_0} \int_{x_0}^{x_1} u_e^{\frac{\beta}{\alpha}} dx \tag{B.4}$$

Then, to make the variables dimensionless it is multiplied by $\frac{1}{L^{m_0+1}} \frac{1}{U_0^{\frac{\beta}{\alpha}+m_0}}$ and the $\theta_0$ term is isolated.

$$\bar{\theta}^{m_0+1} = \frac{\bar{\theta_0}^{m_0+1}}{\bar{u}^{\frac{\beta}{\alpha}+m_0}} + \frac{(m_0+1)b}{Re_L^{m_0} \bar{u}^{\frac{\beta}{\alpha}+m_0}} \int_0^{\bar{x}} \bar{u}^{\beta/\alpha} d\bar{x} \tag{B.5}$$

Solving the integral:

$$\int_0^{\bar{x}} \bar{u}^{\beta/\alpha} d\bar{x} = \int_0^{\bar{x}} (1-\bar{x})^{\beta} d\bar{x} = \frac{-1}{\beta+1} ((1-\bar{x})^{\beta+1} - 1) \tag{B.6}$$

Finally, putting together both parts and substituting the velocity profile expression, the following expression is obtained.

$$\bar{\theta} = \left[ \frac{\bar{\theta_0}^{m_0+1}}{(1-\bar{x})^{\beta+\alpha m_0}} + \frac{(m_0+1)b}{Re_L^{m_0}(1-\bar{x})^{\beta+\alpha m_0}} \frac{1-(1-\bar{x})^{\beta+1}}{\beta+1} \right]^{\frac{1}{m_0+1}} \tag{B.7}$$

From this expression it can be computed the separation point, considering for a laminar case B.8 (or the 0.09 from empirical correlations, according to Thwaites) and for the turbulent B.9.

$$-\frac{\theta^2}{\nu} \frac{du_e}{dx} = C = 0.068 \tag{B.8}$$

$$-\frac{\theta^2}{\nu} \frac{du_e}{dx} = C = 0.0035 \tag{B.9}$$

And substituting for laminar $m_0 = 1, b = 0.2205, H = 2.591$ and for turbulent $m_0 = 0.2, b = 0.0.0086, H = 1.4$ leading to following expressions for computing the separation point, neglecting the $\bar{\theta_0}$ thickness.

**LAMINAR**

$$\bar{x_s} = 1 - \left[ \frac{0.441\alpha}{C_{lam}(1+8.182\alpha) + 0.441\alpha} \right]^{\frac{1}{1+8.182\alpha}} \tag{B.10}$$

**TURBULENT**

$$\frac{0.02212\alpha}{Re_L^{0.2}(1+3.88\alpha)} \frac{(1-(1-\bar{x_s})^{0.83+3.23\alpha})}{(1-\bar{x_s})^{3.4\alpha+1}} = C_{turb} \tag{B.11}$$

For the turbulent expression, a numerical solution is needed in order to obtain the value of the separation point.

# APPENDIX C. CODE

Hereafter, are attached the functions used in the code, examples of the main program that has been used, as well as some exported data from Xfoil that is used on the simulations.

The files can be found in the following order:

- Main script used for flat plate simulations.
- Main script introducing a stagnation point.
- Main script incorporating the hyperbolic tangent.
- Main script with a laminar re-compression case.
- Main script with a turbulent re-compression case.
- Main script importing laminar data from Xfoil.
- Main script with turbulent adapted cases importing data from Xfoil.
- Create grid function.
- Create grid function for Xfoil data.
- Filling A and r matrices function.
- Applying Gauss Method function.
- Computing increments function.
- Falkner Skan function for computing the velocity distribution at x=0.
- Turbulence model function.
- Import M vector function adapted for Xfoil document text.
- Compute displacement thickness function.
- Compute momentum thickness function.
- Compute residual sum function.
- Compute integral from turbulence model function.
- Find intersecting point function.
- Xfoil data of NACA 0012 airfoil with a Reynolds number of 60,000 with an angle of attack of 0 degrees.
- Xfoil data of NACA 0012 airfoil with a Reynolds number of 60,000 with an angle of attack of 2 degrees.
- Coordinates of the SD7003 airfoil.
- Xfoil data of SD7003 airfoil with a Reynolds number of 60,000 with an angle of attack of 2 degrees.
- Inviscid Xfoil data of NACA 0012 airfoil with an angle of attack of 0 degrees.

```
%% MAIN CODE ADAPTED FOR A FLAT PLATE CASE %%
close all
clear all

%% DEFINING PARAMETERS %%

K=1; % homogeneity of the grid
if K~=1
    eta_e=20;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1);
else
    J=1000;
    h1=0.01;
end
N=200;

u0=5; % constant one-way flux
viscosity=15*10^-6/1.225; % viscosity of the fluid

%% Falkner-skan values %%
tolerancia=1e-8;

% flat plate
m=0;
Cf_2_Re=0.33206;
m_vector=zeros(1,N+1); % mantaining a flat plate (m=0)
% stagnation point
% m=1;
% Cf_2_Re=1.23;

%% Starting the grid %%

positions=CreateGrid(N,J,h1,K); % CELL array with xi eta hj_next
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia); % velocity distribution at x=0
Values={}; % At each position it will have f,u,v,b
for j=1:1:J+1
      Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % saving the values at x=0
end

%% MAIN LOOP %%
divergeix=0;
for n=2:1:N+1
    if divergeix==1
        break
    end
    % first of all, duplicate the values of the previous column to obtain
    % an initial guess of the next column to star working with

    for j=1:1:J+1
        Values{j,n}=Values{j,n-1};
    end
         convergeix= 0; % to know if it is needed continue iterating
    sum_r2_anterior=10e10; % starting with a disproportionately large value to ensure↙
that the computed will be smaller


    while convergeix==0
        % Group of 3 solving functions
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %↙
```

```
fill cell array A and r
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % Gauss method ->multiply by  -B1*inv(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing increments

        [sq_sum_r2]=SumaResidu(r);  % compute the sqrt sum of r2 to check if the code diverges
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it is diverging
            sum_r2_anterior=sq_sum_r2;
        else
            Values(:,n)=[]; % the code is diverging so the computed values are wrong
            divergeix=1;
            break
        end

        % if it is not reached a separation point, it is check if the
        % solution is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end
        % if it is needed, the values are actualized
        if convergeix == 0
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
            end
        end
    end
end

%% Plots %%
[a,sp]=size(Values); % dimensions of the cell array

eta_PLOT=zeros(1,a); % eta values
xi_PLOT=zeros(1,sp); % xi values

%% INCIDENT VELOCITY DISTIBUTION %%

fprime1=zeros(1,a); % f values at x=0
fprime2=zeros(1,a); % f' values at x=0
fprime3=zeros(1,a); % f'' values at x=0

for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(1);
    fprime2(1,j)=Values{j,1}(2);
    fprime3(1,j)=Values{j,1}(3);
end

figure
hold on
plot(fprime2,eta_PLOT)
plot(fprime3,eta_PLOT)
ylabel('η')
legend('f´', 'f´´', "Location","best")
title ('INCIDENT VELOCITY PROFILE')
grid
```

```
figure
plot(fprime1,eta_PLOT)
ylabel('η')
title ('INCIDENT VELOCITY PROFILE')
grid



%% COLOR PLOTS %%
matrix_f_p=zeros(a,sp);
matrix_f=zeros(a,sp);
matrix_f_pp=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
        matrix_f(j,i)=Values{j,i}(1);
        matrix_f_pp(j,i)=Values{j,i}(3);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end


figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´] ')
xlabel('x')
ylabel('η')

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f] ')
xlabel('x')
ylabel('η')

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_pp,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´´] ')
xlabel('x')
ylabel('η')

%% COMPUTING CHARECTERISTIC PROPERTIES %%
Rx_values=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u0*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u0)^2;
end
```

```
[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

blasius_cf=0.664./Rx_values.^0.5;
blasius_displacement=1.721.*xi_PLOT./Rx_values.^0.5;
blasius_momentum=0.664.*xi_PLOT./Rx_values.^0.5;

%% DISPLACEMENT AND MOMENTUM THICKNESS %%
figure
hold on
plot(xi_PLOT,displacement,'x', 'MarkerSize', 3)
plot(xi_PLOT,theta,'x', 'MarkerSize', 3)
plot(xi_PLOT,blasius_displacement)
plot(xi_PLOT,blasius_momentum)
xlabel('x')
ylabel('δ^* //  θ')
grid
legend ('δ^*','θ','Blasius δ^*','Blasius θ','Location',"best")
%% CF %%
figure
semilogy(xi_PLOT,cf,'x', 'MarkerSize',3)
hold on
semilogy(xi_PLOT,blasius_cf)
ylabel ('Cf')
xlabel('x')
legend('Computed','Blasius solution')
grid
```

```
%% MAIN CODE ADAPTED FOR A FLAT PLATE CASE %%
close all
clear all
format long

%% DEFINING PARAMETERS %%

K=1;% homogeneity of the grid
if K~=1
    eta_e=7;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1);
else
    J=1000;
    h1=0.01;
end

N=500;

u0=1; % outer velocity distribution
syms X
u_e(X)=u0*X;
D(X)=diff(u_e,X);

viscosity=15*10^-6/1.225;% viscosity of the fluid

%% Falkner-skan values %%
tolerancia=1e-8;

% flat plate
% m=0;
% Cf_2_Re=0.33206;
% stagnation point
m=1;
Cf_2_Re=1.23;

%% Starting the grid %%
positions=CreateGrid(N,J,h1,K); % CELL array with xi eta hj_next
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia); % velocity distribution at x=0
Values={}; % At each position it will have f,u,v,b
for j=1:1:J+1
      Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % saving the values at x=0
end

% Defining the dimensionless pressure gradient (m) €€

m_vector=zeros(1,N+1);
for n=1:1:N+1
    m_vector(1,n)=positions{1,n}(1)./u_e(positions{1,n}(1)).*D(positions{1,n}(1));
end
m_vector(1,1)=1;


for i=201:1:length(m_vector)
    m_vector(1,i)=0;
end

%% MAIN LOOP %%
divergeix=0;
for n=2:1:N+1
```

```
    if divergeix==1
        break
    end
    % first of all, duplicate the values of the previous column to obtain
    % an initial guess of the next column to star working with

    for j=1:1:J+1
        Values{j,n}=Values{j,n-1};
    end

    convergeix= 0; % to know if it is needed continue iterating
    sum_r2_anterior=10e10;% starting with a disproportionately large value to ensure that the computed will be smaller

    while convergeix==0
         % Group of 3 solving functions
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); % fill cell array A and r
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % Gauss method ->multiply by  -B1*inv(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing increments

        [sq_sum_r2]=SumaResidu(r);  % compute the sqrt sum of r2 to check if the code diverges
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it is diverging
            sum_r2_anterior=sq_sum_r2;
        else
            Values(:,n)=[]; % the code is diverging so the computed values are wrong
            divergeix=1;
            break
        end

        % if it is not reached a separation point, it is check if the
        % solution is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end

        % if it is needed, the values are actualized
        if convergeix == 0 % actualitzem els valors
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
            end
        end
    end
end

%% Plots %%
[a,sp]=size(Values); % dimensions of the cell array

eta_PLOT=zeros(1,a); % eta values
xi_PLOT=zeros(1,sp); % xi values

%% INCIDENT VELOCITY DISTIBUTION %%

fprime1=zeros(1,a); % f values at x=0
```

```
for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(2);
end

figure
plot(eta_PLOT,fprime1,'.')
xlabel('eta')
ylabel('f prime')
title ('Velocity profile')

%% COLOR PLOTS %%
matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end


figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none') % mes contorns
colorbar
title('[f´]')
xlabel('x')
ylabel('η')

%% COMPUTING CHARECTERISTIC PROPERTIES %%
Rx_values=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);
bl_t=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u_e(positions{1,n}(1))*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u_e(positions{1,n}(1)))^2;
    for j=1:1:a
        if Values{j,n}(2)>=0.99
            bl_t(1,n)=positions{j,n}(2)*sqrt(viscosity*xi_PLOT(1,n)/u_e(positions{1,n}↙
(1)));
            break
        end
    end
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

displacement_tabulated=0.6479*sqrt(viscosity/u0);
theta_tabulated=0.2923*sqrt(viscosity/u0);

bl_t(1,1)=bl_t(1,2);
```

```
%% BOUNDARY LAYER THICKNESS %%
figure
hold on
plot(xi_PLOT,bl_t)
grid
title('Boundary Layer thickness')
xlabel('x')
ylabel('y')

%% DISPLACEMENT AND MOMENTUM THICKNESS %%
figure
hold on
plot(xi_PLOT,displacement)
plot(xi_PLOT,theta)
xlabel('x')
ylabel('δ^* //  θ')
grid
legend ('δ^*','θ','Location',"best")

%% CF %%
figure
plot(xi_PLOT,cf)
ylabel ('Cf')
xlabel('x')
ylabel('Cf')
```

```
%% MAIN CODE ADAPTED FOR THE HYPERBOLIC TANGENT %%
close all
clear all

%% DEFINING PARAMETERS %%

K=1; % homogeneity of the grid
if K~=1
    eta_e=20;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1);
else
    J=1000;
    h1=0.01;
end
N=200;

u0=5; % constant one-way flux
viscosity=15*10^-6/1.225; % viscosity of the fluid

%% Starting the grid %%

positions=CreateGrid(N,J,h1,K); % CELL array with xi eta hj_next

Values={}; % At each position we will have f,u,v,b
for j=1:1:J+1
    %     k=0.38; % Flat plate
    %     k=0.3;  % Near separation
    %     k=0.68; % Stagnation point
    k=0.68;
    f_prime= tanh(k*positions{j,1}(2));
    if j==1
        f_pp=0;
        f=0;
    else
        f_pp=(f_prime-Values{j-1,1}(2))/(positions{j,1}(2)-positions{j-1,1}(2));
        f=(Values{j-1,1}(1)+f_prime*(positions{j,1}(2)-positions{j-1,1}(2)));
    end

    Values{j,1}=[f, f_prime, f_pp, 1];
end

m_vector=zeros(1,N+1); % mantaining a flat plate (m=0)
m_vector(1,1)=1;
%% MAIN LOOP %%
divergeix=0;
for n=2:1:N+1
    if divergeix==1
        break
    end
    % first of all, duplicate the values of the previous column to obtain
    % an initial guess of the next column to star working with

    for j=1:1:J+1
        Values{j,n}=Values{j,n-1};
    end

    convergeix= 0; % to know if it is needed continue iterating
    sum_r2_anterior=10e10; % starting with a disproportionately large value to ensure↙
that the computed will be smaller
```

```
    while convergeix==0
        % Group of 3 solving functions
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %↙
fill cell array A and r
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % Gauss method ->multiply by  -B1*inv↙
(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing↙
increments

        [sq_sum_r2]=SumaResidu(r);  % compute the sqrt sum of r2 to check if the code↙
diverges
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it is diverging
            sum_r2_anterior=sq_sum_r2;
        else
            Values(:,n)=[]; % the code is diverging so the computed values are wrong
            divergeix=1;
            break
        end

        % if it is not reached a separation point, it is check if the
        % solution is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end
        % if it is needed, the values are actualized
        if convergeix == 0
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
            end
        end
    end
end

%% Plots %%
[a,sp]=size(Values); % dimensions of the cell array

eta_PLOT=zeros(1,a); % eta values
xi_PLOT=zeros(1,sp); % xi values

%% INCIDENT VELOCITY DISTIBUTION %%

fprime1=zeros(1,a); % f values at x=0
fprime2=zeros(1,a); % f' values at x=0
fprime3=zeros(1,a); % f'' values at x=0

for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(1);
    fprime2(1,j)=Values{j,1}(2);
    fprime3(1,j)=Values{j,1}(3);
end

figure
hold on
plot(fprime2,eta_PLOT)
plot(fprime3,eta_PLOT)
ylabel('η')
```

```
legend('f´', 'f´´', "Location","best")
title ('INCIDENT VELOCITY PROFILE')
grid

figure
plot(fprime1,eta_PLOT)
ylabel('η')
title ('INCIDENT VELOCITY PROFILE')
grid



%% COLOR PLOTS %%
matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end


figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´] ')
xlabel('x')
ylabel('η')

%% COMPUTING CHARECTERISTIC PROPERTIES %%
Rx_values=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u0*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u0)^2;
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

blasius_cf=0.664./Rx_values.^0.5;
blasius_displacement=1.721.*xi_PLOT./Rx_values.^0.5;
blasius_momentum=0.664.*xi_PLOT./Rx_values.^0.5;

%% DISPLACEMENT AND MOMENTUM THICKNESS %%
figure
hold on
plot(xi_PLOT,displacement,'x', 'MarkerSize', 3)
plot(xi_PLOT,theta,'x', 'MarkerSize', 3)
plot(xi_PLOT,blasius_displacement)
plot(xi_PLOT,blasius_momentum)
```

```
xlabel('x')
ylabel('δ^* //  θ')
grid
legend ('δ^*','θ','Blasius δ^*','Blasius θ','Location',"best")
%% CF %%
figure
plot(xi_PLOT,cf)
hold on
plot(xi_PLOT,blasius_cf)
ylabel ('Cf')
xlabel('x')
legend('Computed','Blasius solution')
grid
```

```
%% MAIN CODE ADAPTED FOR A LAMINAR RE-COMPRESSION %%
close all
clear all
format long

%% DEFINING PARAMETERS %%

K=1; % homogeneity of the grid
if K~=1
    eta_e=7;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1)+5;
else
    J=1000;
    h1=0.01;
end

N=1000;

viscosity=15*10^-6/1.225;% viscosity of the fluid

%% Falkner-skan values %%
tolerancia=1e-8;

% flat plate
m=0;
Cf_2_Re=0.33206;

% stagnation point
% m=1;
% Cf_2_Re=1.23;

%% Starting the grid %%

positions=CreateGrid(N,J,h1,K); % CELL array with xi eta hj_next
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia); % velocity distribution at x=0
Values={}; % At each position it will have f,u,v,b
for j=1:1:J+1
    Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % saving the values at x=0
end

%% Defining the outer velocity %%

alpha=1/2;
u0=100;
syms X
u_e(X)=u0*(1-X)^alpha;
D(X)=diff(u_e,X);

m_vector=zeros(1,N);
for n=1:1:N
    m_vector(1,n)=positions{1,n}(1)./u_e(positions{1,n}(1)).*D(positions{1,n}(1));
end

%% MAIN LOOP %%
divergeix=0;
for n=2:1:N
    if divergeix==1
        break
    end
```

```
    % first of all, duplicate the values of the previous column to obtain
    % an initial guess of the next column to star working with

    for j=1:1:J+1
        Values{j,n}=Values{j,n-1};
    end

    convergeix= 0; % to know if it is needed continue iterating
    sum_r2_anterior=10e10; % starting with a disproportionately large value to ensure
that the computed will be smaller

    while convergeix==0
        % Group of 3 solving functions
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %
fill cell array A and r
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % Gauss method ->multiply by  -B1*inv
(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing
increments

        [sq_sum_r2]=SumaResidu(r);  % compute the sqrt sum of r2 to check if the code
diverges
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it is diverging
            sum_r2_anterior=sq_sum_r2;
        else
            Values(:,n)=[]; % the code is diverging so the computed values are wrong
            divergeix=1;
            break
        end

        % if it is not reached a separation point, it is check if the
        % solution is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end

        % if it is needed, the values are actualized
        if convergeix == 0 % actualitzem els valors
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
            end
        end
    end
end

%% Plots %%
[a,sp]=size(Values); % dimensions of the cell array

eta_PLOT=zeros(1,a); % eta values
xi_PLOT=zeros(1,sp); % xi values

%% INCIDENT VELOCITY DISTIBUTION %%

fprime1=zeros(1,a); % f' values at x=0

for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(2);
```

```
end

figure
plot(eta_PLOT,fprime1,'.')
xlabel('η')
grid
title ('Velocity profile')

%% COLOR PLOTS %%
matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´] ')
xlabel('x')
ylabel('η')

%% COMPUTING CHARECTERISTIC PROPERTIES %%
Rx_values=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);
bl_t=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u_e(positions{1,n}(1))*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u_e(positions{1,n}(1)))^2;
    for j=1:1:a
        if Values{j,n}(2)>=0.99
            bl_t(1,n)=positions{j,n}(2)*sqrt(viscosity*xi_PLOT(1,n)/u_e(positions{1,n}↙
(1)));
            break
        end
    end
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

displacement_tabulated=0.6479*sqrt(viscosity/u0);
theta_tabulated=0.2923*sqrt(viscosity/u0);

%% BOUNDARY LAYER THICKNESS %%
figure
hold on
plot(xi_PLOT,bl_t)
title('Boudary layer thickness')
```

```
xlabel('x')
ylabel('y')

%% DISPLACEMENT AND MOMENTUM THICKNESS %%
figure
hold on
plot(xi_PLOT,displacement,'x', 'MarkerSize', 3)
plot(xi_PLOT,theta,'x', 'MarkerSize', 3)
grid
legend ('δ^*','θ','Location',"best")

%% CF %%

figure
semilogy(xi_PLOT,cf,'x', 'MarkerSize',3)
ylabel ('Cf')
xlabel('x')
legend('Computed Cf')
grid
```

```
%% MAIN CODE ADAPTED FOR A TURBULENT RE-COMPRESSION %%
close all
clear all
format long

%% DEFINING PARAMETERS %%

K=1; % homogeneity of the grid
if K~=1
    eta_e=7;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1)+5;
else
    J=2000;
    h1=0.01;
end

N=1000;

viscosity=15*10^-6/1.225; % viscosity of the fluid

%% Falkner-skan values %%
tolerancia=1e-8;

% flat plate
m=0;
Cf_2_Re=0.33206;

% stagnation point
% m=1;
% Cf_2_Re=1.23;

%% Starting the grid %%
positions=CreateGrid(N,J,h1,K); % CELL array with xi eta hj_next
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia); % velocity distribution at x=0
Values={}; % At each position it will have f,u,v,b
for j=1:1:J+1
    Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % saving the values at x=0
end

%% Defining the outer velocity %%

alpha=5/4;
u0=100;
syms X
u_e(X)=u0*(1-X)^alpha;
D(X)=diff(u_e,X);

m_vector=zeros(1,N+1);
for n=1:1:N
    m_vector(1,n)=positions{1,n}(1)./u_e(positions{1,n}(1)).*D(positions{1,n}(1));
end
m_vector(1,N+1)=m_vector(1,N);

%% Transition to turbulence parameters and vectors %%

R_tr=5.5e5; % forced transition
xtr_primer=1;
R_trSolved=false;
found_tr=false;
```

```
Rx_values(1,1)=0;
RT_values(1,1)=nan;

%% MAIN LOOP %%
divergeix=0;
for n=2:1:N
    if divergeix==1
        break
    end
    % first of all, duplicate the values of the previous column to obtain
    % an initial guess of the next column to star working with

    for j=1:1:J+1
        Values{j,n}=Values{j,n-1};
    end

    convergeix= 0; % to know if it is needed continue iterating
    sum_r2_anterior=10e10; % starting with a disproportionately large value to ensure↙
that the computed will be smaller



    Rx_values(1,n)=positions{1,n}(1)*double(u_e(positions{1,n}(1)))/viscosity; % local↙
Reynolds

    if Rx_values(1,n)<R_tr % transition point has not been reached yet
        RX2=Rx_values(1,n);
        mom=ComputeMomentum(n,Values,positions,Rx_values);
        RT_values(1,n)=u_e(positions{1,n}(1))*mom(1,n)/viscosity;

        % TRANSITION POINT %

        if (n>2) && (R_trSolved==false)
            RX1=Rx_values(1,n-1);

            RT_C1 =RT_values(1,n-1);
            RT_C2=RT_values(1,n);

            RT_T1=1.174*(1+22400/RX1)*RX1^0.46;
            RT_T2=1.174*(1+22400/RX2)*RX2^0.46;


            Rx_interval=[RX1 RX2];
            Rt_computed=[RT_C1 RT_C2];
            Rt_tabulated=[RT_T1 RT_T2];

            [intercepting_Point, found_tr]=intersectionPoint(Rx_interval, Rt_computed,↙
Rx_interval, Rt_tabulated);

            if found_tr==true
                R_tr=RX2;

                R_trSolved=true;
            end
        end
    end

    if Rx_values(1,n)>=R_tr % there is turbulence
        if xtr_primer==1
```

```
            % if it is the first time, we save the values
            x_tr=positions{j,n}(1);
            u_e_x_tr=double(u_e(x_tr));
        end
        xtr_primer=0;

        Rx_values=zeros(1,n);
        for w=1:1:n %
            Rx_values(1,w)=(u_e(positions{1,w}(1))*(positions{1,w}(1))/viscosity);
        end

        mom=ComputeMomentum(n,Values,positions,Rx_values);
        [Values,em_values]=TurbulenceInfluence(viscosity,Values,positions,n,m,x_tr,↙
alpha,double(u_e(positions{j,n}(1))),u_e_x_tr,mom,u0,0,0);
    end

    while convergeix==0
        % Group of 3 solving functions
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %↙
fill cell array A and r
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % Gauss method ->multiply by  -B1*inv↙
(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing↙
increments

        [sq_sum_r2]=SumaResidu(r);  % compute the sqrt sum of r2 to check if the code↙
diverges
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it is diverging
            sum_r2_anterior=sq_sum_r2;
        else
            Values(:,n)=[]; % the code is diverging so the computed values are wrong
            divergeix=1;
            break
        end

        % if it is not reached a separation point, it is check if the
        % solution is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end

        % if it is needed, the values are actualized
        if convergeix == 0 % actualitzem els valors
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
            end
        end
    end
end

%% Plots %%
[a,sp]=size(Values); % dimensions of the cell array

eta_PLOT=zeros(1,a); % eta values
xi_PLOT=zeros(1,sp); % xi values

%% INCIDENT VELOCITY DISTIBUTION %%
fprime1=zeros(1,a);  % f' values at x=0
```

```
for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(2);
end

figure
plot(eta_PLOT,fprime1,'.')
xlabel('η')
title ('Velocity profile')

%% COLOR PLOTS %%

matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´]')
xlabel('x')
ylabel('η')
%% COMPUTING CHARECTERISTIC PROPERTIES %%

Rx_values=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);
bl_t=zeros(1,sp);


for n=1:1:sp
    Rx_values(1,n)=(u_e(positions{1,n}(1))*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u_e(positions{1,n}(1)))^2;
    for j=1:1:a
        if Values{j,n}(2)>=0.99
            bl_t(1,n)=positions{j,n}(2)*sqrt(viscosity*xi_PLOT(1,n)/u_e(positions{1,n}↙
(1)));
            break
        end
    end
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

displacement_tabulated=0.6479*sqrt(viscosity/u0);
theta_tabulated=0.2923*sqrt(viscosity/u0);
```

```
%% BOUNDARY LAYER THICKNESS %%
figure
plot(xi_PLOT,bl_t)
title('BOUNDARY LAYER THICKNESS')
xlabel('x')
ylabel('y')
grid

%% DISPLACEMENT AND MOMENTUM THICKNESS %%
figure
hold on
plot(xi_PLOT,displacement,'x', 'MarkerSize', 2)
plot(xi_PLOT,theta,'x', 'MarkerSize', 2)
grid
legend ('δ^*','θ','Location',"best")

%% CF %%
figure
plot(xi_PLOT,cf,'x', 'MarkerSize',3)
ylabel ('Cf')
xlabel('x')
grid
%% TRANSITION CRITERION %%

Rtheta_values=zeros(1,sp);
rtT=zeros(1,sp);
for n=1:1:sp
    Rtheta_values(1,n)=u_e(positions{1,n}(1))*theta(1,n)/viscosity;
    rtT(1,n)=1.174*(1+22400/Rx_values(1,n))*Rx_values(1,n)^0.46;
end

figure
hold on
plot(Rx_values,Rtheta_values)
plot(Rx_values,rtT)
title('Transition criterion')
legend ('Computed','Tabulated')
grid
```

```
close all
clear all

%% GRID PARAMETERS DEFINITION %%
K=1; % defining homogeneous grid or not
if K~=1 % not homogeneous
    eta_e=7;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1)+5;
else % homogeneous
    J=1000;
    h1=0.01;
end

%% DEFINING EXTERNAL VELOCITY PROFILE %%

%%%%%%% If defining the external velocity from an Xfoil file %%%%%%%
[m_vector_in,m_vector_out,x_arc_out,x_arc_in,u_e_out,u_e_in] = ...
    ImportMvector('60000SD7003.txt'); % Importing from an Xfoil file

% Comment and uncomment the desired set of values from the intrados or
% extrados:

% EXTRADOS VALUES
m_vector=m_vector_out;
xi_PLOT=x_arc_out;
u_e=u_e_out;

% INTRADOS VALUES
% m_vector=m_vector_in;
% xi_PLOT=x_arc_in;
% u_e=-u_e_in;

N= length(xi_PLOT); % overwriting the value

%% DEFINING THE INCIDENT VELOCITY PROFILE %%

% Falkner-skan function inputs
tolerancia=1e-8;
m=1;% m=0;
Cf_2_Re=1.23;% Cf_2_Re=0.33206;

%% STARTING THE GRID %%

positions=CreateGridXFOIL(N,J,h1,K,xi_PLOT); % create the cell array of
%positions having at each position: xi eta hj_seguent

% If using the Falkner-Skan function
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia);

Values={}; % cell array that would contain f, f', f'' and b

for j=1:1:J+1 % storing the incident velocity profile in values
      Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % If Falkner-skan function used
end

%% TRANSITON AND REYNOLDS PARAMETERS %%

Re=60000;
viscosity=1/Re;
```

```
viscosity=15*10^-6/1.225; % typical viscosity value

R_tr=5e5; % forced Reynolds for transition

% initializing Reynolds values and vectors
Rx_values(1,1)=0;
RT_values(1,1)=nan;
xtr_primer=1; %
R_trSolved=false;
found_tr=false;

%% MAIN LOOP %%

divergeix=0; % if is 0 is not diverging, meaning that the loop must continue
for n=2:1:N

    if divergeix==1 % is diverging
        break % stop the computation
    end

    % first of all, duplicatin the values of the previous column
    % in this way an initial guess is obtained
    for j=1:1:J+1
        Values{j,n}=Values{j,n-1}; % duplicating column
    end

    convergeix= 0; % to determine if the current column has been solved
    % exaggeratedly the values to ensure that a soltion is valid
    sum_r2_anterior=10e10;

    %%% TURBULENCE MODEL %%%

    Rx_values(1,n)=positions{1,n}(1)*u_e(n)/viscosity; % Local Reynolds

    if Rx_values(1,n)<R_tr % The local Reynolds has not reached the transition↙
Reynolds

        RX2=Rx_values(1,n); % saving as a variable the local Reynolds

        mom=ComputeMomentum(n,Values,positions,Rx_values); % momentum thickness until↙
the actual point
        RT_values(1,n)=u_e(n)*mom(1,n)/viscosity; % momentum Reynolds actual

        % Veryfing if the transition Reynolds has been reached (free transition)
        if (n>2) && (R_trSolved==false) % n has to be more than 2 to have another↙
point
            RX1=Rx_values(1,n-1); % previous Reynolds number

            RT_C1 =RT_values(1,n-1); % previous momentum Reynolds number
            RT_C2=RT_values(1,n); % actual momentum Reynolds number

            % Reynolds value of the empirical correlation for tha same
            % interval
            RT_T1=1.174*(1+22400/RX1)*RX1^0.46;
            RT_T2=1.174*(1+22400/RX2)*RX2^0.46;

            % Defining the segments
            Rx_interval=[RX1 RX2];
            Rt_computed=[RT_C1 RT_C2];
            Rt_tabulated=[RT_T1 RT_T2];
```

```
            % Verifying if the transition has occured, if the segments
            % intersect
            [intercepting_Point, found_tr]=intersectionPoint(Rx_interval, Rt_computed,↙
Rx_interval, Rt_tabulated);

            if found_tr==true % if transition has reached
                R_tr=RX1; % saving the transition Reynolds
                R_trSolved=true;
            end
        end
    end

    if Rx_values(1,n)>=R_tr % if the local Reynolds is higher than the transition -↙
turbulence model
        if xtr_primer==1 % if it is the first time that enters in the loop, we save↙
the transition values
            x_tr=positions{j,n-1}(1);
            u_e_x_tr=double(u_e(n-1));
        end
        xtr_primer=0;
        % apply turbulence model
        mom=ComputeMomentum(n,Values,positions,Rx_values);
        [Values,em_values]=TurbulenceInfluence(viscosity,Values,positions,n,m,x_tr,0,↙
u_e(n),u_e_x_tr,mom,0,u_e,xi_PLOT);
    end


    while convergeix==0 % if the column has not yet been solved
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %↙
Fill the A and r cell arrays
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % applying gauss method, multiplying by↙
-B1*inv(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing the↙
increments

        % check if the code is diverging. If it is diverging a separation point has↙
been reached
        [sq_sum_r2]=SumaResidu(r); % squared sum of the residual values
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it has increased regarding↙
the precious value
            sum_r2_anterior=sq_sum_r2;
        else
            % the code is diverging. It is eliminated the last column as the values↙
are not correct
            Values(:,n)=[];
            divergeix=1;
            break
        end

        % If the separation point has not been reached, it is checked if the obtained↙
result is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end
        if convergeix == 0 % if it is not a good solution, the increments are summed
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
```

```
            end
        end
    end
end

%% PLOTS %%
[a,sp]=size(Values); % files, columnes

eta_PLOT=zeros(1,a); % eta values
fprime1=zeros(1,a);
xi_PLOT=zeros(1,sp);

%% Incident velocity profile %%
for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(2);
end

figure
plot(eta_PLOT,fprime1,'.')
xlabel('eta')
ylabel('f prime')
title ('Velocity profile')

%% color plot %%
matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none')
colorbar
title('[f´]')
xlabel('x')
ylabel('η')

%% computation of values for plots %%
Rx_values=zeros(1,sp);
Rtheta_values=zeros(1,sp);
rtT=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);
bl_t=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u_e(n)*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u_e(n))^2;
    for j=1:1:a
        if Values{j,n}(2)>=0.99
            bl_t(1,n)=positions{j,n}(2)*sqrt(viscosity*xi_PLOT(1,n)/u_e(n));
            break
```

```
        end
    end
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

for n=1:1:sp
    Rtheta_values(1,n)=u_e(n)*theta(1,n)/viscosity;
    rtT(1,n)=1.174*(1+22400/Rx_values(1,n))*Rx_values(1,n)^0.46;
end

%% displacement and momentum thicness %%
figure
hold on
plot(xi_PLOT,displacement)
plot(xi_PLOT,theta)
legend ('Displacement thickness','Momentum thickness','Location',"best")

%% CF %%
figure
plot(xi_PLOT,cf)
grid
xlabel('x')
ylabel ('Cf')

%% H %%
figure
plot(xi_PLOT,H)
title('Form Factor evolution')
grid
xlabel('x')
ylabel('H')

%% Reynolds theta %%
figure
plot(xi_PLOT,Rtheta_values)
grid
xlabel('x')
ylabel('Re_θ')

%% TRANSITION CRITERION %%
figure
hold on
plot(Rx_values,Rtheta_values)
plot(Rx_values,rtT)
legend ('Computed','Tabulated')
grid
```

```
close all
clear all

%% GRID PARAMETERS DEFINITION %%
K=1; % defining homogeneous grid or not
if K~=1 % not homogeneous
    eta_e=7;
    h1=0.01;
    J=round(log(1+(K-1)*(eta_e/h1))/log(K)+1)+5;
else % homogeneous
    J=1000;
    h1=0.01;
end

%% DEFINING EXTERNAL VELOCITY PROFILE %%

%%%%%%% If defining the external velocity from an Xfoil file %%%%%%%
[m_vector_in,m_vector_out,x_arc_out,x_arc_in,u_e_out,u_e_in] = ...
    ImportMvector('turbulence.txt'); % Importing from an Xfoil file

% Comment and uncomment the desired set of values from the intrados or
% extrados:

% EXTRADOS VALUES
% m_vector=m_vector_out;
% xi_PLOT=x_arc_out;
% u_e=u_e_out;

% INTRADOS VALUES
m_vector=m_vector_in;
xi_PLOT=x_arc_in;
u_e=-10*u_e_in;

N= length(xi_PLOT); % overwriting the value

%% DEFINING THE INCIDENT VELOCITY PROFILE %%

% Falkner-skan function inputs
tolerancia=1e-8;
m=1;% m=0;
Cf_2_Re=1.23;% Cf_2_Re=0.33206;

%% STARTING THE GRID %%

positions=CreateGridXFOIL(N,J,h1,K,xi_PLOT); % create the cell array of
%positions having at each position: xi eta hj_seguent

% If using the Falkner-Skan function
[h] = FalknerSkanAero(h1,K,J+1,m,Cf_2_Re,tolerancia);

Values={}; % cell array that would contain f, f', f'' and b

for j=1:1:J+1 % storing the incident velocity profile in values
      Values{j,1}=[h(j,1),h(j,2),h(j,3),1]; % If Falkner-skan function used
end

%% TRANSITON AND REYNOLDS PARAMETERS %%

Re=50000;
viscosity=1/Re;
```

```
viscosity=15*10^-6/1.225; % typical viscosity value

R_tr=2.2e5; % forced Reynolds for transition

% initializing Reynolds values and vectors
Rx_values(1,1)=0;
RT_values(1,1)=nan;
xtr_primer=1; %
R_trSolved=false;
found_tr=false;

%% MAIN LOOP %%

divergeix=0; % if is 0 is not diverging, meaning that the loop must continue
for n=2:1:N

    if divergeix==1 % is diverging
        break % stop the computation
    end

    % first of all, duplicatin the values of the previous column
    % in this way an initial guess is obtained
    for j=1:1:J+1
        Values{j,n}=Values{j,n-1}; % duplicating column
    end

    convergeix= 0; % to determine if the current column has been solved
    % exaggeratedly the values to ensure that a soltion is valid
    sum_r2_anterior=10e10;

    %%% TURBULENCE MODEL %%%

    Rx_values(1,n)=positions{1,n}(1)*u_e(n)/viscosity; % Local Reynolds

    if Rx_values(1,n)<R_tr % The local Reynolds has not reached the transition
Reynolds

        RX2=Rx_values(1,n); % saving as a variable the local Reynolds

        mom=ComputeMomentum(n,Values,positions,Rx_values); % momentum thickness until
the actual point
        RT_values(1,n)=u_e(n)*mom(1,n)/viscosity; % momentum Reynolds actual

        % Veryfing if the transition Reynolds has been reached (free transition)
        if (n>2) && (R_trSolved==false) % n has to be more than 2 to have another
point
            RX1=Rx_values(1,n-1); % previous Reynolds number

            RT_C1 =RT_values(1,n-1); % previous momentum Reynolds number
            RT_C2=RT_values(1,n); % actual momentum Reynolds number

            % Reynolds value of the empirical correlation for tha same
            % interval
            RT_T1=1.174*(1+22400/RX1)*RX1^0.46;
            RT_T2=1.174*(1+22400/RX2)*RX2^0.46;

            % Defining the segments
            Rx_interval=[RX1 RX2];
            Rt_computed=[RT_C1 RT_C2];
            Rt_tabulated=[RT_T1 RT_T2];
```

```
            % Verifying if the transition has occured, if the segments
            % intersect
            [intercepting_Point, found_tr]=intersectionPoint(Rx_interval, Rt_computed,↙
Rx_interval, Rt_tabulated);

            if found_tr==true % if transition has reached
                R_tr=RX1; % saving the transition Reynolds
                R_trSolved=true;
            end
        end
    end

    if Rx_values(1,n)>=R_tr % if the local Reynolds is higher than the transition -↙
turbulence model
        if xtr_primer==1 % if it is the first time that enters in the loop, we save↙
the transition values
            x_tr=positions{j,n-1}(1);
            u_e_x_tr=double(u_e(n-1));
        end
        xtr_primer=0;
        % apply turbulence model
        mom=ComputeMomentum(n,Values,positions,Rx_values);
        [Values,em_values]=TurbulenceInfluence(viscosity,Values,positions,n,m,x_tr,0,↙
u_e(n),u_e_x_tr,mom,0,u_e,xi_PLOT);
    end


    while convergeix==0 % if the column has not yet been solved
        [A, r]=Omplim_A_r(J,h1,n, Values, positions,m_vector(1,n),m_vector(1,n-1)); %↙
Fill the A and r cell arrays
        [A_gauss, r_gauss]=MetodeGauss(A,r,J); % applying gauss method, multiplying by↙
-B1*inv(A0)
        [increments] = Calculem_Increments(A_gauss, r_gauss,A,r); % computing the↙
increments

        % check if the code is diverging. If it is diverging a separation point has↙
been reached
        [sq_sum_r2]=SumaResidu(r); % squared sum of the residual values
        if abs(sum_r2_anterior)>abs(sq_sum_r2) % check if it has increased regarding↙
the precious value
            sum_r2_anterior=sq_sum_r2;
        else
            % the code is diverging. It is eliminated the last column as the values↙
are not correct
            Values(:,n)=[];
            divergeix=1;
            break
        end

        % If the separation point has not been reached, it is checked if the obtained↙
result is good enough
        if abs(increments{1,1}(3)) < 1e-8
            convergeix=1;
        end
        if convergeix == 0 % if it is not a good solution, the increments are summed
            for q=1:1:length(increments)
                Values{q,n}(1)= Values{q,n}(1)+ increments{q,1}(1);
                Values{q,n}(2)= Values{q,n}(2)+ increments{q,1}(2);
                Values{q,n}(3)= Values{q,n}(3)+ increments{q,1}(3);
```

```
            end
        end
    end
end

%% PLOTS %%
[a,sp]=size(Values); % files, columnes

eta_PLOT=zeros(1,a); % eta values
fprime1=zeros(1,a);
xi_PLOT=zeros(1,sp);

%% Incident velocity profile %%
for j=1:1:a
    eta_PLOT(1,j)=(positions{j,1}(2));
    fprime1(1,j)=Values{j,1}(2);
end

figure
plot(eta_PLOT,fprime1,'.')
xlabel('eta')
ylabel('f prime')
title ('Velocity profile')

%% color plot %%
matrix_f_p=zeros(a,sp);

for j=1:1:a
    for i=1:1:sp
        matrix_f_p(j,i)=Values{j,i}(2);
    end
end
for i=1:sp
    xi_PLOT(1,i)=(positions{1,i}(1));
end

figure
[X, Y] =meshgrid(xi_PLOT,eta_PLOT);
contourf(X,Y,matrix_f_p,'LevelStep',0.005,'edgecolor','none') % mes contorns
colorbar
title('Dimensionless velocity [f´] values')
xlabel('x')
ylabel('η')

%% computation of values for plots %%
Rx_values=zeros(1,sp);
Rtheta_values=zeros(1,sp);
rtT=zeros(1,sp);
cf=zeros(1,sp);
shear=zeros(1,sp);
bl_t=zeros(1,sp);

for n=1:1:sp
    Rx_values(1,n)=(u_e(n)*(positions{1,n}(1))/viscosity);
    cf(1,n)=2*Values{1,n}(3)/(sqrt(Rx_values(1,n)));
    shear(1,n)=cf(1,n)*0.5*(u_e(n))^2;
    for j=1:1:a
        if Values{j,n}(2)>=0.99
            bl_t(1,n)=positions{j,n}(2)*sqrt(viscosity*xi_PLOT(1,n)/u_e(n));
            break
```

```
        end
    end
end

[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values);
[theta]=ComputeMomentum(sp,Values, positions,Rx_values);

H=displacement./theta;

for n=1:1:sp
    Rtheta_values(1,n)=u_e(n)*theta(1,n)/viscosity;
    rtT(1,n)=1.174*(1+22400/Rx_values(1,n))*Rx_values(1,n)^0.46;
end

%% displacement and momentum thicness %%
figure
hold on
plot(xi_PLOT,displacement)
plot(xi_PLOT,theta)
legend ('Displacement thickness','Momentum thickness','Location',"best")

%% CF %%
figure
semilogy(xi_PLOT,cf)
grid
xlabel('x')
ylabel ('Cf')

%% H %%
figure
plot(xi_PLOT,H)
title('Form Factor evolution')
grid
xlabel('x')
ylabel('H')

%% Reynolds theta %%
figure
plot(xi_PLOT,Rtheta_values)
grid
xlabel('x')
ylabel('Re_θ')

%% TRANSITION CRITERION %%
figure
hold on
plot(Rx_values,Rtheta_values)
plot(Rx_values,rtT)
legend ('Computed','Tabulated')
grid
```

```
%% CREATING THE GRID POSITIONS %%

function [positions]=CreateGrid(N,J,h1,K)
    % Creating a cell array where the positions are going to be stored
    positions=cell(J+1,N+1); % each cell would have:  xi eta hj_next
    % loop sweeping the xi direction
    for n=1:1:N+1
        % loop sweepint the eta direction
        for j=1:1:J+1
           if n==1 && j==1 % initial condition, point (0,0)
               positions{j,n}=[0;0;h1];
           elseif j==1 && n~=1 % during the first row, eta is always 0
               positions{j,n}=[(n-1)*1/N;0;h1];
           elseif j~=1 % other cases
               n_position=(n-1)*1/N;
               eta_position=positions{j-1,n}(2)+positions{j-1,n}(3);
               hj_next=K*positions{j-1,n}(3);
               positions{j,n}=[n_position;eta_position;hj_next];
           end
        end
    end
end
```

```
%% CREATING THE GRID POSITIONS -XFOIL %%
function [positions]=CreateGridXFOIL(N,J,h1,K,x_arc_out)
    positions={}; % Creating a cell array where the positions are going to be stored
    for n=1:1:N  % loop sweeping the xi direction
        for j=1:1:J+1
           if n==1 && j==1 %  (0,0)
               positions{j,n}=[0;0;h1];
           elseif j==1 && n~=1 % during the first row, eta is always 0
               positions{j,n}=[x_arc_out(n);0;h1];
           elseif j~=1 % other cases
               n_position=x_arc_out(n);
               eta_position=positions{j-1,n}(2)+positions{j-1,n}(3);
               hj_next=K*positions{j-1,n}(3);
               positions{j,n}=[n_position;eta_position;hj_next]; % xi eta hj+1
           end
        end
    end
end
```

```
 %% FILLING A AND R %%

function [A, r] = Omplim_A_r(J,h1,n, Values, positions,m_n,m_n1)
A=cell(J+1,3); % initializing the A cell array
r=cell(J+1,1); % initializing the r cell array

% main loop sweeping all the j positions
for j=1:1:J+1
    if j==1 % position in contact with the wall
        % DEFINING SOME NEEDED PARAMETERS
        hj_plus=positions{j,n}(3);

        % DEFINING A, B, C matrices for j=1
        A_j=[1 0 0; 0 1 0; 0 -1 -h1/2];
        C_j=[0 0 0;0 0 0;0 1 -hj_plus/2];
        B_j=0;

        % STORING THE A VALUES
        A{j,1}=B_j;
        A{j,2}=A_j;
        A{j,3}=C_j;

        % STORING THE r VALUES
        % r3 is not going to be 0 but as it is computed r3_{j-1} it is
        % initially defined as 0 and in the next j iteration it would
        % be defined
        r{j,1}=[0;0;0];

    elseif j~=1 %middle positions
        % DEFINING SOME NEEDED PARAMETERS
        xi_12=(positions{j,n}(1)+positions{j,n-1}(1))/2;
        alpha_n=xi_12/((positions{j,n}(1)-positions{j,n-1}(1)));
        alpha_1=((m_n+1)/2)+alpha_n;
        alpha_2=m_n+alpha_n;

        % centering values
        f_n1_j12=(Values{j,n-1}(1)+Values{j-1,n-1}(1))/2; % f_n-1_j-1/2
        v_n1_j12=(Values{j,n-1}(3)+Values{j-1,n-1}(3))/2; % v_n-1_j-1/2
        u_n1_j12=(Values{j,n-1}(2)+Values{j-1,n-1}(2))/2; % u_n-1_j-1/2
        f_n_j12=(Values{j,n}(1)+Values{j-1,n}(1))/2; % f_n_j-1/2
        v_n_j12=(Values{j,n}(3)+Values{j-1,n}(3))/2; % v_n_j-1/2
        u_n_j12=(Values{j,n}(2)+Values{j-1,n}(2))/2; % u_n_j-1/2

        % h values
        hj_plus1=positions{j,n}(3);
        hj =positions{j-1,n}(3);

        % computing s1, ... , s6
        s1j=(1/hj)*Values{j,n}(4)+(alpha_1/2)*Values{j,n}(1)-(alpha_n/2)*f_n1_j12;
        s2j=-(1/hj)*Values{j-1,n}(4)+(alpha_1/2)*Values{j-1,n}(1)-(alpha_n/2)↙
*f_n1_j12;
        s3j=(alpha_1/2)*Values{j,n}(3)+(alpha_n/2)*v_n1_j12;
        s4j=(alpha_1/2)*Values{j-1,n}(3)+(alpha_n/2)*v_n1_j12;
        s5j=-alpha_2*Values{j,n}(2);
        s6j=-alpha_2*Values{j-1,n}(2);

        % computing r1 and r2
        r1j= Values{j-1,n}(1)- Values{j,n}(1)+hj*u_n_j12;

        L_n1_j12=(1/hj)*(Values{j,n-1}(4)*Values{j,n-1}(3)-Values{j-1,n-1}(4)*Values↙
```

```
{j-1,n-1}(3))+...
            ((m_n1+1)/2)*(f_n1_j12*v_n1_j12)+m_n1*(1-(u_n1_j12*u_n1_j12));
        R_n_1_j_12=-L_n1_j12+alpha_n*((f_n1_j12*v_n1_j12)-(u_n1_j12*u_n1_j12))-m_n;
        r2j= R_n_1_j_12-((1/hj)*(Values{j,n}(4)*Values{j,n}(3)-Values{j-1,n}(4)*Values
{j-1,n}(3))...
            +alpha_1*(f_n_j12*v_n_j12)-alpha_2*(u_n_j12*u_n_j12)+alpha_n*
(v_n1_j12*f_n_j12-f_n1_j12*v_n_j12));

        % making distinction between the last position and middle
        % positons
        if 1<j && j<J+1 % middle positions

            % computing the r3_{j-1}
            r3j_1=Values{j-1,n}(2)-Values{j,n}(2)+hj*v_n_j12;

            % Defining A,B,C for middle positions
            A_j=[1 -hj/2 0; s3j s5j s1j; 0 -1 -hj_plus1/2];
            B_j=[-1 -hj/2 0; s4j s6j s2j; 0 0 0];
            C_j=[0 0 0;0 0 0;0 1 -hj_plus1/2];

            % STORING THE r VALUES
            r{j,1}=[r1j;r2j;0];
            r{j-1,1}(3)=r3j_1; % saving the r3 value in the previous iteration

            % STORING THE A VALUES
            A{j,1}=B_j;
            A{j,2}=A_j;
            A{j,3}=C_j;

        elseif j==J+1

            % defining the A, B, C matrices
            A_j=[1 -hj/2 0;s3j s5j s1j;0 1 0];
            B_j=[-1 -hj/2 0;s4j s6j s2j; 0 0 0];
            C_j=0;

            % STORING THE A VALUES
            A{j,1}=B_j;
            A{j,2}=A_j;
            A{j,3}=C_j;

            % STORING THE r VALUES
            % the last r3 value is 0
            r{j,1}=[r1j;r2j;0];

        end
    end
end
end
```

```
 %% APPLYING GAUSS METHOD %%

function [A_gauss, r_gauss] = MetodeGauss(A,r,J)
    A_gauss=cell(J,2); % initializing the A_gauss cell array
    r_gauss={J,1}; % initializing the r_gauss cell array

    % main loop sweeping all the j positions
    for j=1:1:J
        if j==1 % at contact with the surface
            A_gauss{j,1}=-A{j+1,1}*inv(A{j,2})*A{j,3}+A{j+1,2};
            A_gauss{j,2}=A{j+1,3};
            r_gauss{j,1}=-A{j+1,1}*inv(A{j,2})*r{j,1}+r{j+1,1};

        elseif 1<j && j<J % middel positions
            A_gauss{j,1}=-A{j+1,1}*inv(A_gauss{j-1,1})*A_gauss{j-1,2}+A{j+1,2};
            A_gauss{j,2}=A{j+1,3};
            r_gauss{j,1}=-A{j+1,1}*inv(A_gauss{j-1,1})*r_gauss{j-1,1}+r{j+1,1};

        else % the outer limit
            A_gauss{j,1}=-A{j+1,1}*inv(A_gauss{j-1,1})*A_gauss{j-1,2}+A{j+1,2};
            r_gauss{j,1}=-A{j+1,1}*inv(A_gauss{j-1,1})*r_gauss{j-1,1}+r{j+1,1};
        end
    end
end
```

```
%% COMPUNTING INCREMENTS%%
function [increments]= Calculem_Increments(A_gauss, r_gauss,A,r)
    increments={}; % new increments cell arry
    for w=length(r_gauss):-1:0 % start filling from last position
        if w==length(r_gauss)
            increments{w+1,1}=inv(A_gauss{w,1})*r_gauss{w,1};
        elseif w==0
            increments{1,1}=inv(A{1,2})*(r{1,1}-A{1,3}*increments{2,1});
        else
            increments{w+1,1}=inv(A_gauss{w,1})*(r_gauss{w,1}-A_gauss{w,2}*increments↙
{w+2,1});
        end
    end
end
```

```
%% FALKNER-SKAN FUNCTION %%
function [h]=FalknerSkanAero(h1,K,J,m,Cf_2_Re,tolerancia)

increments=zeros(1,J-1);
increments(1,1)=h1; % initial increment value
for j=2:1:J-1
    increments(1,j)= K*increments(1,j-1); %% filling the increments vector
end
eta_inf=sum(increments);
increments=increments/(eta_inf);

beta=m; % adjusting it
a=(m+1)/2;

% Initial parameters and boundary conditions
h00=0;
h10=0;
h1K=1;

h20_0=(Cf_2_Re*eta_inf)/((m+1)/2)^(1/2);
h20_1=h20_0+increments(1,1);
h20=[h20_0,h20_1];

% defining variables and vectors
encontrado=false;
F=[];
n=1;

while encontrado == false % if it is not found
    h=[h00,h10,h20(n)];
    H=[h10,h20(n),-(eta_inf^2)*(a*h00*h20(n)+beta(1-(h10^2)))];
    for i=1:1:J-1
        for j=1:3
            h(i+1,j)=h(i,j)+H(i,j)*increments(1,i);
        end
        H(i+1,1)=h(i+1,2);
        H(i+1,2)=h(i+1,3);
        H(i+1,3)=-(eta_inf^2)*(a*h(i+1,1)*h(i+1,3)+beta*(1-(h(i+1,2)^2)));
    end
    F(n)=h1K-h(end,2);

    if(abs(F(n))<tolerancia) % checking if is ok the solution
        encontrado=true;
    elseif(n>=2)
        h20(n+1)=(F(n)*h20(n-1)-F(n-1)*h20(n))/(F(n)-F(n-1));
    end
    n=n+1;
end

for j=1:length(h) % un-doing the transformations
    h(j,1)=h(j,1)*eta_inf;
    h(j,3)=h(j,3)/eta_inf;
end
```

```
%% TUBULENCE MODEL %%
function [Values, Em_vector]=TurbulenceInfluence(viscosity,Values,positions,n,m,x_tr,↙
alpha,u_e,u_e_x_tr,mom,u0,u_e_vector,x_vector)
% Common values
f_prime_prime_w=Values{1,n}(3);
Re_x_tr=u_e_x_tr*x_tr/viscosity;
eta_e=positions{end,1}(2);
f_e=Values{end,n}(1);
% Graphics for computing the gamma
x_Y=↙
[1.29464,1.3068,1.30825,1.3136,1.32042,1.32236,1.33161,1.34134,1.35059,1.37251,1.39199↙
,1.40708,1.43143,1.46017,1.4928,1.52641,1.5722,1.61361,1.71688,1.78216,1.83136,1.89372↙
,1.97605,2.02331,2.09931,2.15533,2.20356,2.2401,2.27956,2.3239,2.39551];
y_Y=↙
[9.4793,8.7657,8.3918,8.1248,7.9257,7.7072,7.3869,7.0616,6.7995,6.4307,6.091,5.8047,5.↙
4116,5.072,4.8878,4.7085,4.442,4.2482,3.8365,3.6818,3.561,3.4257,3.3052,3.2377,3.1317,↙
3.0886,3.0357,3.0264,2.9928,2.9836,2.9746];
Y_poly=polyfit(x_Y,y_Y,10);

x_sigma=↙
[1.29846,1.30462,1.31569,1.32308,1.33292,1.34523,1.35385,1.37108,1.40062,1.43015,1.467↙
08,1.50523,1.55077,1.61108,1.68123,1.73538,1.78954,1.86092,1.93231,2.01846,2.08246,2.1↙
5631,2.22523,2.29908,2.35569];
y_sigma=↙
[2.01726,1.84959,1.70658,1.60548,1.48466,1.35397,1.25534,1.15671,1.04329,0.93973,0.853↙
42,0.78932,0.7326,0.67096,0.61425,0.58712,0.55014,0.52055,0.51068,0.48603,0.46877,0.44↙
904,0.43918,0.42685,0.41452];
sigma_poly=polyfit(x_sigma,y_sigma,10);
Em_vector=zeros(1,length(Values));

% Size of values
[a,~]=size(Values);
% computing the boundary layer thickness
for j=1:1:a
    if Values{j,n}(2)>=0.99
        bl_t=positions{j,n}(2)*sqrt(viscosity*positions{1,2}(1)/u_e);
        break
    end
end

% for all the j
for j=1:1:length(Values)

    x=positions{j,n}(1);
    eta=positions{j,n}(2);
    f_prime_prime=Values{j,n}(3);
    Re_x=u_e*x/viscosity;
    y=eta*sqrt(viscosity*x/u_e);

    C_2=213*(log10(Re_x_tr)-4.7323);
    G=3/C_2*u_e^3/viscosity^2*Re_x_tr^(-1.34);
    %     [I]=SumIntegral(u_e_vector,x_tr,x,x_vector); % for x-foil
    I=-((1-x)^(-alpha+1)-(1-x_tr)^(-alpha+1))/(u0*(-alpha+1)); % for re-compression↙
u_e=u0(1-x)^alpha
    Gamma_tr=1-exp(-G*(x-x_tr)*I);

    % to determine if inner o outer region
    cf=2*Values{1,n}(3)/(sqrt(Re_x));
    shear=cf*0.5*(u_e)^2;
    y_plus=y*(shear/1.225)^0.5/viscosity;
```

```
    % if (positions{j,n}*(sqrt(viscosity*positions{j,n}(1)/u_e)))<0.2*bl_t
    % alternative way to determine if it is inner or outer region
    if y_plus<105
        p= m*Re_x^(1/4)*f_prime_prime_w^(-3/2);
        N=abs((1-11.8*p)^0.5);
        y_A=N/26*Re_x^(1/4)*f_prime_prime_w^0.5*eta;
        Em=0.16*Re_x^0.5*(1-exp(-y_A))^2*eta^2*f_prime_prime*Gamma_tr;
    else

        displacement=positions{1,n}(1)/Re_x^0.5*(positions{end,n}(2)-Values{end,n}↙
(1));
        H=displacement/mom(end);

        Y_displacement=polyval(Y_poly,H);
        sigma_displacement=polyval(sigma_poly,H);

        sigma=sigma_displacement*displacement;
        Y=Y_displacement*displacement;

        Gamma=0.5*(1-erf((y-Y)/(sqrt(2)*sigma)));

        Em=0.0168*Re_x^0.5*(eta_e-f_e)*Gamma_tr*Gamma;
    end

    Em_vector(1,j)=Em;

end
% actualizing all the values
for j=1:1:length(Values)
    Values{j,n}(4)=1+Em_vector(1,j);
end
end
```

```
%% IMPORTING Xfoil DATA %%
function [m_vector_in,m_vector_out,x_arc_out,x_arc_in,u_e_out,u_e_in] = ImportMvector↙
(name)
A=importdata(name);
s_i=A.data(:,1);
u_e_i=A.data(:,4);
x=A.data(:,2);

% if the x is bigger than 1 we are in the wake zone
i=1;
while x(i)<=1.0
    s(i)=s_i(i);
    u_e(i)=u_e_i(i);
    if length(x)>i
        i=i+1;
    else
        break
    end
end

% Point that separates the outer and inner region
Index= find(abs(u_e)==min(abs(u_e)));
% Outer region
u_e_out=flip(u_e(1:Index));
x_arc_out=flip(s(Index)-s(1:Index));

m_vector_out=ones(1,length(u_e_out));
% As starting from a SP, the first value is 1

for n=2:length(u_e_out)
    % applying backward difference
    m_vector_out(1,n)=x_arc_out(n)/u_e_out(n)*((u_e_out(n)-u_e_out(n-1))/(x_arc_out(n)↙
-x_arc_out(n-1)));

end


% Inner region
u_e_in=u_e(Index:end);
x_arc_in=s(Index:end)-s(Index);

m_vector_in=ones(1,length(u_e_in));
% As starting from a SP, the first value is 1

for n=2:length(u_e_in)
    % applying backward difference
    m_vector_in(1,n)=x_arc_in(n)/u_e_in(n)*((u_e_in(n)-u_e_in(n-1))/(x_arc_in(n)-↙
x_arc_in(n-1)));

end
end
```

```
function[displacement]=DisplacementThicknessCompute(sp,positions,Rx_values,Values)
    displacement=zeros(1,sp);
    for n=1:1:sp
        displacement(1,n)=positions{1,n}(1)/Rx_values(1,n)^0.5*(positions{end,n}(2)-
Values{end,n}(1));
    end

end
```

```
function [theta]=ComputeMomentum(sp,Values, positions,Rx_values)
theta=zeros(1,sp);
[a,~]=size(Values);
for n=1:1:sp
    acum_sum=0;
    for j=1:a
        if j==1
            I=(Values{j,n}(2))^2*positions{j,n}(3);
        else
            I=(Values{j,n}(2))^2*positions{j-1,n}(3);
        end
        acum_sum=acum_sum+I;
    end
    theta(1,n)=(positions{1,n}(1))*(Values{end,n}(1)-acum_sum)/(sqrt(Rx_values(1,n)));
end
end
```

```
function [sq_sum_r2]=SumaResidu(r)
sq_sum_r2=0;
for i=1:1:length(r)
    sq_sum_r2=sq_sum_r2+r{i,1}(2)^2;
end
sq_sum_r2=sqrt(sq_sum_r2);
end
```

```
%% INTEGRAL COMPUTATION OF TUBULENCE MODEL %%
function [sum]=SumIntegral(u_e,xtr,x,x_vector)
if x==xtr
    sum=0;
else
    sum=0;
    for i=2:length(u_e)
        if x>xtr && x_vector(i)<=x
            sum=sum+(1/u_e(i))*(x_vector(i)-x_vector(i-1));
        end
    end
end
end
```

```
function [x_INTERSECTION, found] = intersectionPoint(LINE_1x, LINE_1y, LINE_2x,↙
LINE_2y)

x1 = LINE_1x(1);
x2 = LINE_1x(2);

y1 = LINE_1y(1);
y2 = LINE_1y(2);

x3 = LINE_2x(1);
x4 = LINE_2x(2);

y3 = LINE_2y(1);
y4 = LINE_2y(2);

syms x
eq=(y1-y2)/(x1-x2)*(x-x1)+y1==(y3-y4)/(x3-x4)*(x-x3)+y3;

x_INTERSECTION=double(vpasolve(eq,x));

if isempty(x_INTERSECTION)
    found=false;
elseif (x1<=x_INTERSECTION)&&(x_INTERSECTION<=x2)
    found=true;
else
    found=false;
end

end
```

| # s | x | y | Ue/Vinf | Dstar | Theta | Cf | H |
|---|---|---|---|---|---|---|---|
| 0.00000 | 1.00000 | 0.00126 | 1.02706 | 0.022513 | 0.003203 | -0.000231 | 7.029 |
| 0.00191 | 0.99811 | 0.00153 | 1.02718 | 0.022512 | 0.003200 | -0.000345 | 7.036 |
| 0.00507 | 0.99498 | 0.00196 | 1.02736 | 0.022433 | 0.003195 | -0.000346 | 7.021 |
| 0.00866 | 0.99142 | 0.00246 | 1.02756 | 0.022329 | 0.003190 | -0.000347 | 7.000 |
| 0.01279 | 0.98733 | 0.00302 | 1.02779 | 0.022199 | 0.003184 | -0.000348 | 6.971 |
| 0.01762 | 0.98254 | 0.00368 | 1.02806 | 0.022036 | 0.003178 | -0.000349 | 6.934 |
| 0.02338 | 0.97684 | 0.00446 | 1.02837 | 0.021828 | 0.003170 | -0.000351 | 6.886 |
| 0.03037 | 0.96991 | 0.00540 | 1.02876 | 0.021563 | 0.003161 | -0.000353 | 6.822 |
| 0.03900 | 0.96136 | 0.00656 | 1.02925 | 0.021218 | 0.003149 | -0.000356 | 6.738 |
| 0.04974 | 0.95072 | 0.00797 | 1.02987 | 0.020773 | 0.003134 | -0.000360 | 6.628 |
| 0.06291 | 0.93765 | 0.00968 | 1.03066 | 0.020208 | 0.003116 | -0.000366 | 6.485 |
| 0.07845 | 0.92224 | 0.01167 | 1.03164 | 0.019523 | 0.003094 | -0.000372 | 6.310 |
| 0.09578 | 0.90505 | 0.01385 | 1.03281 | 0.018750 | 0.003068 | -0.000381 | 6.110 |
| 0.11418 | 0.88679 | 0.01611 | 1.03415 | 0.017929 | 0.003040 | -0.000390 | 5.897 |
| 0.13313 | 0.86798 | 0.01839 | 1.03565 | 0.017088 | 0.003009 | -0.000400 | 5.678 |
| 0.15232 | 0.84892 | 0.02065 | 1.03733 | 0.016255 | 0.002976 | -0.000407 | 5.462 |
| 0.17162 | 0.82975 | 0.02288 | 1.03918 | 0.015436 | 0.002941 | -0.000411 | 5.248 |
| 0.19097 | 0.81052 | 0.02506 | 1.04121 | 0.014641 | 0.002904 | -0.000411 | 5.042 |
| 0.21034 | 0.79127 | 0.02719 | 1.04345 | 0.013877 | 0.002865 | -0.000404 | 4.844 |
| 0.22972 | 0.77201 | 0.02928 | 1.04591 | 0.013143 | 0.002823 | -0.000384 | 4.656 |
| 0.24910 | 0.75273 | 0.03132 | 1.04859 | 0.012443 | 0.002779 | -0.000351 | 4.477 |
| 0.26848 | 0.73346 | 0.03331 | 1.05149 | 0.011778 | 0.002733 | -0.000300 | 4.309 |
| 0.28785 | 0.71418 | 0.03525 | 1.05458 | 0.011149 | 0.002686 | -0.000231 | 4.151 |
| 0.30721 | 0.69491 | 0.03713 | 1.05788 | 0.010557 | 0.002637 | -0.000142 | 4.003 |
| 0.32657 | 0.67564 | 0.03897 | 1.06141 | 0.010002 | 0.002586 | -0.000034 | 3.867 |
| 0.34592 | 0.65638 | 0.04075 | 1.06517 | 0.009479 | 0.002534 | 0.000094 | 3.741 |
| 0.36525 | 0.63712 | 0.04247 | 1.06916 | 0.008995 | 0.002479 | 0.000237 | 3.629 |
| 0.38457 | 0.61787 | 0.04414 | 1.07336 | 0.008543 | 0.002423 | 0.000395 | 3.526 |
| 0.40387 | 0.59864 | 0.04575 | 1.07776 | 0.008122 | 0.002365 | 0.000566 | 3.434 |
| 0.42316 | 0.57942 | 0.04729 | 1.08231 | 0.007731 | 0.002307 | 0.000747 | 3.352 |
| 0.44242 | 0.56021 | 0.04877 | 1.08701 | 0.007367 | 0.002247 | 0.000938 | 3.278 |
| 0.46165 | 0.54103 | 0.05017 | 1.09182 | 0.007025 | 0.002188 | 0.001139 | 3.211 |
| 0.48087 | 0.52186 | 0.05151 | 1.09671 | 0.006706 | 0.002128 | 0.001346 | 3.152 |
| 0.50005 | 0.50272 | 0.05277 | 1.10166 | 0.006407 | 0.002068 | 0.001560 | 3.098 |
| 0.51920 | 0.48361 | 0.05394 | 1.10665 | 0.006124 | 0.002008 | 0.001782 | 3.050 |
| 0.53831 | 0.46453 | 0.05504 | 1.11169 | 0.005854 | 0.001948 | 0.002014 | 3.005 |
| 0.55738 | 0.44548 | 0.05604 | 1.11672 | 0.005601 | 0.001888 | 0.002250 | 2.966 |
| 0.57642 | 0.42646 | 0.05694 | 1.12176 | 0.005357 | 0.001829 | 0.002499 | 2.929 |
| 0.59541 | 0.40749 | 0.05774 | 1.12679 | 0.005123 | 0.001770 | 0.002758 | 2.895 |
| 0.61434 | 0.38857 | 0.05844 | 1.13180 | 0.004899 | 0.001711 | 0.003029 | 2.864 |
| 0.63322 | 0.36970 | 0.05901 | 1.13678 | 0.004680 | 0.001651 | 0.003319 | 2.834 |
| 0.65205 | 0.35088 | 0.05947 | 1.14168 | 0.004472 | 0.001593 | 0.003616 | 2.808 |
| 0.67080 | 0.33213 | 0.05979 | 1.14655 | 0.004267 | 0.001534 | 0.003938 | 2.782 |
| 0.68948 | 0.31345 | 0.05998 | 1.15133 | 0.004067 | 0.001475 | 0.004281 | 2.758 |
| 0.70808 | 0.29485 | 0.06001 | 1.15598 | 0.003873 | 0.001416 | 0.004644 | 2.735 |
| 0.72659 | 0.27634 | 0.05989 | 1.16052 | 0.003683 | 0.001357 | 0.005036 | 2.714 |
| 0.74500 | 0.25793 | 0.05959 | 1.16491 | 0.003496 | 0.001298 | 0.005461 | 2.694 |
| 0.76329 | 0.23965 | 0.05912 | 1.16909 | 0.003312 | 0.001238 | 0.005921 | 2.675 |
| 0.78145 | 0.22150 | 0.05845 | 1.17307 | 0.003128 | 0.001178 | 0.006439 | 2.655 |
| 0.79945 | 0.20353 | 0.05757 | 1.17666 | 0.002951 | 0.001118 | 0.006981 | 2.639 |
| 0.81726 | 0.18575 | 0.05648 | 1.17998 | 0.002771 | 0.001057 | 0.007609 | 2.621 |
| 0.83484 | 0.16822 | 0.05514 | 1.18279 | 0.002594 | 0.000996 | 0.008302 | 2.605 |
| 0.85212 | 0.15101 | 0.05355 | 1.18498 | 0.002421 | 0.000934 | 0.009057 | 2.591 |
| 0.86902 | 0.13422 | 0.05171 | 1.18662 | 0.002243 | 0.000872 | 0.009976 | 2.574 |
| 0.88539 | 0.11798 | 0.04960 | 1.18719 | 0.002072 | 0.000809 | 0.010966 | 2.561 |
| 0.90105 | 0.10250 | 0.04724 | 1.18679 | 0.001898 | 0.000746 | 0.012180 | 2.545 |
| 0.91575 | 0.08802 | 0.04468 | 1.18484 | 0.001729 | 0.000684 | 0.013594 | 2.528 |
| 0.92923 | 0.07482 | 0.04196 | 1.18098 | 0.001569 | 0.000624 | 0.015169 | 2.513 |
| 0.94129 | 0.06309 | 0.03918 | 1.17515 | 0.001420 | 0.000568 | 0.016919 | 2.500 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.95184 | 0.05290 | 0.03641 | 1.16728 | 0.001280 | 0.000515 | 0.019005 | 2.483 |
| 0.96096 | 0.04420 | 0.03371 | 1.15674 | 0.001155 | 0.000468 | 0.021169 | 2.469 |
| 0.96878 | 0.03681 | 0.03112 | 1.14372 | 0.001039 | 0.000424 | 0.023692 | 2.451 |
| 0.97552 | 0.03055 | 0.02864 | 1.12726 | 0.000934 | 0.000384 | 0.026547 | 2.430 |
| 0.98136 | 0.02522 | 0.02625 | 1.10668 | 0.000846 | 0.000349 | 0.029075 | 2.421 |
| 0.98647 | 0.02065 | 0.02395 | 1.08254 | 0.000758 | 0.000316 | 0.032474 | 2.398 |
| 0.99099 | 0.01673 | 0.02172 | 1.05209 | 0.000682 | 0.000286 | 0.035556 | 2.383 |
| 0.99503 | 0.01334 | 0.01953 | 1.01468 | 0.000609 | 0.000259 | 0.039420 | 2.356 |
| 0.99867 | 0.01041 | 0.01736 | 0.96759 | 0.000550 | 0.000234 | 0.041971 | 2.349 |
| 1.00198 | 0.00789 | 0.01521 | 0.91074 | 0.000489 | 0.000210 | 0.045499 | 2.324 |
| 1.00503 | 0.00574 | 0.01305 | 0.83633 | 0.000436 | 0.000190 | 0.048366 | 2.292 |
| 1.00784 | 0.00394 | 0.01088 | 0.74379 | 0.000397 | 0.000173 | 0.047227 | 2.291 |
| 1.01047 | 0.00249 | 0.00869 | 0.63360 | 0.000355 | 0.000157 | 0.046040 | 2.266 |
| 1.01294 | 0.00137 | 0.00649 | 0.49597 | 0.000326 | 0.000146 | 0.040115 | 2.240 |
| 1.01528 | 0.00059 | 0.00429 | 0.34098 | 0.000310 | 0.000138 | 0.028862 | 2.245 |
| 1.01750 | 0.00014 | 0.00211 | 0.17326 | 0.000291 | 0.000131 | 0.015855 | 2.230 |
| 1.01962 | 0.00000 | 0.00000 | -0.00224 | 0.000291 | 0.000131 | 0.000204 | 2.230 |
| 1.02174 | 0.00014 | -0.00212 | -0.16961 | 0.000302 | 0.000135 | 0.014999 | 2.229 |
| 1.02396 | 0.00059 | -0.00429 | -0.34131 | 0.000302 | 0.000135 | 0.030180 | 2.229 |
| 1.02629 | 0.00137 | -0.00649 | -0.49717 | 0.000329 | 0.000146 | 0.039049 | 2.263 |
| 1.02876 | 0.00249 | -0.00869 | -0.63270 | 0.000354 | 0.000157 | 0.046604 | 2.254 |
| 1.03139 | 0.00394 | -0.01088 | -0.74456 | 0.000397 | 0.000173 | 0.047196 | 2.294 |
| 1.03421 | 0.00574 | -0.01305 | -0.83659 | 0.000436 | 0.000190 | 0.048315 | 2.292 |
| 1.03725 | 0.00789 | -0.01521 | -0.90992 | 0.000491 | 0.000211 | 0.045134 | 2.328 |
| 1.04057 | 0.01041 | -0.01736 | -0.96856 | 0.000546 | 0.000233 | 0.042750 | 2.339 |
| 1.04421 | 0.01334 | -0.01953 | -1.01449 | 0.000613 | 0.000259 | 0.038750 | 2.367 |
| 1.04824 | 0.01673 | -0.02172 | -1.05208 | 0.000681 | 0.000286 | 0.035826 | 2.378 |
| 1.05276 | 0.02065 | -0.02395 | -1.08228 | 0.000760 | 0.000316 | 0.032299 | 2.401 |
| 1.05788 | 0.02522 | -0.02625 | -1.10686 | 0.000844 | 0.000349 | 0.029256 | 2.417 |
| 1.06372 | 0.03055 | -0.02864 | -1.12714 | 0.000936 | 0.000385 | 0.026417 | 2.433 |
| 1.07045 | 0.03681 | -0.03112 | -1.14377 | 0.001038 | 0.000424 | 0.023761 | 2.449 |
| 1.07828 | 0.04420 | -0.03371 | -1.15675 | 0.001155 | 0.000468 | 0.021152 | 2.470 |
| 1.08739 | 0.05290 | -0.03641 | -1.16712 | 0.001281 | 0.000516 | 0.018955 | 2.484 |
| 1.09794 | 0.06309 | -0.03918 | -1.17533 | 0.001418 | 0.000568 | 0.017009 | 2.497 |
| 1.11000 | 0.07482 | -0.04196 | -1.18100 | 0.001571 | 0.000624 | 0.015131 | 2.515 |
| 1.12349 | 0.08803 | -0.04468 | -1.18470 | 0.001731 | 0.000684 | 0.013555 | 2.530 |
| 1.13819 | 0.10250 | -0.04724 | -1.18674 | 0.001897 | 0.000746 | 0.012195 | 2.544 |
| 1.15385 | 0.11798 | -0.04960 | -1.18724 | 0.002070 | 0.000809 | 0.010990 | 2.560 |
| 1.17022 | 0.13422 | -0.05171 | -1.18661 | 0.002244 | 0.000872 | 0.009968 | 2.574 |
| 1.18711 | 0.15101 | -0.05355 | -1.18506 | 0.002419 | 0.000934 | 0.009074 | 2.590 |
| 1.20440 | 0.16822 | -0.05514 | -1.18280 | 0.002595 | 0.000996 | 0.008293 | 2.605 |
| 1.22198 | 0.18575 | -0.05648 | -1.17996 | 0.002772 | 0.001057 | 0.007601 | 2.622 |
| 1.23979 | 0.20353 | -0.05757 | -1.17668 | 0.002950 | 0.001118 | 0.006986 | 2.638 |
| 1.25778 | 0.22150 | -0.05845 | -1.17300 | 0.003131 | 0.001179 | 0.006423 | 2.656 |
| 1.27594 | 0.23965 | -0.05912 | -1.16911 | 0.003310 | 0.001238 | 0.005931 | 2.673 |
| 1.29423 | 0.25793 | -0.05959 | -1.16492 | 0.003495 | 0.001298 | 0.005464 | 2.693 |
| 1.31264 | 0.27634 | -0.05989 | -1.16052 | 0.003683 | 0.001357 | 0.005035 | 2.714 |
| 1.33115 | 0.29485 | -0.06001 | -1.15598 | 0.003873 | 0.001416 | 0.004643 | 2.735 |
| 1.34975 | 0.31345 | -0.05998 | -1.15133 | 0.004067 | 0.001475 | 0.004281 | 2.758 |
| 1.36843 | 0.33213 | -0.05979 | -1.14655 | 0.004267 | 0.001534 | 0.003938 | 2.782 |
| 1.38719 | 0.35088 | -0.05947 | -1.14167 | 0.004472 | 0.001593 | 0.003614 | 2.808 |
| 1.40601 | 0.36970 | -0.05901 | -1.13677 | 0.004681 | 0.001652 | 0.003316 | 2.834 |
| 1.42489 | 0.38857 | -0.05844 | -1.13182 | 0.004897 | 0.001710 | 0.003035 | 2.863 |
| 1.44383 | 0.40749 | -0.05774 | -1.12678 | 0.005124 | 0.001770 | 0.002756 | 2.895 |
| 1.46282 | 0.42647 | -0.05694 | -1.12177 | 0.005356 | 0.001829 | 0.002502 | 2.928 |
| 1.48185 | 0.44548 | -0.05604 | -1.11673 | 0.005600 | 0.001888 | 0.002252 | 2.966 |
| 1.50092 | 0.46453 | -0.05504 | -1.11168 | 0.005856 | 0.001948 | 0.002011 | 3.006 |
| 1.52004 | 0.48361 | -0.05394 | -1.10666 | 0.006123 | 0.002008 | 0.001783 | 3.050 |
| 1.53919 | 0.50272 | -0.05277 | -1.10165 | 0.006407 | 0.002068 | 0.001559 | 3.099 |
| 1.55837 | 0.52186 | -0.05151 | -1.09671 | 0.006706 | 0.002128 | 0.001346 | 3.152 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.57758 | 0.54103 | -0.05017 | -1.09181 | 0.007026 | 0.002188 | 0.001138 | 3.212 |
| 1.59682 | 0.56021 | -0.04877 | -1.08701 | 0.007367 | 0.002247 | 0.000938 | 3.278 |
| 1.61608 | 0.57942 | -0.04729 | -1.08232 | 0.007730 | 0.002307 | 0.000748 | 3.352 |
| 1.63536 | 0.59864 | -0.04575 | -1.07775 | 0.008124 | 0.002365 | 0.000565 | 3.435 |
| 1.65466 | 0.61787 | -0.04414 | -1.07336 | 0.008542 | 0.002423 | 0.000395 | 3.526 |
| 1.67398 | 0.63712 | -0.04247 | -1.06916 | 0.008996 | 0.002479 | 0.000236 | 3.629 |
| 1.69331 | 0.65638 | -0.04075 | -1.06517 | 0.009480 | 0.002534 | 0.000093 | 3.742 |
| 1.71266 | 0.67564 | -0.03897 | -1.06141 | 0.010001 | 0.002586 | -0.000034 | 3.867 |
| 1.73202 | 0.69491 | -0.03713 | -1.05788 | 0.010557 | 0.002637 | -0.000142 | 4.003 |
| 1.75139 | 0.71418 | -0.03525 | -1.05458 | 0.011149 | 0.002686 | -0.000231 | 4.151 |
| 1.77076 | 0.73346 | -0.03331 | -1.05149 | 0.011778 | 0.002733 | -0.000300 | 4.309 |
| 1.79014 | 0.75274 | -0.03132 | -1.04859 | 0.012443 | 0.002779 | -0.000351 | 4.477 |
| 1.80951 | 0.77201 | -0.02928 | -1.04591 | 0.013143 | 0.002823 | -0.000385 | 4.656 |
| 1.82889 | 0.79127 | -0.02719 | -1.04345 | 0.013877 | 0.002865 | -0.000404 | 4.844 |
| 1.84826 | 0.81052 | -0.02506 | -1.04121 | 0.014642 | 0.002904 | -0.000411 | 5.042 |
| 1.86761 | 0.82975 | -0.02288 | -1.03918 | 0.015436 | 0.002941 | -0.000411 | 5.248 |
| 1.88691 | 0.84892 | -0.02065 | -1.03733 | 0.016255 | 0.002976 | -0.000407 | 5.462 |
| 1.90611 | 0.86798 | -0.01839 | -1.03565 | 0.017088 | 0.003009 | -0.000400 | 5.679 |
| 1.92505 | 0.88679 | -0.01611 | -1.03415 | 0.017927 | 0.003040 | -0.000390 | 5.897 |
| 1.94345 | 0.90505 | -0.01385 | -1.03281 | 0.018749 | 0.003068 | -0.000381 | 6.110 |
| 1.96078 | 0.92224 | -0.01167 | -1.03164 | 0.019523 | 0.003094 | -0.000372 | 6.310 |
| 1.97632 | 0.93765 | -0.00968 | -1.03066 | 0.020207 | 0.003116 | -0.000366 | 6.485 |
| 1.98949 | 0.95072 | -0.00797 | -1.02987 | 0.020772 | 0.003134 | -0.000360 | 6.627 |
| 2.00023 | 0.96136 | -0.00656 | -1.02925 | 0.021217 | 0.003149 | -0.000356 | 6.738 |
| 2.00886 | 0.96991 | -0.00540 | -1.02876 | 0.021563 | 0.003161 | -0.000353 | 6.822 |
| 2.01585 | 0.97684 | -0.00446 | -1.02837 | 0.021827 | 0.003170 | -0.000351 | 6.885 |
| 2.02161 | 0.98254 | -0.00368 | -1.02806 | 0.022036 | 0.003178 | -0.000349 | 6.934 |
| 2.02644 | 0.98733 | -0.00302 | -1.02779 | 0.022199 | 0.003184 | -0.000348 | 6.971 |
| 2.03057 | 0.99142 | -0.00246 | -1.02756 | 0.022330 | 0.003190 | -0.000347 | 7.000 |
| 2.03416 | 0.99498 | -0.00196 | -1.02736 | 0.022434 | 0.003195 | -0.000346 | 7.021 |
| 2.03732 | 0.99811 | -0.00153 | -1.02718 | 0.022513 | 0.003200 | -0.000345 | 7.036 |
| 2.03923 | 1.00000 | -0.00126 | -1.02706 | 0.022513 | 0.003203 | -0.000231 | 7.029 |
| 2.03923 | 1.00010 | -0.00000 | 1.02706 | 0.047521 | 0.006405 | 0.000000 | 7.419 |
| 2.04114 | 1.00201 | -0.00000 | 1.02505 | 0.047258 | 0.006523 | 0.000000 | 7.244 |
| 2.04364 | 1.00451 | -0.00000 | 1.02246 | 0.046841 | 0.006676 | 0.000000 | 7.016 |
| 2.04691 | 1.00777 | -0.00000 | 1.01870 | 0.046254 | 0.006898 | 0.000000 | 6.706 |
| 2.05117 | 1.01204 | -0.00000 | 1.01353 | 0.045443 | 0.007203 | 0.000000 | 6.309 |
| 2.05675 | 1.01761 | -0.00000 | 1.00657 | 0.044327 | 0.007615 | 0.000000 | 5.821 |
| 2.06403 | 1.02490 | -0.00000 | 0.99704 | 0.042802 | 0.008180 | 0.000000 | 5.233 |
| 2.07355 | 1.03442 | -0.00000 | 0.98366 | 0.040735 | 0.008977 | 0.000000 | 4.538 |
| 2.08600 | 1.04686 | -0.00000 | 0.96437 | 0.037975 | 0.010138 | 0.000000 | 3.746 |
| 2.10226 | 1.06312 | -0.00000 | 0.94420 | 0.034255 | 0.011360 | 0.000000 | 3.016 |
| 2.12351 | 1.08438 | -0.00000 | 0.92685 | 0.029587 | 0.012394 | 0.000000 | 2.387 |
| 2.15128 | 1.11215 | -0.00000 | 0.91842 | 0.024395 | 0.012872 | 0.000000 | 1.895 |
| 2.18757 | 1.14844 | -0.00000 | 0.92278 | 0.019580 | 0.012647 | 0.000000 | 1.548 |
| 2.23500 | 1.19587 | -0.00000 | 0.93647 | 0.015931 | 0.012023 | 0.000000 | 1.325 |
| 2.29698 | 1.25785 | -0.00000 | 0.95216 | 0.013539 | 0.011390 | 0.000000 | 1.189 |
| 2.37799 | 1.33885 | -0.00000 | 0.96556 | 0.012068 | 0.010900 | 0.000000 | 1.107 |
| 2.48384 | 1.44471 | -0.00000 | 0.97583 | 0.011183 | 0.010550 | 0.000000 | 1.060 |
| 2.62218 | 1.58305 | -0.00000 | 0.98336 | 0.010653 | 0.010305 | 0.000000 | 1.034 |
| 2.80297 | 1.76384 | -0.00000 | 0.98876 | 0.010334 | 0.010136 | 0.000000 | 1.019 |
| 3.03923 | 2.00010 | -0.00001 | 0.99282 | 0.010131 | 0.010012 | 0.000000 | 1.012 |

| # s | x | y | Ue/Vinf | Dstar | Theta | Cf | H |
|---|---|---|---|---|---|---|---|
| 0.00000 | 1.00000 | 0.00126 | 0.97703 | 0.022914 | 0.007024 | 0.000281 | 3.262 |
| 0.00150 | 0.99852 | 0.00147 | 0.97824 | 0.022956 | 0.006978 | 0.000264 | 3.290 |
| 0.00391 | 0.99612 | 0.00180 | 0.98033 | 0.022949 | 0.006900 | 0.000243 | 3.326 |
| 0.00658 | 0.99349 | 0.00217 | 0.98266 | 0.022939 | 0.006812 | 0.000220 | 3.367 |
| 0.00954 | 0.99055 | 0.00258 | 0.98527 | 0.022928 | 0.006715 | 0.000194 | 3.415 |
| 0.01285 | 0.98727 | 0.00303 | 0.98824 | 0.022919 | 0.006606 | 0.000165 | 3.470 |
| 0.01661 | 0.98355 | 0.00354 | 0.99170 | 0.022913 | 0.006480 | 0.000131 | 3.536 |
| 0.02090 | 0.97930 | 0.00413 | 0.99578 | 0.022911 | 0.006333 | 0.000091 | 3.618 |
| 0.02588 | 0.97436 | 0.00480 | 1.00069 | 0.022922 | 0.006158 | 0.000045 | 3.722 |
| 0.03171 | 0.96858 | 0.00558 | 1.00671 | 0.022939 | 0.005948 | -0.000010 | 3.856 |
| 0.03861 | 0.96174 | 0.00650 | 1.01426 | 0.022974 | 0.005690 | -0.000074 | 4.038 |
| 0.04682 | 0.95361 | 0.00759 | 1.02438 | 0.023020 | 0.005352 | -0.000139 | 4.301 |
| 0.05652 | 0.94400 | 0.00886 | 1.03700 | 0.023078 | 0.004944 | -0.000186 | 4.668 |
| 0.06776 | 0.93285 | 0.01031 | 1.05190 | 0.023159 | 0.004481 | -0.000221 | 5.169 |
| 0.08039 | 0.92032 | 0.01192 | 1.06902 | 0.023235 | 0.003970 | -0.000244 | 5.852 |
| 0.09409 | 0.90673 | 0.01364 | 1.08826 | 0.023252 | 0.003424 | -0.000259 | 6.791 |
| 0.10848 | 0.89245 | 0.01541 | 1.10720 | 0.023175 | 0.002915 | -0.000395 | 7.951 |
| 0.12329 | 0.87775 | 0.01721 | 1.10819 | 0.023183 | 0.002891 | -0.000398 | 8.019 |
| 0.13833 | 0.86282 | 0.01901 | 1.10915 | 0.023046 | 0.002869 | -0.000401 | 8.034 |
| 0.15349 | 0.84776 | 0.02079 | 1.11009 | 0.022800 | 0.002847 | -0.000405 | 8.008 |
| 0.16871 | 0.83264 | 0.02254 | 1.11102 | 0.022473 | 0.002826 | -0.000409 | 7.953 |
| 0.18396 | 0.81749 | 0.02427 | 1.11195 | 0.022080 | 0.002805 | -0.000413 | 7.872 |
| 0.19923 | 0.80232 | 0.02597 | 1.11289 | 0.021634 | 0.002784 | -0.000418 | 7.770 |
| 0.21451 | 0.78713 | 0.02765 | 1.11385 | 0.021144 | 0.002764 | -0.000423 | 7.650 |
| 0.22979 | 0.77193 | 0.02929 | 1.11483 | 0.020618 | 0.002743 | -0.000429 | 7.517 |
| 0.24507 | 0.75674 | 0.03090 | 1.11585 | 0.020061 | 0.002722 | -0.000435 | 7.370 |
| 0.26036 | 0.74154 | 0.03248 | 1.11692 | 0.019480 | 0.002701 | -0.000441 | 7.213 |
| 0.27564 | 0.72634 | 0.03403 | 1.11803 | 0.018880 | 0.002679 | -0.000448 | 7.047 |
| 0.29091 | 0.71113 | 0.03555 | 1.11920 | 0.018263 | 0.002657 | -0.000456 | 6.874 |
| 0.30619 | 0.69593 | 0.03703 | 1.12045 | 0.017635 | 0.002634 | -0.000465 | 6.696 |
| 0.32145 | 0.68074 | 0.03849 | 1.12176 | 0.016998 | 0.002610 | -0.000474 | 6.513 |
| 0.33671 | 0.66554 | 0.03991 | 1.12317 | 0.016357 | 0.002585 | -0.000485 | 6.326 |
| 0.35197 | 0.65035 | 0.04129 | 1.12468 | 0.015713 | 0.002560 | -0.000496 | 6.138 |
| 0.36721 | 0.63517 | 0.04264 | 1.12631 | 0.015070 | 0.002533 | -0.000508 | 5.949 |
| 0.38245 | 0.61999 | 0.04396 | 1.12807 | 0.014433 | 0.002506 | -0.000520 | 5.760 |
| 0.39767 | 0.60482 | 0.04524 | 1.12997 | 0.013800 | 0.002477 | -0.000532 | 5.572 |
| 0.41289 | 0.58965 | 0.04647 | 1.13203 | 0.013180 | 0.002446 | -0.000540 | 5.388 |
| 0.42809 | 0.57450 | 0.04767 | 1.13426 | 0.012567 | 0.002415 | -0.000546 | 5.205 |
| 0.44327 | 0.55936 | 0.04883 | 1.13669 | 0.011971 | 0.002381 | -0.000547 | 5.027 |
| 0.45845 | 0.54423 | 0.04994 | 1.13933 | 0.011390 | 0.002346 | -0.000539 | 4.854 |
| 0.47360 | 0.52911 | 0.05101 | 1.14220 | 0.010825 | 0.002310 | -0.000520 | 4.686 |
| 0.48874 | 0.51400 | 0.05204 | 1.14530 | 0.010284 | 0.002272 | -0.000486 | 4.527 |
| 0.50386 | 0.49892 | 0.05301 | 1.14864 | 0.009761 | 0.002232 | -0.000434 | 4.373 |
| 0.51896 | 0.48384 | 0.05393 | 1.15222 | 0.009261 | 0.002191 | -0.000362 | 4.227 |
| 0.53404 | 0.46879 | 0.05480 | 1.15603 | 0.008782 | 0.002149 | -0.000266 | 4.087 |
| 0.54909 | 0.45376 | 0.05561 | 1.16009 | 0.008327 | 0.002105 | -0.000147 | 3.956 |
| 0.56412 | 0.43875 | 0.05637 | 1.16441 | 0.007896 | 0.002060 | -0.000004 | 3.834 |
| 0.57912 | 0.42377 | 0.05706 | 1.16900 | 0.007488 | 0.002013 | 0.000163 | 3.720 |
| 0.59409 | 0.40881 | 0.05769 | 1.17386 | 0.007106 | 0.001965 | 0.000351 | 3.616 |
| 0.60903 | 0.39388 | 0.05825 | 1.17897 | 0.006748 | 0.001916 | 0.000557 | 3.523 |
| 0.62394 | 0.37898 | 0.05874 | 1.18433 | 0.006407 | 0.001865 | 0.000786 | 3.436 |
| 0.63881 | 0.36411 | 0.05916 | 1.18989 | 0.006091 | 0.001814 | 0.001027 | 3.358 |
| 0.65365 | 0.34928 | 0.05950 | 1.19564 | 0.005794 | 0.001762 | 0.001282 | 3.289 |
| 0.66844 | 0.33449 | 0.05976 | 1.20158 | 0.005510 | 0.001709 | 0.001560 | 3.223 |
| 0.68319 | 0.31974 | 0.05993 | 1.20763 | 0.005248 | 0.001657 | 0.001838 | 3.168 |
| 0.69789 | 0.30505 | 0.06001 | 1.21383 | 0.004997 | 0.001604 | 0.002137 | 3.116 |
| 0.71253 | 0.29040 | 0.06000 | 1.22013 | 0.004760 | 0.001551 | 0.002449 | 3.069 |
| 0.72712 | 0.27581 | 0.05988 | 1.22651 | 0.004536 | 0.001498 | 0.002766 | 3.028 |
| 0.74164 | 0.26129 | 0.05966 | 1.23303 | 0.004318 | 0.001445 | 0.003115 | 2.988 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.75609 | 0.24684 | 0.05933 | 1.23957 | 0.004113 | 0.001392 | 0.003463 | 2.954 |
| 0.77047 | 0.23248 | 0.05888 | 1.24625 | 0.003911 | 0.001339 | 0.003850 | 2.921 |
| 0.78475 | 0.21820 | 0.05831 | 1.25297 | 0.003719 | 0.001286 | 0.004245 | 2.892 |
| 0.79894 | 0.20403 | 0.05760 | 1.25974 | 0.003534 | 0.001233 | 0.004658 | 2.866 |
| 0.81301 | 0.18999 | 0.05676 | 1.26665 | 0.003351 | 0.001179 | 0.005113 | 2.841 |
| 0.82694 | 0.17609 | 0.05577 | 1.27366 | 0.003171 | 0.001125 | 0.005611 | 2.818 |
| 0.84071 | 0.16237 | 0.05463 | 1.28072 | 0.002997 | 0.001071 | 0.006139 | 2.797 |
| 0.85428 | 0.14887 | 0.05334 | 1.28791 | 0.002823 | 0.001017 | 0.006735 | 2.776 |
| 0.86760 | 0.13563 | 0.05188 | 1.29515 | 0.002654 | 0.000962 | 0.007383 | 2.757 |
| 0.88060 | 0.12272 | 0.05025 | 1.30240 | 0.002489 | 0.000908 | 0.008073 | 2.742 |
| 0.89320 | 0.11025 | 0.04847 | 1.30986 | 0.002321 | 0.000853 | 0.008930 | 2.722 |
| 0.90529 | 0.09832 | 0.04654 | 1.31714 | 0.002159 | 0.000798 | 0.009859 | 2.705 |
| 0.91673 | 0.08706 | 0.04449 | 1.32422 | 0.002004 | 0.000745 | 0.010866 | 2.691 |
| 0.92741 | 0.07660 | 0.04235 | 1.33117 | 0.001852 | 0.000693 | 0.012078 | 2.673 |
| 0.93721 | 0.06704 | 0.04016 | 1.33764 | 0.001706 | 0.000642 | 0.013447 | 2.656 |
| 0.94608 | 0.05845 | 0.03796 | 1.34338 | 0.001573 | 0.000595 | 0.014891 | 2.642 |
| 0.95402 | 0.05081 | 0.03579 | 1.34861 | 0.001444 | 0.000550 | 0.016647 | 2.623 |
| 0.96108 | 0.04408 | 0.03367 | 1.35281 | 0.001324 | 0.000509 | 0.018663 | 2.602 |
| 0.96734 | 0.03817 | 0.03162 | 1.35561 | 0.001216 | 0.000470 | 0.020769 | 2.585 |
| 0.97289 | 0.03298 | 0.02963 | 1.35714 | 0.001117 | 0.000435 | 0.023049 | 2.568 |
| 0.97784 | 0.02842 | 0.02772 | 1.35740 | 0.001025 | 0.000402 | 0.025680 | 2.549 |
| 0.98228 | 0.02439 | 0.02586 | 1.35589 | 0.000940 | 0.000371 | 0.028548 | 2.531 |
| 0.98627 | 0.02083 | 0.02405 | 1.35247 | 0.000858 | 0.000342 | 0.032020 | 2.506 |
| 0.98990 | 0.01766 | 0.02228 | 1.34602 | 0.000786 | 0.000316 | 0.035482 | 2.488 |
| 0.99321 | 0.01485 | 0.02054 | 1.33675 | 0.000717 | 0.000291 | 0.039525 | 2.466 |
| 0.99625 | 0.01234 | 0.01882 | 1.32348 | 0.000652 | 0.000267 | 0.044209 | 2.440 |
| 0.99906 | 0.01010 | 0.01712 | 1.30462 | 0.000593 | 0.000246 | 0.049299 | 2.413 |
| 1.00167 | 0.00812 | 0.01542 | 1.27914 | 0.000543 | 0.000226 | 0.053299 | 2.403 |
| 1.00412 | 0.00636 | 0.01372 | 1.24712 | 0.000486 | 0.000206 | 0.060660 | 2.357 |
| 1.00641 | 0.00483 | 0.01201 | 1.20180 | 0.000448 | 0.000191 | 0.063948 | 2.349 |
| 1.00857 | 0.00352 | 0.01030 | 1.14826 | 0.000409 | 0.000175 | 0.068123 | 2.332 |
| 1.01062 | 0.00241 | 0.00857 | 1.07959 | 0.000369 | 0.000161 | 0.073745 | 2.290 |
| 1.01257 | 0.00152 | 0.00683 | 0.99105 | 0.000345 | 0.000152 | 0.073166 | 2.279 |
| 1.01444 | 0.00084 | 0.00509 | 0.89082 | 0.000324 | 0.000143 | 0.070528 | 2.272 |
| 1.01623 | 0.00036 | 0.00337 | 0.77506 | 0.000304 | 0.000136 | 0.067407 | 2.239 |
| 1.01796 | 0.00009 | 0.00166 | 0.64503 | 0.000298 | 0.000133 | 0.057226 | 2.238 |
| 1.01962 | 0.00000 | 0.00000 | 0.51355 | 0.000294 | 0.000132 | 0.046658 | 2.229 |
| 1.02129 | 0.00009 | -0.00166 | 0.37833 | 0.000293 | 0.000131 | 0.034265 | 2.233 |
| 1.02301 | 0.00036 | -0.00337 | 0.23690 | 0.000294 | 0.000132 | 0.021630 | 2.222 |
| 1.02481 | 0.00084 | -0.00509 | 0.09494 | 0.000307 | 0.000137 | 0.008254 | 2.230 |
| 1.02667 | 0.00152 | -0.00683 | -0.04470 | 0.000307 | 0.000137 | 0.003891 | 2.230 |
| 1.02863 | 0.00241 | -0.00857 | -0.16715 | 0.000350 | 0.000156 | 0.012503 | 2.248 |
| 1.03068 | 0.00352 | -0.01030 | -0.27834 | 0.000361 | 0.000162 | 0.020627 | 2.226 |
| 1.03284 | 0.00483 | -0.01201 | -0.38311 | 0.000384 | 0.000170 | 0.025947 | 2.256 |
| 1.03513 | 0.00636 | -0.01372 | -0.47061 | 0.000424 | 0.000186 | 0.028447 | 2.274 |
| 1.03757 | 0.00812 | -0.01542 | -0.54761 | 0.000456 | 0.000201 | 0.030831 | 2.270 |
| 1.04019 | 0.01010 | -0.01712 | -0.61458 | 0.000499 | 0.000218 | 0.031053 | 2.291 |
| 1.04300 | 0.01234 | -0.01882 | -0.67380 | 0.000538 | 0.000234 | 0.031520 | 2.293 |
| 1.04604 | 0.01485 | -0.02054 | -0.72574 | 0.000585 | 0.000253 | 0.030693 | 2.310 |
| 1.04935 | 0.01766 | -0.02228 | -0.77084 | 0.000636 | 0.000274 | 0.029633 | 2.323 |
| 1.05297 | 0.02083 | -0.02405 | -0.81009 | 0.000693 | 0.000297 | 0.028214 | 2.337 |
| 1.05697 | 0.02439 | -0.02586 | -0.84588 | 0.000748 | 0.000320 | 0.027193 | 2.341 |
| 1.06141 | 0.02842 | -0.02772 | -0.87903 | 0.000806 | 0.000344 | 0.026042 | 2.347 |
| 1.06636 | 0.03298 | -0.02964 | -0.90735 | 0.000881 | 0.000372 | 0.024048 | 2.370 |
| 1.07191 | 0.03817 | -0.03162 | -0.93368 | 0.000950 | 0.000400 | 0.022922 | 2.372 |
| 1.07817 | 0.04408 | -0.03367 | -0.95717 | 0.001032 | 0.000433 | 0.021302 | 2.387 |
| 1.08522 | 0.05081 | -0.03579 | -0.97816 | 0.001120 | 0.000467 | 0.019847 | 2.398 |
| 1.09316 | 0.05845 | -0.03796 | -0.99700 | 0.001214 | 0.000504 | 0.018476 | 2.408 |
| 1.10204 | 0.06704 | -0.04016 | -1.01396 | 0.001313 | 0.000544 | 0.017224 | 2.416 |
| 1.11184 | 0.07660 | -0.04235 | -1.02852 | 0.001423 | 0.000586 | 0.015916 | 2.429 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.12251 | 0.08706 | -0.04449 | -1.04103 | 0.001536 | 0.000630 | 0.014768 | 2.439 |
| 1.13396 | 0.09832 | -0.04654 | -1.05168 | 0.001652 | 0.000675 | 0.013740 | 2.449 |
| 1.14605 | 0.11025 | -0.04847 | -1.06041 | 0.001773 | 0.000721 | 0.012755 | 2.460 |
| 1.15865 | 0.12272 | -0.05025 | -1.06765 | 0.001893 | 0.000767 | 0.011915 | 2.469 |
| 1.17165 | 0.13563 | -0.05188 | -1.07368 | 0.002012 | 0.000812 | 0.011183 | 2.477 |
| 1.18497 | 0.14887 | -0.05334 | -1.07825 | 0.002137 | 0.000859 | 0.010434 | 2.489 |
| 1.19854 | 0.16237 | -0.05463 | -1.08191 | 0.002259 | 0.000904 | 0.009813 | 2.498 |
| 1.21231 | 0.17609 | -0.05577 | -1.08473 | 0.002380 | 0.000949 | 0.009243 | 2.508 |
| 1.22624 | 0.18999 | -0.05676 | -1.08674 | 0.002503 | 0.000994 | 0.008704 | 2.519 |
| 1.24031 | 0.20403 | -0.05760 | -1.08826 | 0.002622 | 0.001038 | 0.008246 | 2.527 |
| 1.25449 | 0.21820 | -0.05831 | -1.08904 | 0.002748 | 0.001082 | 0.007768 | 2.539 |
| 1.26878 | 0.23248 | -0.05888 | -1.08953 | 0.002868 | 0.001126 | 0.007379 | 2.548 |
| 1.28315 | 0.24684 | -0.05933 | -1.08952 | 0.002991 | 0.001169 | 0.006987 | 2.559 |
| 1.29761 | 0.26129 | -0.05966 | -1.08918 | 0.003114 | 0.001212 | 0.006630 | 2.569 |
| 1.31213 | 0.27581 | -0.05988 | -1.08850 | 0.003240 | 0.001255 | 0.006289 | 2.581 |
| 1.32672 | 0.29040 | -0.06000 | -1.08759 | 0.003363 | 0.001298 | 0.005980 | 2.591 |
| 1.34136 | 0.30505 | -0.06001 | -1.08643 | 0.003489 | 0.001341 | 0.005684 | 2.603 |
| 1.35606 | 0.31974 | -0.05993 | -1.08510 | 0.003614 | 0.001383 | 0.005413 | 2.613 |
| 1.37081 | 0.33449 | -0.05976 | -1.08353 | 0.003743 | 0.001425 | 0.005145 | 2.626 |
| 1.38560 | 0.34928 | -0.05950 | -1.08184 | 0.003871 | 0.001468 | 0.004898 | 2.638 |
| 1.40043 | 0.36411 | -0.05916 | -1.07996 | 0.004002 | 0.001510 | 0.004658 | 2.651 |
| 1.41531 | 0.37898 | -0.05874 | -1.07800 | 0.004132 | 0.001552 | 0.004438 | 2.663 |
| 1.43021 | 0.39388 | -0.05825 | -1.07591 | 0.004265 | 0.001594 | 0.004224 | 2.676 |
| 1.44516 | 0.40881 | -0.05769 | -1.07373 | 0.004399 | 0.001636 | 0.004021 | 2.689 |
| 1.46013 | 0.42377 | -0.05706 | -1.07142 | 0.004538 | 0.001678 | 0.003821 | 2.704 |
| 1.47513 | 0.43875 | -0.05637 | -1.06907 | 0.004675 | 0.001720 | 0.003638 | 2.718 |
| 1.49016 | 0.45376 | -0.05561 | -1.06662 | 0.004817 | 0.001762 | 0.003455 | 2.733 |
| 1.50521 | 0.46879 | -0.05480 | -1.06410 | 0.004962 | 0.001805 | 0.003279 | 2.749 |
| 1.52029 | 0.48384 | -0.05393 | -1.06156 | 0.005105 | 0.001847 | 0.003118 | 2.764 |
| 1.53539 | 0.49892 | -0.05301 | -1.05891 | 0.005257 | 0.001890 | 0.002949 | 2.782 |
| 1.55051 | 0.51400 | -0.05204 | -1.05625 | 0.005409 | 0.001932 | 0.002795 | 2.800 |
| 1.56565 | 0.52911 | -0.05101 | -1.05350 | 0.005568 | 0.001975 | 0.002637 | 2.819 |
| 1.58080 | 0.54423 | -0.04994 | -1.05073 | 0.005729 | 0.002018 | 0.002488 | 2.839 |
| 1.59597 | 0.55936 | -0.04883 | -1.04792 | 0.005895 | 0.002061 | 0.002342 | 2.860 |
| 1.61116 | 0.57450 | -0.04767 | -1.04505 | 0.006069 | 0.002104 | 0.002194 | 2.884 |
| 1.62636 | 0.58965 | -0.04647 | -1.04216 | 0.006248 | 0.002148 | 0.002052 | 2.908 |
| 1.64158 | 0.60482 | -0.04524 | -1.03920 | 0.006437 | 0.002192 | 0.001906 | 2.936 |
| 1.65680 | 0.61999 | -0.04396 | -1.03623 | 0.006632 | 0.002237 | 0.001766 | 2.965 |
| 1.67204 | 0.63517 | -0.04264 | -1.03323 | 0.006837 | 0.002281 | 0.001625 | 2.997 |
| 1.68728 | 0.65035 | -0.04129 | -1.03019 | 0.007053 | 0.002327 | 0.001485 | 3.032 |
| 1.70254 | 0.66554 | -0.03991 | -1.02714 | 0.007282 | 0.002372 | 0.001345 | 3.070 |
| 1.71780 | 0.68074 | -0.03849 | -1.02406 | 0.007526 | 0.002418 | 0.001204 | 3.113 |
| 1.73306 | 0.69593 | -0.03703 | -1.02099 | 0.007785 | 0.002464 | 0.001064 | 3.160 |
| 1.74834 | 0.71113 | -0.03555 | -1.01792 | 0.008061 | 0.002510 | 0.000925 | 3.212 |
| 1.76361 | 0.72634 | -0.03403 | -1.01486 | 0.008359 | 0.002556 | 0.000786 | 3.270 |
| 1.77889 | 0.74154 | -0.03248 | -1.01185 | 0.008676 | 0.002602 | 0.000651 | 3.334 |
| 1.79417 | 0.75674 | -0.03090 | -1.00888 | 0.009020 | 0.002648 | 0.000517 | 3.406 |
| 1.80946 | 0.77193 | -0.02929 | -1.00600 | 0.009388 | 0.002693 | 0.000389 | 3.486 |
| 1.82474 | 0.78713 | -0.02765 | -1.00320 | 0.009784 | 0.002737 | 0.000266 | 3.575 |
| 1.84002 | 0.80231 | -0.02597 | -1.00052 | 0.010211 | 0.002780 | 0.000150 | 3.673 |
| 1.85529 | 0.81749 | -0.02427 | -0.99796 | 0.010670 | 0.002822 | 0.000043 | 3.781 |
| 1.87054 | 0.83264 | -0.02254 | -0.99554 | 0.011160 | 0.002862 | -0.000052 | 3.899 |
| 1.88576 | 0.84776 | -0.02079 | -0.99325 | 0.011683 | 0.002901 | -0.000136 | 4.027 |
| 1.90092 | 0.86282 | -0.01901 | -0.99111 | 0.012237 | 0.002938 | -0.000205 | 4.165 |
| 1.91596 | 0.87775 | -0.01721 | -0.98909 | 0.012821 | 0.002974 | -0.000260 | 4.311 |
| 1.93077 | 0.89245 | -0.01541 | -0.98721 | 0.013427 | 0.003008 | -0.000302 | 4.463 |
| 1.94516 | 0.90673 | -0.01364 | -0.98550 | 0.014048 | 0.003040 | -0.000331 | 4.621 |
| 1.95886 | 0.92032 | -0.01192 | -0.98399 | 0.014661 | 0.003069 | -0.000349 | 4.777 |
| 1.97149 | 0.93285 | -0.01031 | -0.98268 | 0.015250 | 0.003095 | -0.000358 | 4.927 |
| 1.98273 | 0.94400 | -0.00886 | -0.98160 | 0.015787 | 0.003117 | -0.000362 | 5.065 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.99243 | 0.95361 | -0.00759 | -0.98072 | 0.016260 | 0.003135 | -0.000363 | 5.186 |
| 2.00063 | 0.96174 | -0.00650 | -0.98001 | 0.016665 | 0.003150 | -0.000362 | 5.291 |
| 2.00754 | 0.96858 | -0.00558 | -0.97944 | 0.017006 | 0.003162 | -0.000361 | 5.378 |
| 2.01337 | 0.97436 | -0.00480 | -0.97897 | 0.017294 | 0.003172 | -0.000360 | 5.452 |
| 2.01835 | 0.97930 | -0.00413 | -0.97858 | 0.017540 | 0.003181 | -0.000359 | 5.515 |
| 2.02264 | 0.98355 | -0.00354 | -0.97825 | 0.017750 | 0.003188 | -0.000358 | 5.568 |
| 2.02640 | 0.98727 | -0.00303 | -0.97797 | 0.017930 | 0.003194 | -0.000356 | 5.613 |
| 2.02971 | 0.99055 | -0.00258 | -0.97773 | 0.018090 | 0.003200 | -0.000355 | 5.654 |
| 2.03267 | 0.99349 | -0.00217 | -0.97751 | 0.018226 | 0.003205 | -0.000354 | 5.687 |
| 2.03533 | 0.99612 | -0.00180 | -0.97732 | 0.018345 | 0.003209 | -0.000353 | 5.717 |
| 2.03775 | 0.99852 | -0.00147 | -0.97714 | 0.018449 | 0.003213 | -0.000352 | 5.742 |
| 2.03925 | 1.00000 | -0.00126 | -0.97703 | 0.018475 | 0.003216 | -0.000201 | 5.745 |
| 2.03925 | 1.00010 | -0.00000 | 0.97703 | 0.043885 | 0.010240 | 0.000000 | 4.286 |
| 2.04074 | 1.00160 | 0.00000 | 0.97572 | 0.043652 | 0.010326 | 0.000000 | 4.227 |
| 2.04259 | 1.00344 | 0.00000 | 0.97419 | 0.043323 | 0.010427 | 0.000000 | 4.155 |
| 2.04487 | 1.00572 | 0.00001 | 0.97205 | 0.042890 | 0.010568 | 0.000000 | 4.059 |
| 2.04769 | 1.00854 | 0.00001 | 0.96883 | 0.042355 | 0.010780 | 0.000000 | 3.929 |
| 2.05116 | 1.01202 | 0.00002 | 0.96479 | 0.041676 | 0.011047 | 0.000000 | 3.773 |
| 2.05546 | 1.01631 | 0.00003 | 0.96017 | 0.040809 | 0.011353 | 0.000000 | 3.595 |
| 2.06076 | 1.02161 | 0.00005 | 0.95511 | 0.039709 | 0.011687 | 0.000000 | 3.398 |
| 2.06731 | 1.02816 | 0.00008 | 0.94936 | 0.038343 | 0.012066 | 0.000000 | 3.178 |
| 2.07539 | 1.03624 | 0.00012 | 0.94307 | 0.036663 | 0.012479 | 0.000000 | 2.938 |
| 2.08538 | 1.04623 | 0.00017 | 0.93660 | 0.034629 | 0.012899 | 0.000000 | 2.685 |
| 2.09771 | 1.05856 | 0.00025 | 0.93058 | 0.032222 | 0.013284 | 0.000000 | 2.426 |
| 2.11293 | 1.07378 | 0.00036 | 0.92597 | 0.029470 | 0.013571 | 0.000000 | 2.172 |
| 2.13173 | 1.09258 | 0.00051 | 0.92392 | 0.026472 | 0.013693 | 0.000000 | 1.933 |
| 2.15495 | 1.11580 | 0.00072 | 0.92547 | 0.023414 | 0.013605 | 0.000000 | 1.721 |
| 2.18361 | 1.14446 | 0.00101 | 0.93089 | 0.020533 | 0.013320 | 0.000000 | 1.542 |
| 2.21901 | 1.17986 | 0.00140 | 0.93933 | 0.018036 | 0.012909 | 0.000000 | 1.397 |
| 2.26273 | 1.22357 | 0.00195 | 0.94916 | 0.016026 | 0.012468 | 0.000000 | 1.285 |
| 2.31671 | 1.27755 | 0.00268 | 0.95885 | 0.014488 | 0.012064 | 0.000000 | 1.201 |
| 2.38337 | 1.34420 | 0.00369 | 0.96751 | 0.013350 | 0.011725 | 0.000000 | 1.139 |
| 2.46568 | 1.42650 | 0.00504 | 0.97485 | 0.012526 | 0.011452 | 0.000000 | 1.094 |
| 2.56733 | 1.52813 | 0.00686 | 0.98086 | 0.011942 | 0.011237 | 0.000000 | 1.063 |
| 2.69285 | 1.65363 | 0.00929 | 0.98566 | 0.011534 | 0.011071 | 0.000000 | 1.042 |
| 2.84785 | 1.80859 | 0.01253 | 0.98941 | 0.011252 | 0.010944 | 0.000000 | 1.028 |
| 3.03925 | 1.99995 | 0.01680 | 0.99265 | 0.011046 | 0.010837 | 0.000000 | 1.019 |

1.00000  0.0
0.99681  0.00031
0.98745  0.00132
0.97235  0.00310
0.95193  0.00547
0.92639  0.00824
0.89600  0.01139
0.86112  0.01494
0.82224  0.01884
0.77985  0.02304
0.73449  0.02744
0.68673  0.03197
0.63717  0.03649
0.58641  0.04086
0.53499  0.04494
0.48350  0.04859
0.43249  0.05171
0.38250  0.05415
0.33405  0.05581
0.28760  0.05658
0.24358  0.05639
0.20240  0.05518
0.16442  0.05292
0.12993  0.04961
0.09921  0.04526
0.07244  0.03993
0.04978  0.03372
0.03130  0.02677
0.01702  0.01932
0.00697  0.01172
0.00127  0.00438
0.00025 -0.00186
0.00457 -0.00741
0.01408 -0.01285
0.02839 -0.01759
0.04763 -0.02141
0.07182 -0.02438
0.10073 -0.02660
0.13407 -0.02809
0.17150 -0.02888
0.21268 -0.02900
0.25719 -0.02852
0.30456 -0.02752
0.35426 -0.02608
0.40572 -0.02428
0.45837 -0.02217
0.51161 -0.01980
0.56484 -0.01723
0.61748 -0.01450
0.66898 -0.01167
0.71883 -0.00887
0.76644 -0.00628
0.81118 -0.00403
0.85241 -0.00220
0.88957 -0.00082
0.92210  0.00008
0.94952  0.00052
0.97134  0.00057
0.98718  0.00037
0.99679  0.00011

1.00001 -0.00000

| # s | x | y | Ue/Vinf | Dstar | Theta | Cf | H |
|---|---|---|---|---|---|---|---|
| 0.00000 | 1.00000 | 0.00000 | 0.95112 | 0.016977 | 0.007284 | 0.001360 | 2.331 |
| 0.00100 | 0.99901 | 0.00009 | 0.95248 | 0.016816 | 0.007238 | 0.001385 | 2.323 |
| 0.00265 | 0.99736 | 0.00026 | 0.95357 | 0.016779 | 0.007201 | 0.001380 | 2.330 |
| 0.00455 | 0.99547 | 0.00045 | 0.95491 | 0.016721 | 0.007156 | 0.001378 | 2.337 |
| 0.00677 | 0.99327 | 0.00068 | 0.95633 | 0.016685 | 0.007109 | 0.001369 | 2.347 |
| 0.00938 | 0.99067 | 0.00096 | 0.95806 | 0.016634 | 0.007051 | 0.001360 | 2.359 |
| 0.01250 | 0.98757 | 0.00131 | 0.96013 | 0.016583 | 0.006983 | 0.001346 | 2.375 |
| 0.01629 | 0.98380 | 0.00174 | 0.96263 | 0.016523 | 0.006901 | 0.001329 | 2.394 |
| 0.02098 | 0.97914 | 0.00229 | 0.96574 | 0.016444 | 0.006800 | 0.001308 | 2.418 |
| 0.02676 | 0.97341 | 0.00297 | 0.96959 | 0.016367 | 0.006677 | 0.001275 | 2.451 |
| 0.03371 | 0.96650 | 0.00379 | 0.97423 | 0.016318 | 0.006531 | 0.001223 | 2.498 |
| 0.04175 | 0.95851 | 0.00472 | 0.97969 | 0.016309 | 0.006363 | 0.001147 | 2.563 |
| 0.05054 | 0.94978 | 0.00571 | 0.98578 | 0.016371 | 0.006179 | 0.001041 | 2.649 |
| 0.05978 | 0.94060 | 0.00672 | 0.99249 | 0.016497 | 0.005981 | 0.000909 | 2.758 |
| 0.06932 | 0.93111 | 0.00774 | 0.99986 | 0.016698 | 0.005767 | 0.000751 | 2.895 |
| 0.07905 | 0.92144 | 0.00876 | 1.00805 | 0.016948 | 0.005534 | 0.000574 | 3.062 |
| 0.08888 | 0.91166 | 0.00978 | 1.01717 | 0.017241 | 0.005281 | 0.000386 | 3.265 |
| 0.09876 | 0.90183 | 0.01079 | 1.02734 | 0.017563 | 0.005004 | 0.000195 | 3.510 |
| 0.10867 | 0.89198 | 0.01180 | 1.03868 | 0.017896 | 0.004702 | 0.000013 | 3.806 |
| 0.11859 | 0.88211 | 0.01281 | 1.05154 | 0.018228 | 0.004368 | -0.000114 | 4.173 |
| 0.12851 | 0.87224 | 0.01381 | 1.06600 | 0.018537 | 0.004003 | -0.000188 | 4.631 |
| 0.13843 | 0.86237 | 0.01481 | 1.08040 | 0.018829 | 0.003650 | -0.000231 | 5.159 |
| 0.14836 | 0.85249 | 0.01581 | 1.09463 | 0.019077 | 0.003312 | -0.000254 | 5.760 |
| 0.15829 | 0.84261 | 0.01680 | 1.10878 | 0.019256 | 0.002985 | -0.000267 | 6.450 |
| 0.16822 | 0.83272 | 0.01779 | 1.11560 | 0.019438 | 0.002832 | -0.000427 | 6.863 |
| 0.17816 | 0.82283 | 0.01878 | 1.11651 | 0.019598 | 0.002813 | -0.000428 | 6.966 |
| 0.18810 | 0.81294 | 0.01977 | 1.11740 | 0.019691 | 0.002795 | -0.000430 | 7.045 |
| 0.19804 | 0.80306 | 0.02075 | 1.11825 | 0.019715 | 0.002778 | -0.000432 | 7.098 |
| 0.20797 | 0.79318 | 0.02173 | 1.11909 | 0.019699 | 0.002760 | -0.000434 | 7.136 |
| 0.21789 | 0.78330 | 0.02270 | 1.11992 | 0.019640 | 0.002743 | -0.000437 | 7.159 |
| 0.22781 | 0.77343 | 0.02367 | 1.12074 | 0.019542 | 0.002727 | -0.000439 | 7.166 |
| 0.23773 | 0.76355 | 0.02463 | 1.12155 | 0.019418 | 0.002710 | -0.000442 | 7.164 |
| 0.24765 | 0.75368 | 0.02559 | 1.12237 | 0.019262 | 0.002694 | -0.000446 | 7.149 |
| 0.25758 | 0.74380 | 0.02654 | 1.12318 | 0.019080 | 0.002678 | -0.000449 | 7.124 |
| 0.26751 | 0.73391 | 0.02750 | 1.12400 | 0.018871 | 0.002662 | -0.000453 | 7.088 |
| 0.27744 | 0.72403 | 0.02844 | 1.12483 | 0.018634 | 0.002646 | -0.000457 | 7.042 |
| 0.28736 | 0.71415 | 0.02938 | 1.12567 | 0.018388 | 0.002630 | -0.000461 | 6.991 |
| 0.29728 | 0.70427 | 0.03032 | 1.12652 | 0.018119 | 0.002614 | -0.000465 | 6.930 |
| 0.30720 | 0.69440 | 0.03125 | 1.12738 | 0.017833 | 0.002599 | -0.000470 | 6.863 |
| 0.31710 | 0.68454 | 0.03217 | 1.12826 | 0.017532 | 0.002582 | -0.000475 | 6.789 |
| 0.32700 | 0.67468 | 0.03309 | 1.12917 | 0.017214 | 0.002566 | -0.000480 | 6.708 |
| 0.33690 | 0.66482 | 0.03399 | 1.13009 | 0.016886 | 0.002550 | -0.000486 | 6.622 |
| 0.34680 | 0.65497 | 0.03489 | 1.13105 | 0.016555 | 0.002533 | -0.000492 | 6.535 |
| 0.35669 | 0.64512 | 0.03578 | 1.13204 | 0.016209 | 0.002517 | -0.000498 | 6.441 |
| 0.36658 | 0.63527 | 0.03666 | 1.13306 | 0.015851 | 0.002500 | -0.000505 | 6.341 |
| 0.37647 | 0.62542 | 0.03753 | 1.13412 | 0.015500 | 0.002482 | -0.000512 | 6.244 |
| 0.38635 | 0.61557 | 0.03839 | 1.13522 | 0.015136 | 0.002465 | -0.000520 | 6.142 |
| 0.39623 | 0.60573 | 0.03923 | 1.13636 | 0.014758 | 0.002447 | -0.000528 | 6.032 |
| 0.40611 | 0.59588 | 0.04007 | 1.13756 | 0.014392 | 0.002428 | -0.000536 | 5.927 |
| 0.41598 | 0.58605 | 0.04089 | 1.13881 | 0.014017 | 0.002409 | -0.000545 | 5.818 |
| 0.42585 | 0.57621 | 0.04170 | 1.14012 | 0.013642 | 0.002390 | -0.000553 | 5.708 |
| 0.43571 | 0.56638 | 0.04250 | 1.14150 | 0.013261 | 0.002370 | -0.000561 | 5.595 |
| 0.44557 | 0.55656 | 0.04328 | 1.14294 | 0.012893 | 0.002350 | -0.000567 | 5.487 |
| 0.45542 | 0.54674 | 0.04404 | 1.14446 | 0.012515 | 0.002329 | -0.000573 | 5.374 |
| 0.46527 | 0.53692 | 0.04479 | 1.14606 | 0.012146 | 0.002307 | -0.000578 | 5.264 |
| 0.47511 | 0.52710 | 0.04553 | 1.14774 | 0.011779 | 0.002285 | -0.000582 | 5.154 |
| 0.48496 | 0.51728 | 0.04625 | 1.14951 | 0.011408 | 0.002262 | -0.000583 | 5.042 |
| 0.49480 | 0.50746 | 0.04695 | 1.15138 | 0.011058 | 0.002239 | -0.000581 | 4.939 |
| 0.50464 | 0.49764 | 0.04763 | 1.15336 | 0.010702 | 0.002215 | -0.000575 | 4.832 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.51448 | 0.48783 | 0.04830 | 1.15544 | 0.010357 | 0.002190 | -0.000564 | 4.729 |
| 0.52432 | 0.47801 | 0.04895 | 1.15763 | 0.010022 | 0.002165 | -0.000548 | 4.630 |
| 0.53415 | 0.46820 | 0.04958 | 1.15994 | 0.009661 | 0.002138 | -0.000520 | 4.518 |
| 0.54397 | 0.45840 | 0.05019 | 1.16237 | 0.009356 | 0.002111 | -0.000492 | 4.431 |
| 0.55378 | 0.44861 | 0.05079 | 1.16491 | 0.009020 | 0.002084 | -0.000448 | 4.329 |
| 0.56358 | 0.43882 | 0.05135 | 1.16758 | 0.008695 | 0.002056 | -0.000393 | 4.230 |
| 0.57337 | 0.42904 | 0.05190 | 1.17036 | 0.008401 | 0.002027 | -0.000335 | 4.144 |
| 0.58316 | 0.41927 | 0.05242 | 1.17326 | 0.008096 | 0.001998 | -0.000258 | 4.053 |
| 0.59295 | 0.40950 | 0.05292 | 1.17630 | 0.007807 | 0.001968 | -0.000173 | 3.968 |
| 0.60273 | 0.39973 | 0.05339 | 1.17946 | 0.007543 | 0.001937 | -0.000087 | 3.895 |
| 0.61250 | 0.38997 | 0.05383 | 1.18279 | 0.007250 | 0.001905 | 0.000038 | 3.805 |
| 0.62227 | 0.38021 | 0.05425 | 1.18625 | 0.007000 | 0.001873 | 0.000149 | 3.737 |
| 0.63203 | 0.37045 | 0.05463 | 1.18987 | 0.006737 | 0.001840 | 0.000291 | 3.661 |
| 0.64178 | 0.36071 | 0.05499 | 1.19361 | 0.006500 | 0.001807 | 0.000428 | 3.597 |
| 0.65152 | 0.35098 | 0.05532 | 1.19749 | 0.006268 | 0.001773 | 0.000580 | 3.536 |
| 0.66125 | 0.34125 | 0.05561 | 1.20155 | 0.006030 | 0.001738 | 0.000762 | 3.470 |
| 0.67096 | 0.33154 | 0.05587 | 1.20569 | 0.005817 | 0.001703 | 0.000931 | 3.416 |
| 0.68067 | 0.32184 | 0.05610 | 1.20998 | 0.005605 | 0.001667 | 0.001123 | 3.362 |
| 0.69036 | 0.31215 | 0.05628 | 1.21429 | 0.005425 | 0.001632 | 0.001279 | 3.324 |
| 0.70004 | 0.30247 | 0.05643 | 1.21888 | 0.005216 | 0.001595 | 0.001513 | 3.270 |
| 0.70972 | 0.29279 | 0.05654 | 1.22363 | 0.005005 | 0.001557 | 0.001783 | 3.214 |
| 0.71938 | 0.28313 | 0.05661 | 1.22808 | 0.004867 | 0.001522 | 0.001904 | 3.197 |
| 0.72902 | 0.27349 | 0.05663 | 1.23305 | 0.004673 | 0.001485 | 0.002189 | 3.148 |
| 0.73865 | 0.26386 | 0.05660 | 1.23793 | 0.004508 | 0.001448 | 0.002421 | 3.114 |
| 0.74827 | 0.25424 | 0.05653 | 1.24287 | 0.004354 | 0.001411 | 0.002645 | 3.086 |
| 0.75786 | 0.24465 | 0.05641 | 1.24813 | 0.004178 | 0.001372 | 0.002971 | 3.045 |
| 0.76743 | 0.23508 | 0.05623 | 1.25318 | 0.004036 | 0.001335 | 0.003202 | 3.023 |
| 0.77698 | 0.22554 | 0.05600 | 1.25870 | 0.003864 | 0.001296 | 0.003591 | 2.981 |
| 0.78651 | 0.21601 | 0.05571 | 1.26373 | 0.003740 | 0.001260 | 0.003795 | 2.969 |
| 0.79601 | 0.20652 | 0.05535 | 1.26950 | 0.003571 | 0.001220 | 0.004250 | 2.928 |
| 0.80548 | 0.19706 | 0.05494 | 1.27482 | 0.003440 | 0.001182 | 0.004552 | 2.909 |
| 0.81493 | 0.18762 | 0.05445 | 1.28059 | 0.003287 | 0.001143 | 0.005020 | 2.876 |
| 0.82434 | 0.17823 | 0.05389 | 1.28552 | 0.003192 | 0.001107 | 0.005141 | 2.883 |
| 0.83372 | 0.16887 | 0.05325 | 1.29236 | 0.002999 | 0.001063 | 0.005997 | 2.820 |
| 0.84305 | 0.15956 | 0.05254 | 1.29756 | 0.002885 | 0.001026 | 0.006330 | 2.812 |
| 0.85234 | 0.15031 | 0.05173 | 1.30324 | 0.002757 | 0.000987 | 0.006824 | 2.793 |
| 0.86157 | 0.14113 | 0.05084 | 1.30860 | 0.002647 | 0.000949 | 0.007168 | 2.790 |
| 0.87073 | 0.13201 | 0.04985 | 1.31517 | 0.002494 | 0.000907 | 0.008046 | 2.751 |
| 0.87983 | 0.12299 | 0.04876 | 1.32130 | 0.002348 | 0.000865 | 0.008974 | 2.716 |
| 0.88884 | 0.11405 | 0.04756 | 1.32582 | 0.002256 | 0.000828 | 0.009257 | 2.726 |
| 0.89776 | 0.10523 | 0.04624 | 1.33218 | 0.002111 | 0.000785 | 0.010398 | 2.689 |
| 0.90659 | 0.09652 | 0.04480 | 1.33803 | 0.001965 | 0.000742 | 0.011786 | 2.648 |
| 0.91529 | 0.08796 | 0.04323 | 1.34188 | 0.001856 | 0.000703 | 0.012623 | 2.641 |
| 0.92386 | 0.07957 | 0.04152 | 1.34564 | 0.001748 | 0.000663 | 0.013484 | 2.638 |
| 0.93223 | 0.07139 | 0.03968 | 1.35013 | 0.001619 | 0.000620 | 0.015076 | 2.611 |
| 0.94037 | 0.06350 | 0.03771 | 1.35394 | 0.001474 | 0.000576 | 0.017631 | 2.558 |
| 0.94822 | 0.05594 | 0.03561 | 1.35333 | 0.001372 | 0.000538 | 0.019075 | 2.551 |
| 0.95573 | 0.04876 | 0.03339 | 1.35206 | 0.001268 | 0.000499 | 0.020864 | 2.541 |
| 0.96286 | 0.04202 | 0.03107 | 1.34961 | 0.001161 | 0.000460 | 0.023156 | 2.524 |
| 0.96951 | 0.03581 | 0.02868 | 1.34507 | 0.001051 | 0.000421 | 0.026380 | 2.493 |
| 0.97557 | 0.03024 | 0.02630 | 1.33613 | 0.000951 | 0.000386 | 0.029936 | 2.461 |
| 0.98097 | 0.02536 | 0.02398 | 1.32234 | 0.000874 | 0.000356 | 0.032414 | 2.456 |
| 0.98573 | 0.02116 | 0.02176 | 1.30676 | 0.000786 | 0.000326 | 0.037176 | 2.414 |
| 0.98990 | 0.01755 | 0.01965 | 1.28464 | 0.000739 | 0.000303 | 0.037912 | 2.439 |
| 0.99357 | 0.01447 | 0.01766 | 1.26397 | 0.000648 | 0.000276 | 0.046576 | 2.348 |
| 0.99681 | 0.01184 | 0.01578 | 1.22929 | 0.000632 | 0.000262 | 0.043461 | 2.415 |
| 0.99965 | 0.00961 | 0.01403 | 1.20411 | 0.000566 | 0.000240 | 0.050088 | 2.362 |
| 1.00214 | 0.00772 | 0.01240 | 1.17279 | 0.000556 | 0.000225 | 0.044544 | 2.470 |
| 1.00434 | 0.00611 | 0.01090 | 1.15287 | 0.000468 | 0.000199 | 0.058647 | 2.350 |
| 1.00630 | 0.00475 | 0.00949 | 1.11282 | 0.000444 | 0.000185 | 0.056908 | 2.399 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.00806 | 0.00361 | 0.00815 | 1.07228 | 0.000380 | 0.000166 | 0.071147 | 2.289 |
| 1.00966 | 0.00267 | 0.00685 | 1.01296 | 0.000374 | 0.000157 | 0.062349 | 2.384 |
| 1.01113 | 0.00189 | 0.00560 | 0.95488 | 0.000313 | 0.000140 | 0.080875 | 2.235 |
| 1.01252 | 0.00126 | 0.00436 | 0.86417 | 0.000316 | 0.000137 | 0.067969 | 2.306 |
| 1.01383 | 0.00076 | 0.00316 | 0.79426 | 0.000288 | 0.000121 | 0.064373 | 2.372 |
| 1.01505 | 0.00040 | 0.00199 | 0.68411 | 0.000232 | 0.000109 | 0.086379 | 2.123 |
| 1.01620 | 0.00017 | 0.00086 | 0.54100 | 0.000290 | 0.000125 | 0.046146 | 2.313 |
| 1.01728 | 0.00008 | -0.00021 | 0.45474 | 0.000247 | 0.000109 | 0.047912 | 2.256 |
| 1.01831 | 0.00013 | -0.00124 | 0.33749 | 0.000216 | 0.000094 | 0.039382 | 2.292 |
| 1.01932 | 0.00035 | -0.00223 | 0.16027 | 0.000168 | 0.000075 | 0.025415 | 2.230 |
| 1.02036 | 0.00075 | -0.00319 | -0.09744 | 0.000168 | 0.000075 | 0.015448 | 2.230 |
| 1.02146 | 0.00131 | -0.00414 | -0.17992 | 0.000391 | 0.000156 | 0.009334 | 2.509 |
| 1.02267 | 0.00206 | -0.00509 | -0.25705 | 0.000299 | 0.000141 | 0.025257 | 2.117 |
| 1.02402 | 0.00300 | -0.00606 | -0.38839 | 0.000274 | 0.000123 | 0.037710 | 2.231 |
| 1.02555 | 0.00415 | -0.00707 | -0.47581 | 0.000352 | 0.000149 | 0.031774 | 2.365 |
| 1.02728 | 0.00551 | -0.00813 | -0.56860 | 0.000340 | 0.000155 | 0.045939 | 2.195 |
| 1.02922 | 0.00711 | -0.00923 | -0.62837 | 0.000451 | 0.000185 | 0.030608 | 2.432 |
| 1.03140 | 0.00897 | -0.01036 | -0.67510 | 0.000479 | 0.000208 | 0.035352 | 2.297 |
| 1.03384 | 0.01113 | -0.01150 | -0.75460 | 0.000455 | 0.000206 | 0.045230 | 2.204 |
| 1.03654 | 0.01358 | -0.01264 | -0.78843 | 0.000607 | 0.000245 | 0.027049 | 2.483 |
| 1.03953 | 0.01635 | -0.01377 | -0.82332 | 0.000614 | 0.000265 | 0.032970 | 2.317 |
| 1.04290 | 0.01951 | -0.01494 | -0.86664 | 0.000653 | 0.000282 | 0.032701 | 2.315 |
| 1.04678 | 0.02320 | -0.01614 | -0.88990 | 0.000779 | 0.000319 | 0.024806 | 2.442 |
| 1.05135 | 0.02759 | -0.01738 | -0.91572 | 0.000827 | 0.000348 | 0.025713 | 2.376 |
| 1.05677 | 0.03286 | -0.01867 | -0.93789 | 0.000932 | 0.000385 | 0.022430 | 2.418 |
| 1.06310 | 0.03906 | -0.01995 | -0.95736 | 0.001028 | 0.000424 | 0.020677 | 2.421 |
| 1.07016 | 0.04601 | -0.02116 | -0.97056 | 0.001157 | 0.000470 | 0.017886 | 2.460 |
| 1.07775 | 0.05352 | -0.02227 | -0.97846 | 0.001293 | 0.000519 | 0.015623 | 2.491 |
| 1.08582 | 0.06153 | -0.02328 | -0.98303 | 0.001432 | 0.000569 | 0.013820 | 2.515 |
| 1.09432 | 0.06998 | -0.02420 | -0.99017 | 0.001513 | 0.000610 | 0.013685 | 2.479 |
| 1.10317 | 0.07879 | -0.02502 | -0.99514 | 0.001634 | 0.000655 | 0.012516 | 2.495 |
| 1.11225 | 0.08784 | -0.02574 | -0.99835 | 0.001758 | 0.000700 | 0.011455 | 2.513 |
| 1.12146 | 0.09703 | -0.02637 | -1.00119 | 0.001866 | 0.000742 | 0.010791 | 2.516 |
| 1.13075 | 0.10631 | -0.02692 | -1.00215 | 0.001996 | 0.000786 | 0.009838 | 2.539 |
| 1.14012 | 0.11567 | -0.02738 | -1.00604 | 0.002046 | 0.000818 | 0.010048 | 2.501 |
| 1.14957 | 0.12510 | -0.02778 | -1.00590 | 0.002197 | 0.000863 | 0.008904 | 2.546 |
| 1.15907 | 0.13460 | -0.02811 | -1.00672 | 0.002293 | 0.000901 | 0.008544 | 2.546 |
| 1.16863 | 0.14416 | -0.02838 | -1.00823 | 0.002367 | 0.000934 | 0.008402 | 2.533 |
| 1.17822 | 0.15375 | -0.02860 | -1.00802 | 0.002488 | 0.000973 | 0.007793 | 2.556 |
| 1.18785 | 0.16337 | -0.02877 | -1.00818 | 0.002582 | 0.001009 | 0.007482 | 2.560 |
| 1.19749 | 0.17302 | -0.02890 | -1.00815 | 0.002677 | 0.001044 | 0.007180 | 2.564 |
| 1.20717 | 0.18269 | -0.02898 | -1.00840 | 0.002757 | 0.001076 | 0.006996 | 2.561 |
| 1.21686 | 0.19238 | -0.02902 | -1.00901 | 0.002825 | 0.001106 | 0.006896 | 2.553 |
| 1.22658 | 0.20210 | -0.02903 | -1.00665 | 0.002996 | 0.001149 | 0.006106 | 2.606 |
| 1.23631 | 0.21184 | -0.02900 | -1.00815 | 0.003000 | 0.001172 | 0.006438 | 2.560 |
| 1.24607 | 0.22159 | -0.02895 | -1.00721 | 0.003116 | 0.001207 | 0.006040 | 2.582 |
| 1.25584 | 0.23136 | -0.02886 | -1.00626 | 0.003221 | 0.001240 | 0.005745 | 2.596 |
| 1.26563 | 0.24115 | -0.02875 | -1.00605 | 0.003287 | 0.001269 | 0.005667 | 2.590 |
| 1.27543 | 0.25095 | -0.02862 | -1.00516 | 0.003385 | 0.001301 | 0.005430 | 2.601 |
| 1.28523 | 0.26075 | -0.02846 | -1.00511 | 0.003440 | 0.001328 | 0.005409 | 2.590 |
| 1.29505 | 0.27057 | -0.02828 | -1.00436 | 0.003531 | 0.001358 | 0.005213 | 2.599 |
| 1.30488 | 0.28039 | -0.02808 | -1.00329 | 0.003632 | 0.001390 | 0.004984 | 2.613 |
| 1.31471 | 0.29023 | -0.02787 | -1.00295 | 0.003691 | 0.001416 | 0.004941 | 2.606 |
| 1.32456 | 0.30007 | -0.02763 | -1.00258 | 0.003755 | 0.001443 | 0.004874 | 2.603 |
| 1.33441 | 0.30991 | -0.02738 | -1.00179 | 0.003842 | 0.001472 | 0.004718 | 2.610 |
| 1.34426 | 0.31977 | -0.02712 | -1.00078 | 0.003936 | 0.001501 | 0.004541 | 2.622 |
| 1.35413 | 0.32962 | -0.02684 | -1.00029 | 0.003997 | 0.001527 | 0.004488 | 2.618 |
| 1.36399 | 0.33949 | -0.02654 | -0.99928 | 0.004089 | 0.001556 | 0.004331 | 2.628 |
| 1.37387 | 0.34936 | -0.02624 | -0.99927 | 0.004120 | 0.001577 | 0.004379 | 2.612 |
| 1.38375 | 0.35923 | -0.02592 | -0.99780 | 0.004243 | 0.001609 | 0.004128 | 2.636 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.39363 | 0.36911 | -0.02559 | -0.99744 | 0.004290 | 0.001633 | 0.004124 | 2.627 |
| 1.40352 | 0.37899 | -0.02525 | -0.99741 | 0.004321 | 0.001653 | 0.004163 | 2.613 |
| 1.41340 | 0.38887 | -0.02490 | -0.99585 | 0.004452 | 0.001686 | 0.003906 | 2.641 |
| 1.42329 | 0.39875 | -0.02454 | -0.99544 | 0.004500 | 0.001708 | 0.003894 | 2.634 |
| 1.43318 | 0.40864 | -0.02417 | -0.99506 | 0.004548 | 0.001731 | 0.003878 | 2.628 |
| 1.44308 | 0.41852 | -0.02379 | -0.99411 | 0.004637 | 0.001758 | 0.003754 | 2.638 |
| 1.45297 | 0.42841 | -0.02340 | -0.99360 | 0.004693 | 0.001780 | 0.003720 | 2.636 |
| 1.46286 | 0.43830 | -0.02300 | -0.99299 | 0.004756 | 0.001804 | 0.003664 | 2.636 |
| 1.47276 | 0.44818 | -0.02260 | -0.99225 | 0.004830 | 0.001829 | 0.003586 | 2.641 |
| 1.48266 | 0.45807 | -0.02218 | -0.99147 | 0.004904 | 0.001853 | 0.003507 | 2.646 |
| 1.49256 | 0.46797 | -0.02176 | -0.99081 | 0.004968 | 0.001876 | 0.003455 | 2.648 |
| 1.50246 | 0.47786 | -0.02133 | -0.98991 | 0.005050 | 0.001902 | 0.003363 | 2.656 |
| 1.51237 | 0.48775 | -0.02089 | -0.98923 | 0.005113 | 0.001925 | 0.003316 | 2.657 |
| 1.52228 | 0.49765 | -0.02044 | -0.98876 | 0.005160 | 0.001946 | 0.003301 | 2.652 |
| 1.53218 | 0.50755 | -0.01999 | -0.98777 | 0.005250 | 0.001971 | 0.003200 | 2.663 |
| 1.54209 | 0.51745 | -0.01953 | -0.98720 | 0.005303 | 0.001993 | 0.003173 | 2.661 |
| 1.55200 | 0.52735 | -0.01906 | -0.98620 | 0.005393 | 0.002018 | 0.003077 | 2.672 |
| 1.56192 | 0.53725 | -0.01859 | -0.98561 | 0.005444 | 0.002039 | 0.003055 | 2.669 |
| 1.57183 | 0.54715 | -0.01811 | -0.98460 | 0.005534 | 0.002065 | 0.002964 | 2.680 |
| 1.58174 | 0.55705 | -0.01762 | -0.98372 | 0.005610 | 0.002088 | 0.002900 | 2.686 |
| 1.59165 | 0.56695 | -0.01712 | -0.98285 | 0.005684 | 0.002112 | 0.002842 | 2.691 |
| 1.60156 | 0.57684 | -0.01662 | -0.98202 | 0.005753 | 0.002135 | 0.002792 | 2.695 |
| 1.61147 | 0.58674 | -0.01612 | -0.98109 | 0.005832 | 0.002158 | 0.002730 | 2.702 |
| 1.62138 | 0.59664 | -0.01560 | -0.98013 | 0.005912 | 0.002182 | 0.002666 | 2.709 |
| 1.63129 | 0.60653 | -0.01508 | -0.97913 | 0.005995 | 0.002207 | 0.002600 | 2.717 |
| 1.64120 | 0.61643 | -0.01456 | -0.97818 | 0.006072 | 0.002230 | 0.002546 | 2.723 |
| 1.65112 | 0.62633 | -0.01402 | -0.97709 | 0.006164 | 0.002255 | 0.002472 | 2.733 |
| 1.66103 | 0.63624 | -0.01348 | -0.97599 | 0.006254 | 0.002280 | 0.002403 | 2.744 |
| 1.67096 | 0.64614 | -0.01294 | -0.97488 | 0.006344 | 0.002304 | 0.002337 | 2.753 |
| 1.68089 | 0.65606 | -0.01239 | -0.97362 | 0.006451 | 0.002331 | 0.002252 | 2.768 |
| 1.69082 | 0.66597 | -0.01184 | -0.97248 | 0.006542 | 0.002355 | 0.002191 | 2.777 |
| 1.70075 | 0.67589 | -0.01128 | -0.97132 | 0.006636 | 0.002380 | 0.002128 | 2.788 |
| 1.71069 | 0.68581 | -0.01072 | -0.97009 | 0.006736 | 0.002406 | 0.002061 | 2.800 |
| 1.72062 | 0.69573 | -0.01017 | -0.96881 | 0.006843 | 0.002432 | 0.001988 | 2.814 |
| 1.73056 | 0.70565 | -0.00961 | -0.96755 | 0.006947 | 0.002458 | 0.001922 | 2.827 |
| 1.74049 | 0.71557 | -0.00905 | -0.96631 | 0.007047 | 0.002483 | 0.001863 | 2.838 |
| 1.75041 | 0.72548 | -0.00850 | -0.96508 | 0.007145 | 0.002508 | 0.001809 | 2.849 |
| 1.76034 | 0.73538 | -0.00795 | -0.96378 | 0.007255 | 0.002534 | 0.001744 | 2.863 |
| 1.77025 | 0.74528 | -0.00741 | -0.96253 | 0.007355 | 0.002559 | 0.001691 | 2.874 |
| 1.78016 | 0.75518 | -0.00688 | -0.96134 | 0.007447 | 0.002584 | 0.001649 | 2.882 |
| 1.79006 | 0.76507 | -0.00635 | -0.96013 | 0.007545 | 0.002609 | 0.001602 | 2.892 |
| 1.79995 | 0.77495 | -0.00584 | -0.95894 | 0.007638 | 0.002633 | 0.001562 | 2.901 |
| 1.80984 | 0.78482 | -0.00533 | -0.95776 | 0.007731 | 0.002657 | 0.001522 | 2.909 |
| 1.81972 | 0.79468 | -0.00483 | -0.95664 | 0.007814 | 0.002681 | 0.001492 | 2.915 |
| 1.82959 | 0.80454 | -0.00435 | -0.95554 | 0.007896 | 0.002704 | 0.001463 | 2.920 |
| 1.83945 | 0.81439 | -0.00388 | -0.95443 | 0.007980 | 0.002727 | 0.001433 | 2.926 |
| 1.84931 | 0.82424 | -0.00342 | -0.95339 | 0.008053 | 0.002750 | 0.001413 | 2.929 |
| 1.85917 | 0.83409 | -0.00298 | -0.95238 | 0.008122 | 0.002772 | 0.001397 | 2.930 |
| 1.86901 | 0.84393 | -0.00255 | -0.95139 | 0.008190 | 0.002794 | 0.001381 | 2.932 |
| 1.87885 | 0.85376 | -0.00214 | -0.95044 | 0.008250 | 0.002815 | 0.001371 | 2.931 |
| 1.88868 | 0.86358 | -0.00175 | -0.94954 | 0.008306 | 0.002835 | 0.001364 | 2.929 |
| 1.89849 | 0.87338 | -0.00138 | -0.94869 | 0.008354 | 0.002855 | 0.001362 | 2.926 |
| 1.90828 | 0.88317 | -0.00103 | -0.94787 | 0.008398 | 0.002875 | 0.001363 | 2.921 |
| 1.91805 | 0.89293 | -0.00071 | -0.94713 | 0.008431 | 0.002894 | 0.001371 | 2.914 |
| 1.92780 | 0.90267 | -0.00042 | -0.94647 | 0.008454 | 0.002911 | 0.001386 | 2.904 |
| 1.93751 | 0.91238 | -0.00015 | -0.94590 | 0.008461 | 0.002927 | 0.001411 | 2.891 |
| 1.94717 | 0.92204 | 0.00008 | -0.94545 | 0.008456 | 0.002942 | 0.001445 | 2.874 |
| 1.95673 | 0.93159 | 0.00027 | -0.94513 | 0.008432 | 0.002954 | 0.001490 | 2.854 |
| 1.96611 | 0.94098 | 0.00042 | -0.94497 | 0.008390 | 0.002965 | 0.001548 | 2.830 |
| 1.97520 | 0.95007 | 0.00052 | -0.94501 | 0.008327 | 0.002972 | 0.001621 | 2.801 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.98384 | 0.95870 | 0.00058 | -0.94529 | 0.008238 | 0.002976 | 0.001712 | 2.768 |
| 1.99170 | 0.96657 | 0.00059 | -0.94580 | 0.008135 | 0.002976 | 0.001814 | 2.733 |
| 1.99851 | 0.97337 | 0.00056 | -0.94648 | 0.008023 | 0.002973 | 0.001922 | 2.698 |
| 2.00421 | 0.97907 | 0.00050 | -0.94730 | 0.007906 | 0.002968 | 0.002035 | 2.664 |
| 2.00885 | 0.98372 | 0.00044 | -0.94807 | 0.007808 | 0.002962 | 0.002131 | 2.636 |
| 2.01264 | 0.98750 | 0.00036 | -0.94898 | 0.007700 | 0.002953 | 0.002237 | 2.607 |
| 2.01576 | 0.99062 | 0.00029 | -0.94981 | 0.007609 | 0.002945 | 0.002328 | 2.584 |
| 2.01838 | 0.99324 | 0.00022 | -0.95060 | 0.007527 | 0.002937 | 0.002412 | 2.562 |
| 2.02060 | 0.99545 | 0.00015 | -0.95127 | 0.007462 | 0.002931 | 0.002480 | 2.546 |
| 2.02250 | 0.99736 | 0.00009 | -0.95176 | 0.007417 | 0.002927 | 0.002529 | 2.534 |
| 2.02416 | 0.99901 | 0.00004 | -0.95227 | 0.007370 | 0.002922 | 0.002580 | 2.522 |
| 2.02515 | 1.00001 | 0.00000 | -0.95112 | 0.007499 | 0.002940 | 0.002454 | 2.551 |
| 2.02515 | 1.00010 | -0.00001 | 0.95112 | 0.024476 | 0.010224 | 0.000000 | 2.394 |
| 2.02615 | 1.00110 | -0.00005 | 0.95199 | 0.024064 | 0.010183 | 0.000000 | 2.363 |
| 2.02732 | 1.00227 | -0.00011 | 0.95214 | 0.023853 | 0.010176 | 0.000000 | 2.344 |
| 2.02870 | 1.00365 | -0.00018 | 0.95221 | 0.023645 | 0.010173 | 0.000000 | 2.324 |
| 2.03032 | 1.00526 | -0.00027 | 0.95222 | 0.023427 | 0.010172 | 0.000000 | 2.303 |
| 2.03222 | 1.00716 | -0.00036 | 0.95219 | 0.023190 | 0.010174 | 0.000000 | 2.279 |
| 2.03446 | 1.00940 | -0.00047 | 0.95214 | 0.022926 | 0.010176 | 0.000000 | 2.253 |
| 2.03709 | 1.01203 | -0.00060 | 0.95207 | 0.022630 | 0.010179 | 0.000000 | 2.223 |
| 2.04019 | 1.01512 | -0.00074 | 0.95202 | 0.022296 | 0.010181 | 0.000000 | 2.190 |
| 2.04382 | 1.01875 | -0.00090 | 0.95198 | 0.021919 | 0.010183 | 0.000000 | 2.152 |
| 2.04810 | 1.02302 | -0.00108 | 0.95198 | 0.021493 | 0.010183 | 0.000000 | 2.111 |
| 2.05312 | 1.02804 | -0.00129 | 0.95205 | 0.021015 | 0.010180 | 0.000000 | 2.064 |
| 2.05903 | 1.03395 | -0.00152 | 0.95223 | 0.020480 | 0.010172 | 0.000000 | 2.013 |
| 2.06598 | 1.04089 | -0.00177 | 0.95256 | 0.019888 | 0.010158 | 0.000000 | 1.958 |
| 2.07414 | 1.04905 | -0.00205 | 0.95310 | 0.019236 | 0.010136 | 0.000000 | 1.898 |
| 2.08374 | 1.05864 | -0.00236 | 0.95391 | 0.018529 | 0.010102 | 0.000000 | 1.834 |
| 2.09503 | 1.06992 | -0.00270 | 0.95505 | 0.017773 | 0.010057 | 0.000000 | 1.767 |
| 2.10829 | 1.08319 | -0.00307 | 0.95657 | 0.016979 | 0.009997 | 0.000000 | 1.698 |
| 2.12389 | 1.09878 | -0.00346 | 0.95852 | 0.016160 | 0.009923 | 0.000000 | 1.629 |
| 2.14223 | 1.11711 | -0.00388 | 0.96088 | 0.015335 | 0.009835 | 0.000000 | 1.559 |
| 2.16378 | 1.13866 | -0.00431 | 0.96363 | 0.014523 | 0.009737 | 0.000000 | 1.492 |
| 2.18912 | 1.16400 | -0.00476 | 0.96669 | 0.013741 | 0.009631 | 0.000000 | 1.427 |
| 2.21891 | 1.19378 | -0.00520 | 0.96995 | 0.013006 | 0.009521 | 0.000000 | 1.366 |
| 2.25394 | 1.22880 | -0.00564 | 0.97328 | 0.012331 | 0.009413 | 0.000000 | 1.310 |
| 2.29511 | 1.26997 | -0.00603 | 0.97657 | 0.011722 | 0.009309 | 0.000000 | 1.259 |
| 2.34351 | 1.31838 | -0.00637 | 0.97972 | 0.011183 | 0.009213 | 0.000000 | 1.214 |
| 2.40041 | 1.37528 | -0.00661 | 0.98266 | 0.010714 | 0.009125 | 0.000000 | 1.174 |
| 2.46731 | 1.44217 | -0.00671 | 0.98536 | 0.010313 | 0.009046 | 0.000000 | 1.140 |
| 2.54595 | 1.52082 | -0.00663 | 0.98778 | 0.009977 | 0.008977 | 0.000000 | 1.111 |
| 2.63840 | 1.61327 | -0.00628 | 0.98993 | 0.009701 | 0.008917 | 0.000000 | 1.088 |
| 2.74709 | 1.72196 | -0.00560 | 0.99178 | 0.009477 | 0.008866 | 0.000000 | 1.069 |
| 2.87487 | 1.84973 | -0.00447 | 0.99336 | 0.009299 | 0.008823 | 0.000000 | 1.054 |
| 3.02508 | 1.99993 | -0.00279 | 0.99506 | 0.009148 | 0.008777 | 0.000000 | 1.042 |

| # s | x | y | Ue/Vinf | Dstar | Theta | Cf | H |
|---|---|---|---|---|---|---|---|
| 0.00000 | 1.00000 | 0.00126 | 0.75725 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.00150 | 0.99852 | 0.00147 | 0.81249 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.00391 | 0.99612 | 0.00180 | 0.84009 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.00658 | 0.99349 | 0.00217 | 0.85812 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.00954 | 0.99055 | 0.00258 | 0.87185 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.01285 | 0.98727 | 0.00303 | 0.88375 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.01661 | 0.98355 | 0.00354 | 0.89430 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.02090 | 0.97930 | 0.00413 | 0.90447 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.02588 | 0.97436 | 0.00480 | 0.91317 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.03171 | 0.96858 | 0.00558 | 0.92277 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.03861 | 0.96174 | 0.00650 | 0.93206 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.04682 | 0.95361 | 0.00759 | 0.94056 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.05652 | 0.94400 | 0.00886 | 0.95062 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.06776 | 0.93285 | 0.01031 | 0.95966 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.08039 | 0.92032 | 0.01192 | 0.96841 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.09409 | 0.90673 | 0.01364 | 0.97720 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.10848 | 0.89245 | 0.01541 | 0.98486 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.12329 | 0.87775 | 0.01721 | 0.99239 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.13833 | 0.86282 | 0.01901 | 0.99901 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.15349 | 0.84776 | 0.02079 | 1.00547 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.16871 | 0.83264 | 0.02254 | 1.01121 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.18396 | 0.81749 | 0.02427 | 1.01681 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.19923 | 0.80232 | 0.02597 | 1.02203 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.21451 | 0.78713 | 0.02765 | 1.02705 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.22979 | 0.77193 | 0.02929 | 1.03177 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.24507 | 0.75674 | 0.03090 | 1.03652 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.26036 | 0.74154 | 0.03248 | 1.04103 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.27564 | 0.72634 | 0.03403 | 1.04528 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.29091 | 0.71113 | 0.03555 | 1.04964 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.30619 | 0.69593 | 0.03703 | 1.05381 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.32145 | 0.68074 | 0.03849 | 1.05789 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.33671 | 0.66554 | 0.03991 | 1.06203 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.35197 | 0.65035 | 0.04129 | 1.06600 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.36721 | 0.63517 | 0.04264 | 1.07006 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.38245 | 0.61999 | 0.04396 | 1.07388 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.39767 | 0.60482 | 0.04524 | 1.07801 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.41289 | 0.58965 | 0.04647 | 1.08169 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.42809 | 0.57450 | 0.04767 | 1.08574 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.44327 | 0.55936 | 0.04883 | 1.08964 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.45845 | 0.54423 | 0.04994 | 1.09346 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.47360 | 0.52911 | 0.05101 | 1.09753 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.48874 | 0.51400 | 0.05204 | 1.10120 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.50386 | 0.49892 | 0.05301 | 1.10526 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.51896 | 0.48384 | 0.05393 | 1.10916 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.53404 | 0.46879 | 0.05480 | 1.11297 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.54909 | 0.45376 | 0.05561 | 1.11685 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.56412 | 0.43875 | 0.05637 | 1.12087 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.57912 | 0.42377 | 0.05706 | 1.12469 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.59409 | 0.40881 | 0.05769 | 1.12854 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.60903 | 0.39388 | 0.05825 | 1.13252 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.62394 | 0.37898 | 0.05874 | 1.13638 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.63881 | 0.36411 | 0.05916 | 1.14006 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.65365 | 0.34928 | 0.05950 | 1.14388 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.66844 | 0.33449 | 0.05976 | 1.14789 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.68319 | 0.31974 | 0.05993 | 1.15124 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.69789 | 0.30505 | 0.06001 | 1.15531 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.71253 | 0.29040 | 0.06000 | 1.15866 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.72712 | 0.27581 | 0.05988 | 1.16221 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.74164 | 0.26129 | 0.05966 | 1.16562 | 0.000000 | 0.000000 | 0.000000 | 1.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.75609 | 0.24684 | 0.05933 | 1.16909 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.77047 | 0.23248 | 0.05888 | 1.17214 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.78475 | 0.21820 | 0.05831 | 1.17526 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.79894 | 0.20403 | 0.05760 | 1.17794 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.81301 | 0.18999 | 0.05676 | 1.18067 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.82694 | 0.17609 | 0.05577 | 1.18302 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.84071 | 0.16237 | 0.05463 | 1.18506 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.85428 | 0.14887 | 0.05334 | 1.18676 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.86760 | 0.13563 | 0.05188 | 1.18792 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.88060 | 0.12272 | 0.05025 | 1.18858 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.89320 | 0.11025 | 0.04847 | 1.18867 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.90529 | 0.09832 | 0.04654 | 1.18805 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.91673 | 0.08706 | 0.04449 | 1.18620 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.92741 | 0.07660 | 0.04235 | 1.18357 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.93721 | 0.06704 | 0.04016 | 1.17961 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.94608 | 0.05845 | 0.03796 | 1.17431 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.95402 | 0.05081 | 0.03579 | 1.16764 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.96108 | 0.04408 | 0.03367 | 1.15981 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.96734 | 0.03817 | 0.03162 | 1.14964 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.97289 | 0.03298 | 0.02963 | 1.13803 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.97784 | 0.02842 | 0.02772 | 1.12424 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.98228 | 0.02439 | 0.02586 | 1.10813 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.98627 | 0.02083 | 0.02405 | 1.08905 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.98990 | 0.01766 | 0.02228 | 1.06623 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.99321 | 0.01485 | 0.02054 | 1.03907 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.99625 | 0.01234 | 0.01882 | 1.00797 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 0.99906 | 0.01010 | 0.01712 | 0.97022 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.00167 | 0.00812 | 0.01542 | 0.92343 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.00412 | 0.00636 | 0.01372 | 0.87050 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.00641 | 0.00483 | 0.01201 | 0.80342 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.00857 | 0.00352 | 0.01030 | 0.72567 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01062 | 0.00241 | 0.00857 | 0.63320 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01257 | 0.00152 | 0.00683 | 0.52606 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01444 | 0.00084 | 0.00509 | 0.40515 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01623 | 0.00036 | 0.00337 | 0.27427 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01796 | 0.00009 | 0.00166 | 0.13649 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.01962 | 0.00000 | 0.00000 | -0.00017 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.02129 | 0.00009 | -0.00166 | -0.13605 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.02301 | 0.00036 | -0.00337 | -0.27420 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.02481 | 0.00084 | -0.00509 | -0.40503 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.02667 | 0.00152 | -0.00683 | -0.52742 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.02863 | 0.00241 | -0.00857 | -0.63279 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.03068 | 0.00352 | -0.01030 | -0.72525 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.03284 | 0.00483 | -0.01201 | -0.80449 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.03513 | 0.00636 | -0.01372 | -0.86903 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.03757 | 0.00812 | -0.01542 | -0.92435 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.04019 | 0.01010 | -0.01712 | -0.96992 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.04300 | 0.01234 | -0.01882 | -1.00800 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.04604 | 0.01485 | -0.02054 | -1.03954 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.04935 | 0.01766 | -0.02228 | -1.06616 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.05297 | 0.02083 | -0.02405 | -1.08896 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.05697 | 0.02439 | -0.02586 | -1.10770 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.06141 | 0.02842 | -0.02772 | -1.12454 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.06636 | 0.03298 | -0.02964 | -1.13797 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.07191 | 0.03817 | -0.03162 | -1.14980 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.07817 | 0.04408 | -0.03367 | -1.15958 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.08522 | 0.05081 | -0.03579 | -1.16772 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.09316 | 0.05845 | -0.03796 | -1.17444 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.10204 | 0.06704 | -0.04016 | -1.17962 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.11184 | 0.07660 | -0.04235 | -1.18352 | 0.000000 | 0.000000 | 0.000000 | 1.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.12251 | 0.08706 | -0.04449 | -1.18635 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.13396 | 0.09832 | -0.04654 | -1.18785 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.14605 | 0.11025 | -0.04847 | -1.18872 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.15865 | 0.12272 | -0.05025 | -1.18862 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.17165 | 0.13563 | -0.05188 | -1.18789 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.18497 | 0.14887 | -0.05334 | -1.18677 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.19854 | 0.16237 | -0.05463 | -1.18515 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.21231 | 0.17609 | -0.05577 | -1.18293 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.22624 | 0.18999 | -0.05676 | -1.18071 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.24031 | 0.20403 | -0.05760 | -1.17795 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.25449 | 0.21820 | -0.05831 | -1.17528 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.26878 | 0.23248 | -0.05888 | -1.17218 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.28315 | 0.24684 | -0.05933 | -1.16895 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.29761 | 0.26129 | -0.05966 | -1.16562 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.31213 | 0.27581 | -0.05988 | -1.16233 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.32672 | 0.29040 | -0.06000 | -1.15871 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.34136 | 0.30505 | -0.06001 | -1.15510 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.35606 | 0.31974 | -0.05993 | -1.15151 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.37081 | 0.33449 | -0.05976 | -1.14770 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.38560 | 0.34928 | -0.05950 | -1.14393 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.40043 | 0.36411 | -0.05916 | -1.14018 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.41531 | 0.37898 | -0.05874 | -1.13630 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.43021 | 0.39388 | -0.05825 | -1.13250 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.44516 | 0.40881 | -0.05769 | -1.12859 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.46013 | 0.42377 | -0.05706 | -1.12471 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.47513 | 0.43875 | -0.05637 | -1.12081 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.49016 | 0.45376 | -0.05561 | -1.11699 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.50521 | 0.46879 | -0.05480 | -1.11288 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.52029 | 0.48384 | -0.05393 | -1.10916 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.53539 | 0.49892 | -0.05301 | -1.10518 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.55051 | 0.51400 | -0.05204 | -1.10138 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.56565 | 0.52911 | -0.05101 | -1.09740 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.58080 | 0.54423 | -0.04994 | -1.09348 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.59597 | 0.55936 | -0.04883 | -1.08969 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.61116 | 0.57450 | -0.04767 | -1.08567 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.62636 | 0.58965 | -0.04647 | -1.08188 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.64158 | 0.60482 | -0.04524 | -1.07777 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.65680 | 0.61999 | -0.04396 | -1.07409 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.67204 | 0.63517 | -0.04264 | -1.06993 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.68728 | 0.65035 | -0.04129 | -1.06605 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.70254 | 0.66554 | -0.03991 | -1.06202 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.71780 | 0.68074 | -0.03849 | -1.05792 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.73306 | 0.69593 | -0.03703 | -1.05378 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.74834 | 0.71113 | -0.03555 | -1.04968 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.76361 | 0.72634 | -0.03403 | -1.04529 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.77889 | 0.74154 | -0.03248 | -1.04100 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.79417 | 0.75674 | -0.03090 | -1.03648 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.80946 | 0.77193 | -0.02929 | -1.03183 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.82474 | 0.78713 | -0.02765 | -1.02709 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.84002 | 0.80231 | -0.02597 | -1.02204 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.85529 | 0.81749 | -0.02427 | -1.01676 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.87054 | 0.83264 | -0.02254 | -1.01127 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.88576 | 0.84776 | -0.02079 | -1.00538 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.90092 | 0.86282 | -0.01901 | -0.99911 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.91596 | 0.87775 | -0.01721 | -0.99229 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.93077 | 0.89245 | -0.01541 | -0.98505 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.94516 | 0.90673 | -0.01364 | -0.97696 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.95886 | 0.92032 | -0.01192 | -0.96868 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.97149 | 0.93285 | -0.01031 | -0.95958 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 1.98273 | 0.94400 | -0.00886 | -0.95039 | 0.000000 | 0.000000 | 0.000000 | 1.000 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1.99243 | 0.95361 | -0.00759 | -0.94112 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.00063 | 0.96174 | -0.00650 | -0.93169 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.00754 | 0.96858 | -0.00558 | -0.92270 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.01337 | 0.97436 | -0.00480 | -0.91363 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.01835 | 0.97930 | -0.00413 | -0.90394 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.02264 | 0.98355 | -0.00354 | -0.89420 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.02640 | 0.98727 | -0.00303 | -0.88432 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.02971 | 0.99055 | -0.00258 | -0.87165 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.03267 | 0.99349 | -0.00217 | -0.85841 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.03533 | 0.99612 | -0.00180 | -0.84014 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.03775 | 0.99852 | -0.00147 | -0.81209 | 0.000000 | 0.000000 | 0.000000 | 1.000 |
| 2.03925 | 1.00000 | -0.00126 | -0.75725 | 0.000000 | 0.000000 | 0.000000 | 1.000 |