Senior Research Project
Degree in Aerospace Engineering

Numerical Resolution of Supersonic Flow using the MacCormack Method

Author: Antonio González Carvajal

Tutor: Miguel Pérez-Saborid Sánchez-Pastor

Aerospace Engineering and Fluid Mechanics Department
Superior Technique School of Engineering
University of Seville
Seville, 2018

Trabajo de Fin de Grado
Grado en Ingeniería Aeroespacial

Resolución numérica de flujo supersónico usando el método de MacCormack

Autor:

Antonio González Carvajal

Tutor:

Miguel Pérez-Saborid Sánchez-Pastor

Profesor Titular

Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo de Fin de Grado: Numerical Resolution of Supersonic Flow using the MacCormack's Technique

Autor: Antonio González Carvajal

Tutor: Miguel Pérez-Saborid Sánchez-Pastor

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi familia,

a mis amigos

y especialmente a mis abuelas y a Lorena.

Agradecimientos

Han sido varios meses los que le he dedicado a este trabajo y tengo que decir que a pesar de todo mi esfuerzo, no habría podido conseguirlo sin muchas personas. Porque a veces he estado al borde de un ataque de nervios, al borde de desistir, tirar la toalla, no verme capaz, y ellos me han sado fuerzas para seguir intentándolo. Este trabajo no habría sido terminado sin su apoyo.

En primer lugar mi familia, que siempre ha estado ahí dándome ánimos no solo en el tiempo que he estado trabajando en el proyecto, sino durante toda la carrera. Han sido mis padres los que han conseguido calmarme cuando perdía los nervios, sentía que no avanzaba y casi quería llorar. Ha sido mi hermana la que me distraía y me hacía reír aun cuando tenía miles de preocupaciones en la cabeza. Y han sido mis abuelas las que siempre se preocupaban por mí y me han dado tanto, desde conversaciones tranquilizadoras con la perspectiva de la edad hasta platos de cuchareo para comer en familia.

No solo a ellos tengo que agradecerles por supuesto. Tampoco habría conseguido terminar si no fuese por mis amigos. Mis amigos en Sevilla me recordaban en Julio que aún tenía tiempo y que vería las cosas mejor después de una cerveza en la Alameda. Amigos en Chipiona que consiguen hacer de los veranos una auténtica pasada. Gente tan maravillosa que a veces sientes que no las mereces de lo bien que lo pasas con ellos.

A todos ellos, gracias. Os lo debo todo y más.

Index

1 Introduction

1.1 Objectives and motivation of the project

In the aerospace degree a lot of focus is put into the fluid dynamics field, and extensive knowledge about fluid behaviour is acquired during the years that the student spends at the university. This knowledge is so huge that it is necessary a wide range of subjects to develop it. However, the necessity of evaluating the students, combined with the desire of showcasing the theoretical knowledge learned, turns into the development of analytical solutions that can be solved in class and subsequently be asked in the exam. To achieve these solutions, a number of simplifications must be made to make the equations manageable and usually the geometry of the problems is limited in order to simplify the problem even further. This turns the fluid dynamic subjects into a complex exercise of manipulating equations, apply simplifications and try to reach the analytical solution, losing in the process the relation to the physics of the problem. Several times the ability to solve the chosen problem does not increase the comprehension of the key concepts and does not develop an insight of the physical behaviours that occur in the fluid.

With the fast development of computers this issue can be somewhat mitigated. Nowadays, there are several programs that allow for an easy implementation of the equations that model the behaviour of the fluid and the numerical simulation of the flow, that then can show the solutions to the students. These programs increase the connection of the students to the fundamental concepts of fluid dynamics due to the great power that visualising the results has and makes them more involved as they are the ones that create the programs. Needless to say that also the use of computational fluid dynamics extends greatly the problems that can be solved, most of which do not count with an analytical solution. Following the stated by Pablo José Ruiz Contreras in the introduction of *“El Método de Colocación para el problema de convección de Rayleigh-Bénard”* [Ref 4], the fact that the program is created by the student avoids the “black-box” interaction with commercial fluid dynamic programs, whose use would not add nearly as much of learning value because the student is disconnected from the physical equations of the flow.

With all that said, the main motivation of this project is that in all the problems solved in all the subjects along the aerospace degree, always simplifications have been used, even when using the computer to solve the program. Thus, since the complete Navier-Stokes equations where presented to us, they have not been used fully in the resolution of a problem, there has always been some modifications to make them more manageable. This has created in me a desire of at least seen them once implemented completely and to really see, as much as possible, that the fluids really behave corresponding to them. It is like if I needed to see a simulated flow govern by those equations to really believe that they describe the behaviour of a fluid.

This is how it was suggested by my tutor that the project could be dedicated to the resolution of supersonic flows using computational fluid dynamics. The resolution of subsonic flows using the complete equations had already been done by Manuel Carreño in [Ref 2], and it was decided that this project could expand in that direction using the MacCormack's method to solve supersonic flows and more importantly, to apply them to non-uniform grids, which has been by far the most difficult thing.

1.2 Supersonic Flow, Boundary Layers and Shock Waves

Inside any physical medium there is a maximum velocity at which the perturbation of any property at one point of the medium can be transmitted to the rest of it. It is the speed of the sound, depends on the internal properties of the medium, and marks the limit at which the information can travel through it. For instance, the speed of sound inside a fluid is:

$$a = \gamma \frac{p}{\rho}$$

Equation 1.1. Speed of sound in a fluid.

When a solid object travels faster through a fluid than the speed of sound, the fluid that is ahead of it cannot receive the information that the object exists in time, and thus cannot gradually adapt itself to the geometry of the incoming object. The result is that it has to make the adaptation instantly, suddenly increasing its pressure in a physical mechanism called "Shock Wave".

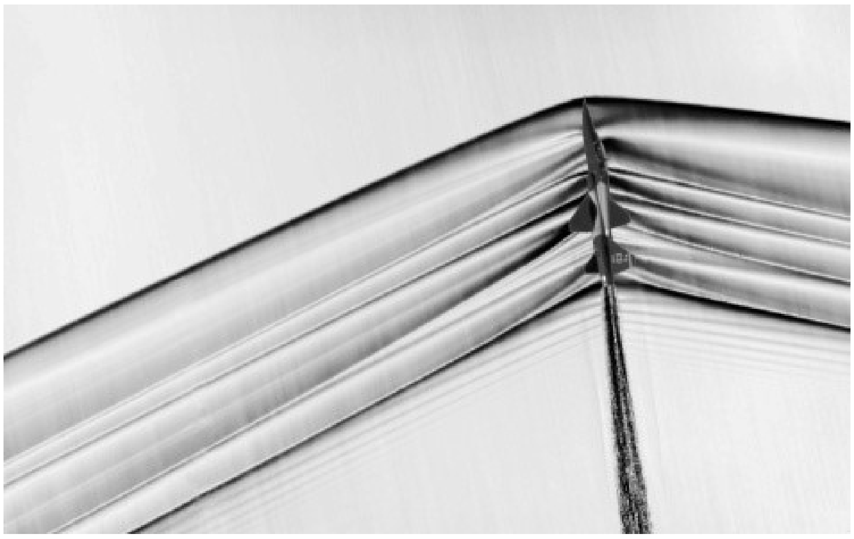


Figure 1.1. Visible shock waves in a supersonic plane.

The angle of the shock wave depends on the Mach number, which is the velocity of the object with respect to the speed of the sound.

$$M = \frac{V}{a}$$

Equation 1.2. Mach number definition.

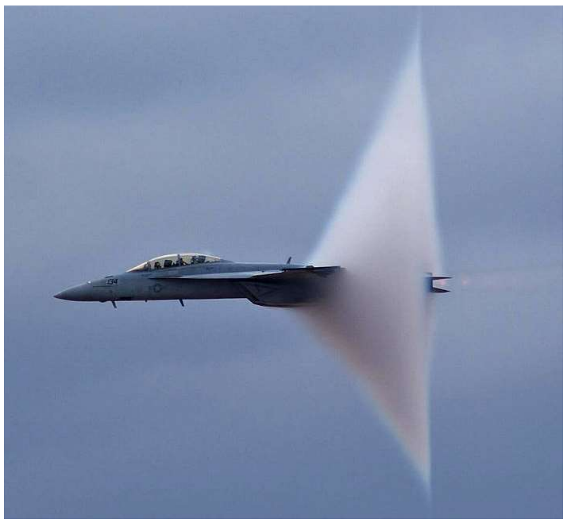


Figure 1.2. Visualized Mach cone due to condensation.

In real life, where the flow is in a three-dimensional space, the shock wave creates a cone known as the Mach cone, and it usually creates condensation phenomena that allow its visualization.

The way a shock wave affects the properties of the flow depends greatly on the geometry of the object, Mach number, the internal properties of the unaltered flow, etc. and it is the objective of this project to analyse these dependencies.

It could also happen that the shock wave somehow interacted with the boundary layer created by the object.

The boundary layer is a region of space in which the velocity of the flow transitions between zero, due to the no-slip condition that happens at the surface of the object, and the velocity of the exterior flow.

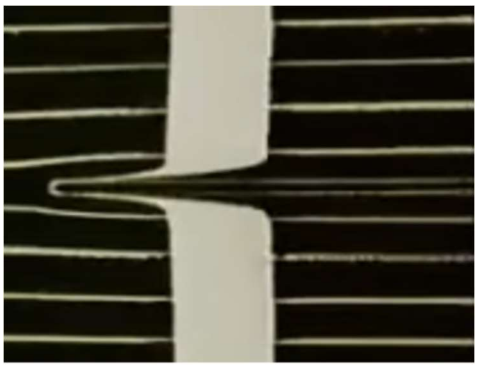


Figure 1.3. Boundary layer in a flat plate.

The Fluid Mechanics professor Ludwig Prandtl presented the concept in 1904 and proved that the viscosity effects where confined to this small region very close to the surface of the object. In the rest of the flow field the viscosity effects can be considered negligible.

It would also be interesting to investigate if the supersonic flow can also suffer the phenomenon of turbulence and boundary layer detachment.

The turbulence is a well-studied behaviour in which the viscosity is no longer able to mitigate the small perturbations that affect the physical flow and the flow transitions from a laminar state, where the flow is orderly divided in layers, to a turbulent state where there are important fluctuations in the magnitudes and the layers merge.

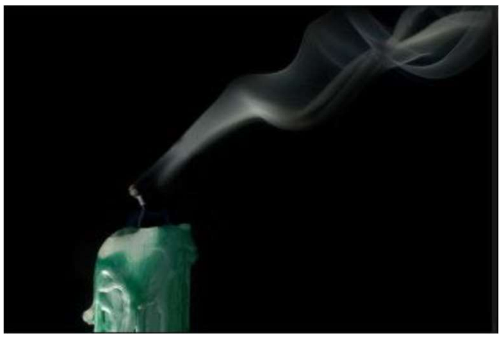


Figure 1.4. In the smoke of a candle it can be appreciated the jump from laminar to turbulent behaviour

On the other hand, the detachment of the boundary layer happens when the pressure gradient in the boundary layer is against the direction of the flow, and is big enough to be stronger than the momentum transmitted to the inner layers from the outer flow by viscosity effects. When that happens, the inner layers stop moving and a recirculating area is created around which the rest of the flow slips.

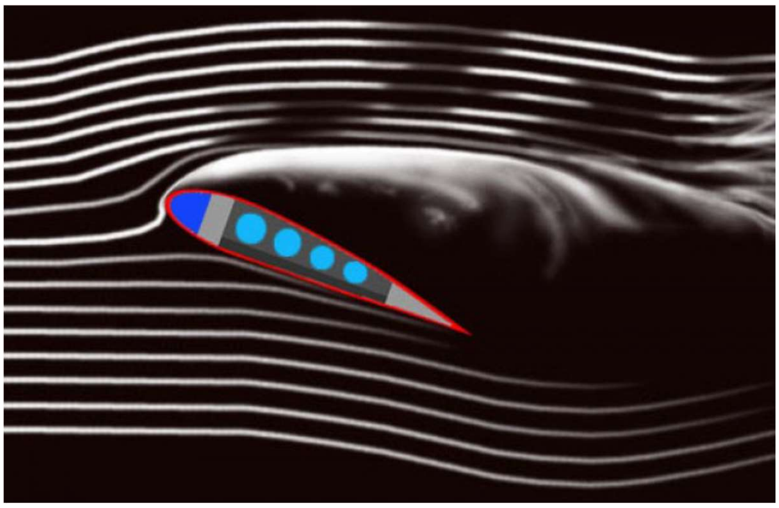


Figure 1.5. Boundary layer detachment around a wing profile.

1.3 Structure of the project

This project will apply the numerical technique developed by McCormack into a program to solve supersonic problems with different geometries and conditions. It will begin after this introduction by explaining in the **second chapter** the theoretical knowledge that is needed to understand the project. First, the equations of the flow that have been implemented, followed by an explanation of the method used for uniform grids. After that, it will be explained the changes in the method necessary to develop the program for non-uniform grids.

The **third chapter** will be focused on explaining the program that has been written section by section, first for the uniform grids, and after that for the non-uniform grids. At the end of the explanation of each section the program used in each case will be explicitly showed.

In the **fourth chapter**, the results of the different applications will be showed, beginning with the flat plate. After the plate, the forward and rearward steps will be explained, followed by the rearward slope, where it will be analysed the Prandtl-Meyer Expansion Waves. The last application will be the study of a leading edge and the shock wave created around it.

Finally, the project will end with a brief discussion about conclusions and further developments in the **fifth chapter**, the references in the **chapter six**, and the summary in spanish language demanded by the University of Seville normative in **Appendix 1**.

2 Theoretical basis

From the multiple methods proposed by CFD, in this project it has been chosen a time marching calculation using the MacCormack's technique, for several reasons.

It was first introduced by Stanford University professor Robert W. MacCormack in 1969. Among its advantages there is that it is one of the easiest and more understandable methods that allows the solving of the complete Navier-Stokes equations, which is the goal of this project, even though it is still as complex as expected for a full-on implementation of this equations.

The current chapter is going to explain the method step by step as it has been used in this project, and after that, the following chapter will explain the coding implementation by going through the program used to obtain the results.

2.1 CDF equations

If the resolution of the complete Navier-Stokes equations by numeric means is required, first an adaptation is recommended to make the equations more manageable. To begin with, the equations for a bidimensional flow are remembered here.

- Conservation of mass:

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \vec{V} = 0$$

Equation 2.1

- Conservation of momentum in the x and y directions:

$$\frac{\partial(\rho u)}{\partial t} + \nabla \cdot (\rho u \cdot \vec{V}) = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \rho f_x$$

Equation 2.2

$$\frac{\partial(\rho v)}{\partial t} + \nabla \cdot (\rho v \cdot \vec{V}) = -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \rho f_y$$

Equation 2.3

- Conservation of energy:

$$\begin{aligned}\frac{\partial}{\partial t}\left[\rho\left(e + \frac{V^2}{2}\right)\right] + \nabla \cdot \left[\rho\left(e + \frac{V^2}{2}\right) \cdot \vec{V}\right] \\ = \rho\dot{\vec{q}} + \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) - \frac{\partial(up)}{\partial x} - \frac{\partial(vp)}{\partial y} + \frac{\partial(u\tau_{xx})}{\partial x} + \frac{\partial(u\tau_{yx})}{\partial y} \\ + \frac{\partial(v\tau_{xy})}{\partial x} + \frac{\partial(v\tau_{yy})}{\partial y} + \rho\vec{f} \cdot \vec{V}\end{aligned}$$

Equation 2.4

Just with a first glance of the equations, it can be recognised a simpler, more compacted way of expressing them by grouping together the derivative terms with respect to the same variable. The result would be an expression with the following form:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=S$$

Equation 2.5

Where *U, F, G* and *S* are vectors with one component for each of the equations that govern the behaviour of the flow. For the bidimensional flow that is being discussed, their expressions are shown here:

$$U=\begin{Bmatrix}\rho\\ \rho u\\ \rho v\\ \rho\left(e+\frac{V^2}{2}\right)\end{Bmatrix}$$

Equation 2.6

$$F=\begin{Bmatrix}\rho u\\ \rho u^2+p-\tau_{xx}\\ \rho vu-\tau_{xy}\\ \rho\left(e+\frac{V^2}{2}\right)u+pu-k\frac{\partial T}{\partial x}-u\tau_{xx}-v\tau_{xy}\end{Bmatrix}$$

Equation 2.7

$$G=\begin{Bmatrix}\rho v\\ \rho uv-\tau_{yx}\\ \rho v^2+p-\tau_{yy}\\ \rho\left(e+\frac{V^2}{2}\right)v+pv-k\frac{\partial T}{\partial y}-u\tau_{yx}-v\tau_{yy}\end{Bmatrix}$$

Equation 2.8

$$S=\begin{Bmatrix}0\\ \rho f_x\\ \rho f_y\\ \rho\left(uf_x+vf_y\right)+\rho\dot{\vec{q}}\end{Bmatrix}$$

Equation 2.9

This way it can be obtained a particularly suitable expression to manage with CFD methods. This expression is often called the "Conservation Form", and is extensively used throughout several CFD methods, not only in the case of the MacCormack's technique.

In the equations Equation 2.7 and Equation 2.8, an expression for τ_{xx} , τ_{yx} and τ_{yy} is needed to successfully evaluate the variable vectors *F* and *G*. Assuming a newtonian fluid, these are given by the expressions obtained by Stokes for this kind of fluids:

$$\tau_{xx} = \lambda(\nabla \cdot \vec{V}) + 2\mu\frac{\partial u}{\partial x}$$

Equation 2.10

$$\tau_{yy} = \lambda(\nabla \cdot \vec{V}) + 2\mu\frac{\partial v}{\partial y}$$

Equation 2.11

$$\tau_{xy} = \tau_{yx} = \mu\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right)$$

Equation 2.12

In addition, it is going to be used the hypothesis made also by Stokes about the value of the volumetric viscosity coefficient, $\lambda = -\frac{2}{3}\mu$. So, the expressions Equation 2.10 and Equation 2.11 change into:

$$\tau_{xx} = \frac{2}{3}\mu\left(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right)$$

Equation 2.13

$$\tau_{yy} = \frac{2}{3}\mu\left(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\right)$$

Equation 2.14

With these expressions, the system consists of eight variables: $\rho,\ u,\ v,\ p,\ T,\ e,\ \mu$ and k; but only has four equations. Thus, we must provide four more equations to be able to solve the system, counting on the fact that we consider V not an independent variable, as it can be easily obtained from the velocity components u and v. The first of them is the assumption of a perfect gas, from which the perfect gas equation is derived:

$$p = \rho RT$$

Equation 2.15

Where R is the perfect gas constant of value 287 J/ (kg K). The second is an assumption built on top of the previous one, the calorically perfect gas. The relation that describes this assumption is the following:

$$e = c_v T$$

Equation 2.16

Where c_v is the specific heat at constant volume and is, by the calorically perfect definition, considered constant. Given the value of the ratio of specific heats, γ, equal to 1.4, it is obtained with the following relation:

$$c_v = \frac{R}{\gamma - 1}$$

The next one is a model for the viscosity provided by the Sutherland's law:

$$\mu = \mu_0 \left(\frac{T}{T_0}\right)^{3/2} \frac{T_0 + 110}{T + 110}$$

Equation 2.17

In which μ_0 and T_0 are reference values taken at standard sea level. The last equation is given by assuming the Prandtl number constant and approximately equal to 0.71 in the case of air. From the Prandtl number definition it is obtained:

$$Pr = \frac{\mu c_p}{k} = 0.71$$

Equation 2.18

Where c_p is the specific heat at constant pressure and is easily obtained as:

$$c_p = \gamma c_v$$

2.2 MacCormack's Method for Uniform Grids

The MacCormack's technique relies on transforming the governing equations into the *conservative form* and then performs a two-phase discretization and combines them to assure second order accuracy, as it is going to be demonstrated in this subchapter. From Equation 2.5, by isolating the time derivative in one side, it is obtained:

$$\frac{\partial U}{\partial t} = S - \frac{\partial F}{\partial x} - \frac{\partial G}{\partial y}$$

Equation 2.19

Now, the flow domain is discretised into points following an equispaced rectangular grid, with each point being designated by its coordinates *i* and *j*. The number of points in the X and Y axis are Nx and Ny respectively, as shown in Figure 2.1.

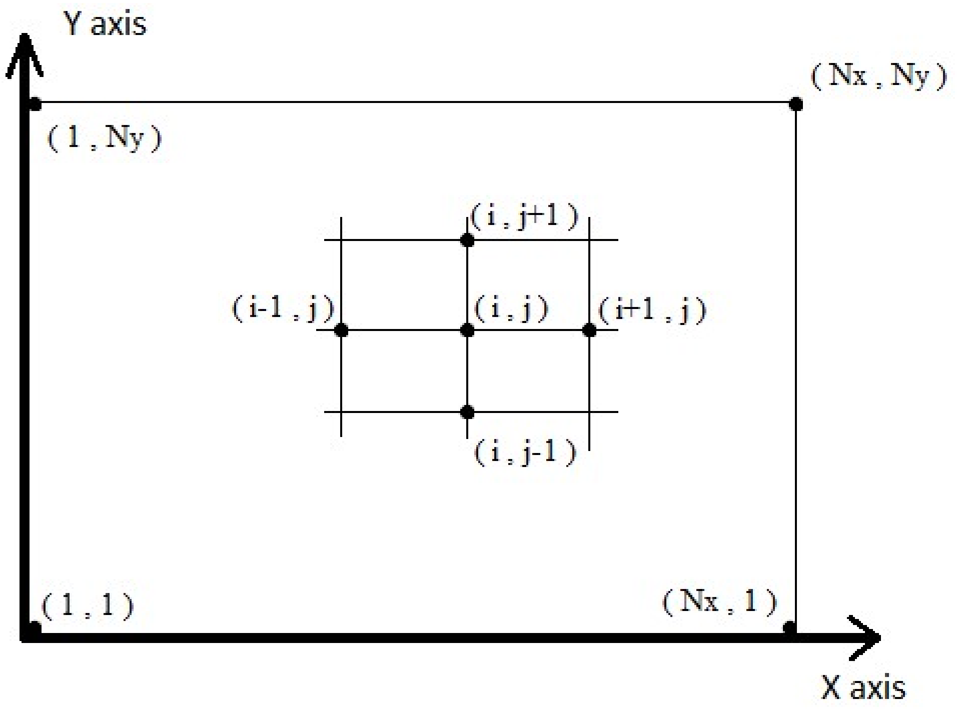


Figure 2.1. Computational domain and Uniform Grid

For each time step, the properties of the flow are calculated by finite difference approximation **only for the inner points** of the grid. The properties of the flow in the boundary must be treated differently and obtained by other means. That is, given by the boundary conditions or extrapolated from the flow properties inside the domain.

The boundary conditions that have previously been mentioned have to be explicitly enforced in the method. However, the Forward-Rearward steps that are going to be explained later offer as their result the vector *U*. Moreover, the physical variables, that are going to be referred as "primitive" from this point on, also have to be obtained to assess the final results.

The boundary conditions could be enforced by combining the enforced primitive variables and by combining *U* values when an extrapolation is needed. Despite that, in this project it has been chosen the more straightforward and comprehensible approach of decoding the primitive variables in each step, then enforcing the boundary conditions on them, and after that, recalculating the vectors *U, F, G* and *S*.

So, after each step, the primitive variables are decoded from *U* as follows:

$$\rho = U_1$$

Equation 2.20

$$u = \frac{\rho u}{\rho} = \frac{U_2}{U_1}$$

Equation 2.21

$$v = \frac{\rho v}{\rho} = \frac{U_3}{U_1}$$

Equation 2.22

$$e = \frac{U_4}{U_1} - \frac{u^2 + v^2}{2}$$

Equation 2.23

Using the Equation 2.15 the pressure is obtained, and from the Equation 2.16:

$$T = \frac{e}{c_v}$$

Equation 2.24

After that, the viscosity is calculated from Sutherland's law, Equation 2.17, and using Equation 2.18:

$$k = \frac{\mu c_p}{Pr}$$

Equation 2.25

For the first step, starting with Equation 2.19, the derivatives are approximated using finite differences, with the values Δx, Δy and Δt substituting ∂x, ∂y and ∂t respectively. Their appropriate values are going to be discussed in the next chapter. The magnitudes are projected in the *forward* direction for both space directions, and the result is the *predicted* value of the vector *U* at the time $t + \Delta t$, which is denoted by $\overline{U}^{t+\Delta t}$. The result expression, already solved for $\overline{U}^{t+\Delta t}$, is:

$$\overline{U}_{i,j}^{t+\Delta t} = U_{i,j}^{t} - \frac{\Delta t}{\Delta x}\left(F_{i+1,j}^{t} - F_{i,j}^{t}\right) - \frac{\Delta t}{\Delta y}\left(G_{i,j+1}^{t} - G_{i,j}^{t}\right) + \Delta t \cdot S$$

Equation 2.26

Now, there is an important remark that has to be made. The vectors *F* and *G* are here finitely differenced, but they are themselves formed by derivative terms. These terms also have to be calculated using a finite difference approximation, but, to ensure the second order accuracy of the method, these finite differences must be in the opposite direction that of the vector they form part of. If the differentiation is not in the same direction, then central difference is used.

For example, in the forward step, the viscosity τ_{xx}, that forms part of the vector *F*, must be calculated using *rearward* difference in the X direction, and using *central* difference in the Y direction. Thus, the final expression for τ_{xx} is the following:

$$\tau_{xx}{}^{t}{}_{i,j} = \frac{2}{3}\mu_{i,j}^{t}\left(2\frac{u_{i,j}^{t}-u_{i-1,j}^{t}}{\Delta x} - \frac{v_{i,j+1}^{t}-v_{i,j-1}^{t}}{2\,\Delta y}\right)$$

Equation 2.27

The same reasoning with τ_{yy} conducts to:

$$\tau_{yy}{}^{t}{}_{i,j} = \frac{2}{3}\mu_{i,j}^{t}\left(2\frac{v_{i,j}^{t}-v_{i,j-1}^{t}}{\Delta y} - \frac{u_{i+1,j}^{t}-u_{i-1,j}^{t}}{2\,\Delta x}\right)$$

Equation 2.28

The case of τ_{xy} is different. As it takes part in both *F* and *G* calculations, two different calculations of τ_{xy} have to be made, one for each vector, with distinct differentiation in each case. Thus, the expressions reached for τ_{xy} used in *F* and *G* are:

$$\left.\tau_{xy}{}^{t}{}_{i,j}\right|_{F} = \mu_{i,j}^{t}\left(\frac{v_{i,j}^{t}-v_{i-1,j}^{t}}{\Delta x} + \frac{u_{i,j+1}^{t}-u_{i,j-1}^{t}}{2\,\Delta y}\right)$$

Equation 2.29

$$\left.\tau_{xy}{}^{t}{}_{i,j}\right|_{G} = \mu_{i,j}^{t}\left(\frac{v_{i+1,j}^{t}-v_{i-1,j}^{t}}{2\,\Delta x} + \frac{u_{i,j}^{t}-u_{i,j-1}^{t}}{\Delta y}\right)$$

Equation 2.30

Also, the terms referring heat conduction with the Fourier's law are approximated with finite differences:

$$\frac{\partial T^{t}}{\partial x}_{\,i,j} \approx \frac{T^{t}{}_{i,j} - T^{t}{}_{i-1,j}}{\Delta x}$$

Equation 2.31

$$\frac{\partial T^{t}}{\partial y}_{\,i,j} \approx \frac{T^{t}{}_{i,j} - T^{t}{}_{i,j-1}}{\Delta y}$$

Equation 2.32

It must be taken into account that these expressions cannot be used in the limits of the computational domain, as some of them would require points that are outside of the boundaries. So, for example, in the lower boundary the central and rearward differences done in the Y direction are substituted by forward differentiation. In the upper border, the forward and central are substituted with rearward ones, and the same logic applies to the left and the right boundaries for the X direction, respectively. As stated by John D. Anderson in his book

"*Computational Fluid Dynamics*" [Ref 1], doing this only compromises slightly the second order accuracy of the method.

After calculating the vectors *F* and *G*, the Equation 2.26 is applied, the primitive variables are extracted as shown in equations Equation 2.20 to Equation 2.25 and the boundary conditions are enforced onto these variables.

The second step proceeds in a similar manner. However, there is a clear difference. Now the vectors *F* and *G* used are calculated using the *predicted* values of the variables. The equivalent to Equation 2.26 now is the following expression:

$$\bar{\bar{U}}_{i,j}^{t+\Delta t} = U_{i,j}^{t} - \frac{\Delta t}{\Delta x}\left(\bar{F}_{i,j}^{t+\Delta t} - \bar{F}_{i,j-1}^{t+\Delta t}\right) - \frac{\Delta t}{\Delta y}\left(\bar{G}_{i,j}^{t+\Delta t} - \bar{G}_{i,j-1}^{t+\Delta t}\right) + \Delta t \cdot S$$

Equation 2.33

Where $\bar{\bar{U}}^{t+\Delta t}$ is called the *corrected* value of *U*. As it can be seen in the Equation 2.33, the vectors $\bar{F}^{t+\Delta t}$ and $\bar{G}^{t+\Delta}$ are obtained using the *predicted* values calculated from the previous step. Also, the finite differences are now projected *rearward.*

Similarly to what happened in the *forward* step, the derivative terms that form part of $\bar{F}$ and $\bar{G}$ must be approximated using the opposite direction in their finite differences. So, the expressions for τ_{xx} , τ_{xy} , τ_{yy} , $\frac{\partial T}{\partial x}$ and $\frac{\partial T}{\partial y}$ used in the corrector step are similar to the expressions Equation 2.27 to Equation 2.32, but with different differentiation. To avoid excessive repetition in this theoretical chapter, these expressions are not explicitly stated. However, if the reader so chooses, their implementation can be found in the section "`Encode variables for corrector step`" of the script that appears at the end of subchapter 3.1.

Exactly as it happened in the *predictor* step, two calculations of ${\bar{\tau}_{xy}}^{t+\Delta t}$ must be made, one for the vector $\bar{F}$ and other for $\bar{G}$. Also, the same considerations regarding the substitution of some types of differences with others in the boundaries of the computational domain apply.

Once the value of $\bar{\bar{U}}^{t+\Delta t}$ is calculated, the primitive variables are decoded to apply the boundary conditions on them, and then the vector $\bar{\bar{U}}^{t+\Delta t}$ is recalculated with the boundary conditions already applied.

Having the values of $\bar{U}^{t+\Delta t}$ and $\bar{\bar{U}}^{t+\Delta t}$ from the *predictor* and *corrector* steps respectively, the value of $U^{t+\Delta t}$ is obtained as their arithmetic mean.

$$U_{i,j}^{t+\Delta t} = \frac{1}{2}\left(\bar{U}_{i,j}^{t+\Delta t} + \bar{\bar{U}}_{i,j}^{t+\Delta t}\right)$$

Equation 2.34

Finally, the physical variables are obtained as the final result of the method at time $t + \Delta t$. The cycle is then restarted, advancing in time and capturing the behaviour of the flow.

To demonstrate the second order accuracy of the method, first it is going to be introduced the Taylor expansion series for a generic function $\varphi(x)$ around the point x_0 for both a positive and a negative increase in x value.

$$\varphi(x_0 + \Delta x) = \varphi(x_0) + \Delta x \frac{d\varphi}{dx}(x_0) + \frac{\Delta x^2}{2}\frac{d^2\varphi}{d^2x}(x_0) + \frac{\Delta x^3}{6}\frac{d^3\varphi}{d^3x}(x_0) + \cdots$$

Equation 2.35

$$\varphi(x_0 - \Delta x) = \varphi(x_0) - \Delta x \frac{d\varphi}{dx}(x_0) + \frac{\Delta x^2}{2}\frac{d^2\varphi}{d^2x}(x_0) - \frac{\Delta x^3}{6}\frac{d^3\varphi}{d^3x}(x_0) + \cdots$$

Equation 2.36

The central finite difference is going to be treated first. Subtracting Equation 2.36 from Equation 2.35 and solving for $\frac{d\varphi}{dx}(x_0)$:

$$\frac{d\varphi}{dx}(x_0) = \frac{\varphi(x_0 + \Delta x) - \varphi(x_0 - \Delta x)}{2\,\Delta x} - \frac{\Delta x^2}{6}\frac{d^3\varphi}{d^3x}(x_0) - \cdots$$

Equation 2.37

$$\frac{d\varphi}{dx}(x_0) \approx \frac{\varphi(x_0 + \Delta x) - \varphi(x_0 - \Delta x)}{2\,\Delta x}$$

Equation 2.38

Using the approximation given by Equation 2.38 the neglected terms are two derivative orders inferior to the estimated value. That means that the central difference is by itself of second order accuracy.

Continuing the analysis of the finite differences used, from the Equation 2.35 it can be obtained:

$$\frac{d\varphi}{dx}(x_0) = \frac{\varphi(x_0 + \Delta x) - \varphi(x_0)}{\Delta x} - \frac{\Delta x}{2}\frac{d^2\varphi}{d^2x}(x_0) - \frac{\Delta x^2}{6}\frac{d^3\varphi}{d^3x}(x_0) - \cdots$$

Equation 2.39

$$\frac{d\varphi}{dx}(x_0) \approx \frac{\varphi(x_0 + \Delta x) - \varphi(x_0)}{\Delta x}$$

Equation 2.40

So, when using the approximation given by Equation 2.40, the neglected terms are now one derivative order inferior to the estimated value, thus proving the forward finite difference is of first order accuracy.

From Equation 2.36:

$$\frac{d\varphi}{dx}(x_0) = \frac{\varphi(x_0) - \varphi(x_0 - \Delta x)}{\Delta x} + \frac{\Delta x}{2}\frac{d^2\varphi}{d^2x}(x_0) - \frac{\Delta x^2}{6}\frac{d^3\varphi}{d^3x}(x_0) + \cdots$$

Equation 2.41

$$\frac{d\varphi}{dx}(x_0) \approx \frac{\varphi(x_0) - \varphi(x_0 - \Delta x)}{\Delta x}$$

Equation 2.42

So, as expected, the rearward finite difference is also of first order accuracy. Despite that, MacCormack's technique combines the forward and rearward differences into one single approximation for the calculation of the variables at the next moment in time. Taking Equation 2.39 and adding Equation 2.41, the second order derivatives cancel each other.

$$\frac{d\varphi}{dx}(x_0) = \frac{\varphi(x_0 + \Delta x) - \varphi(x_0 - \Delta x)}{2\,\Delta x} - \frac{\Delta x^2}{6}\frac{d^3\varphi}{d^3x}(x_0) - \cdots$$

Equation 2.43

The Equation 2.43 is identical to Equation 2.37, which means that the arithmetic mean of forward and rearward differences is equivalent to a central difference, and so the method has a second order accuracy, what this subchapter was trying to demonstrate.

As a final remark before assessing the non-uniform grids, it is worth mentioning that Manuel Carreño, in his publication "*Aplicaciones del método de MacCormack a diversos problemas fluidomecánicos*" [Ref 2] recommends the usage of interchangeable steps in each iteration of the method, as well as does John D. Anderson [Ref 1]. They state that the alternate use of the forward and rearward step can improve the performance of the method and prevent the formation of preferred directions for the simulated flow.

That means that for some iterations the normal forward-rearward differences in the steps would be used, and for others the order would be reversed to a rearward-forward sequence, effectively changing the scheme to a corrector followed by a predictor. Mr. Carreño even suggest sometimes using two predictor or two corrector steps.

These options were interesting for the supposed improvement in the behaviour of the flow and they were tried to be implemented. However, despite all the effort put into them, no valid program was achieved, and all attempts presented an unstable and divergent behaviour wherever that magnitude gradients were just a little bit high.

2.3 MacCormack's Method for Non-Uniform Grids

A rectangular uniform grid is rarely appropriate for the problems that CFD faces. In fact, limiting this project only to them would have set important constrains in the variety of problems and specially geometries that could have been analysed. For example, the geometries inside the flow would have been limited to horizontal and vertical walls.

To overcome these limitations, the implementation of non-uniform grids must be set. The strategy chosen has been to transition from the non-rectangular, non-uniform ***physical*** domain to a rectangular and uniform ***computational*** domain using a transformation, make the calculations in the computational domain, and then go back to the physical domain when it is needed. For instance, to enforce boundary conditions, which are only known in the physical world.

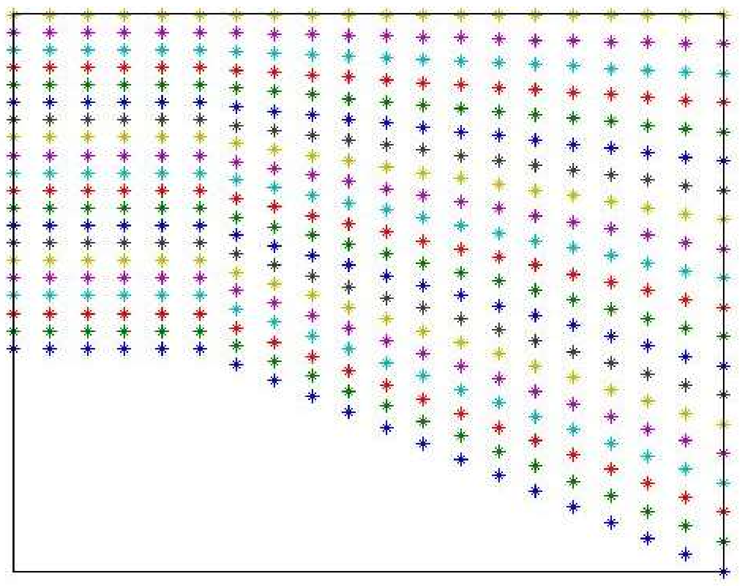


Figure 2.2. Example of non-uniform grid used. The points represent the physical equivalents of the computational uniform grid points.

To discuss the way the equations have to be adapted it is going to be assumed that the transformation is given by a mathematical relation that transforms the variables of direction and space (x, y, t) from the physical space to an alternative set of variables that define the transformed space (ξ, η, τ). This mathematical relation can be given two ways:

- **Direct transformation**: gives the transformed variables as functions of the physical ones.

$$\xi = \xi(x, y, t)$$

Equation 2.44

$$\eta = \eta(x, y, t)$$

Equation 2.45

$$\tau = \tau(t)$$

Equation 2.46

- **Inverse transformation**: gives the physical variables as functions of the transformed ones.

$$x = x(\xi, \eta, \tau)$$

Equation 2.47

$$y = y(\xi, \eta, \tau)$$

Equation 2.48

$$t = t(\tau)$$

Equation 2.49

Usually in these relations the time transformation is not really used, and so Equation 2.46 and Equation 2.49 turn into $t = \tau$. As this has been the case for this project and no time transformation has been needed, in the following chapters the identity between t and τ is going to be assumed and only the letter t will be used to denote time.

To properly use the transformations stated above, the derivatives with respect to x and y that appear in the original equations (review subchapter 2.1) must be replaced with derivatives with respect to ξ and η. When the relation is given by the direct transformation, the expressions are straightforward. Utilizing the chain rule of differential calculus:

$$\frac{\partial}{\partial x} = \left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) + \left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial x}\right)$$

Equation 2.50

$$\frac{\partial}{\partial y} = \left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial y}\right) + \left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial y}\right)$$

Equation 2.51

$$\frac{\partial}{\partial t} = \left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial t}\right) + \left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial t}\right)$$

Equation 2.52

Also, expressions for the second order derivatives exist, but as they have not been necessary for the implementation of the program, they are not going to be developed. In Equation 2.50, Equation 2.51 and Equation 2.52 the derivatives of the transformed variables with respect to the physical variables are numbers, which depend on the point they are evaluated at, and are called **direct metrics**. They can be easily calculated from the direct transformation.

$$Direct\ metrics\colon \left(\frac{\partial \xi}{\partial x}\right), \left(\frac{\partial \eta}{\partial x}\right), \left(\frac{\partial \xi}{\partial y}\right), \left(\frac{\partial \eta}{\partial y}\right), \left(\frac{\partial \xi}{\partial t}\right), \left(\frac{\partial \eta}{\partial t}\right).$$

The same expressions are not nearly as easily obtained when the transformation is given in the inverse form. Now the direct metrics cannot be calculated directly. Instead, the derivatives easily accessible are those of the physical variables with respect to the transformed ones. They are called **inverse metrics**.

$$Inverse\ metrics\colon \left(\frac{\partial x}{\partial \xi}\right), \left(\frac{\partial x}{\partial \eta}\right), \left(\frac{\partial y}{\partial \xi}\right), \left(\frac{\partial y}{\partial \eta}\right), \left(\frac{\partial x}{\partial \tau}\right), \left(\frac{\partial x}{\partial \tau}\right).$$

Here, another simplification of the transformation will be made. The spatial variables are not going to be dependent on time. This possible dependency has not been used in this project anyway, and allows to significantly simplify the explanation. Starting from the exact differential expressions:

$$d\xi = \frac{\partial \xi}{\partial x} dx + \frac{\partial \xi}{\partial y} dy$$

Equation 2.53

$$d\eta = \frac{\partial \eta}{\partial x} dx + \frac{\partial \eta}{\partial y} dy$$

Equation 2.54

$$dx = \frac{\partial x}{\partial \xi} d\xi + \frac{\partial x}{\partial \eta} d\eta$$

Equation 2.55

$$dy = \frac{\partial y}{\partial \xi} d\xi + \frac{\partial y}{\partial \eta} d\eta$$

Equation 2.56

Grouping the first two equations and the second two, using the matrix notation and solving for the physical differentials in the second group, the relation obtained is:

$$\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix}^{-1}$$

And using the determinant method to create the inverse of the matrix in the right side:

$$\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} \end{bmatrix} = \frac{\begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial y}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{bmatrix}}{\begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix}} = \frac{\begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial y}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{bmatrix}}{J}$$

Equation 2.57

The determinant in the denominator, with the terms in the down-up diagonal transported, is denominated as the ***Jacobian*** of the transformation. The relations of the direct metrics and the inverse metrics, using the Jacobian, are therefore now known.

If these expressions are used to change Equation 2.50 and Equation 2.51, the results are the expressions that allow replacing the derivatives with respect to the physical variables but this time utilizing the *inverse metrics*, that are much more easily obtained from the *inverse transformation* expressions.

$$\frac{\partial}{\partial x} = \frac{1}{J}\left[\left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial y}{\partial \eta}\right) - \left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial y}{\partial \xi}\right)\right]$$

Equation 2.58

$$\frac{\partial}{\partial y} = \frac{1}{J}\left[\left(\frac{\partial}{\partial \eta}\right)\left(\frac{\partial x}{\partial \xi}\right) - \left(\frac{\partial}{\partial \xi}\right)\left(\frac{\partial x}{\partial \eta}\right)\right]$$

Equation 2.59

It will not always be the case that the relation between the two set of variables is given by a mathematical function. Actually, most of the applications in this project do not follow this rule. Instead the grid is numerically generated, and as a result, the metrics do not have a mathematical expression. When that happens, the direct and inverse metrics will be calculated using central finite differentiation which, as will be discussed in the next chapter, can cause some troubles due to the intrinsic approximation errors.

The overall goal of this subchapter is to transform Equation 2.5 into an alternative equation with variants of the vectors *U, F, G* and *S*, and derivatives with respect to ξ and η as follows:

$$\frac{\partial U_1}{\partial t} + \frac{\partial F_1}{\partial \xi} + \frac{\partial G_1}{\partial \eta} = S_1$$

The first steps are to use Equation 2.50 and Equation 2.51 into Equation 2.5, and multiply by the Jacobian.

$$J\left(\frac{\partial U}{\partial t}\right) + J\left(\frac{\partial F}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) + J\left(\frac{\partial F}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial x}\right) + J\left(\frac{\partial G}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial y}\right) + J\left(\frac{\partial G}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial y}\right) = J \cdot S$$

Equation 2.60

From the multiplication chain derivation rule, and rearranging the terms, the following relations are held:

$$J\left(\frac{\partial F}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial x}\right) = \frac{\partial}{\partial \xi}\left[JF\left(\frac{\partial \xi}{\partial x}\right)\right] - F\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right)$$

Equation 2.61

$$J\left(\frac{\partial F}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial x}\right) = \frac{\partial}{\partial \eta}\left[JF\left(\frac{\partial \eta}{\partial x}\right)\right] - F\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right)$$

Equation 2.62

$$J\left(\frac{\partial G}{\partial \xi}\right)\left(\frac{\partial \xi}{\partial y}\right)=\frac{\partial}{\partial \xi}\left[JG\left(\frac{\partial \xi}{\partial y}\right)\right]-G\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right)$$

Equation 2.63

$$J\left(\frac{\partial G}{\partial \eta}\right)\left(\frac{\partial \eta}{\partial y}\right)=\frac{\partial}{\partial \eta}\left[JG\left(\frac{\partial \eta}{\partial y}\right)\right]-G\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial y}\right)$$

Equation 2.64

Substituting them into the Equation 2.60, the result is this other equation:

$$J\left(\frac{\partial U}{\partial t}\right)+\frac{\partial}{\partial \xi}\left[JF\left(\frac{\partial \xi}{\partial x}\right)+JG\left(\frac{\partial \xi}{\partial y}\right)\right]+\frac{\partial}{\partial \eta}\left[JF\left(\frac{\partial \eta}{\partial x}\right)+JG\left(\frac{\partial \eta}{\partial y}\right)\right]-F\left[\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right)+\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right)\right]$$
$$-G\left[\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right)+\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial y}\right)\right]=J\cdot S$$

The terms multiplied by *F* and *G* can be rewritten by the means of the Equation 2.57, and be this way proven equal to zero.

$$\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial x}\right)+\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial x}\right) = \frac{\partial}{\partial \xi}\left(\frac{\partial y}{\partial \eta}\right)-\frac{\partial}{\partial \eta}\left(\frac{\partial y}{\partial \xi}\right) = \frac{\partial^2 y}{\partial \xi \partial \eta}-\frac{\partial^2 y}{\partial \eta \partial \xi} = 0$$

$$\frac{\partial}{\partial \xi}\left(J\frac{\partial \xi}{\partial y}\right)+\frac{\partial}{\partial \eta}\left(J\frac{\partial \eta}{\partial y}\right) = \frac{\partial}{\partial \xi}\left(-\frac{\partial x}{\partial \eta}\right)+\frac{\partial}{\partial \eta}\left(\frac{\partial x}{\partial \xi}\right) = \frac{\partial^2 x}{\partial \eta \partial \xi}-\frac{\partial^2 x}{\partial \xi \partial \eta} = 0$$

So, the initial goal is reached, and the governing equation is:

$$\frac{\partial U_1}{\partial t}+\frac{\partial F_1}{\partial \xi}+\frac{\partial G_1}{\partial \eta}=S_1$$

Equation 2.65

Where the new vectors $U_1, F_1, G_1$and $S_1$are related to the original vectors *U, F, G* and *S* as these expressions, using the **direct metrics**:

$$U_1=J\cdot U$$

Equation 2.66

$$F_1=J\cdot F\left(\frac{\partial \xi}{\partial x}\right)+J\cdot G\left(\frac{\partial \xi}{\partial y}\right)$$

Equation 2.67

$$G_1=J\cdot F\left(\frac{\partial \eta}{\partial x}\right)+J\cdot G\left(\frac{\partial \eta}{\partial y}\right)$$

Equation 2.68

$$S_1=J\cdot S$$

Equation 2.69

Alternatively, the vectors F_1 and G_1 can be calculated with the **inverse metrics**:

$$F_1 = F\left(\frac{\partial y}{\partial \eta}\right) - G\left(\frac{\partial x}{\partial \eta}\right)$$

Equation 2.70

$$G_1 = -F\left(\frac{\partial y}{\partial \xi}\right) + G\left(\frac{\partial x}{\partial \xi}\right)$$

Equation 2.71

The possibility of using both sets of expressions has been very important in the success of the project, for reasons that will be explained in the next chapter.

Exactly as it happened in subchapter 2.2, the variables that form part of the new vectors F_1 and G_1 must be differentiated in the opposite direction than the vector the form part of. As now the original vectors *F* and *G* both are into the calculation of the new ones, keeping track of what variable must be differentiated in what direction has been in some situations very messy. However, it is not difficult to follow if the concepts explained previously are clear and the procedure is done orderly, step by step.

If the viscosity terms stated in equations Equation 2.12, Equation 2.13 and Equation 2.14 and the temperature transfer terms $\frac{\partial T}{\partial x}$ and $\frac{\partial T}{\partial y}$ are transformed by the derivative relations Equation 2.50 and Equation 2.51, the results are the following:

$$\tau_{xx} = \frac{2}{3}\mu\left[2\left(\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x}\right) - \left(\frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial y}\right)\right]$$

Equation 2.72

$$\tau_{yy} = \frac{2}{3}\mu\left[2\left(\frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial y}\right) - \left(\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x}\right)\right]$$

Equation 2.73

$$\tau_{xy} = \tau_{yx} = \mu\left[\left(\frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial x}\right) + \left(\frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial y}\right)\right]$$

Equation 2.74

$$\frac{\partial T}{\partial x} = \frac{\partial T}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial T}{\partial \eta}\frac{\partial \eta}{\partial x}$$

Equation 2.75

$$\frac{\partial T}{\partial y} = \frac{\partial T}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial T}{\partial \eta}\frac{\partial \eta}{\partial y}$$

Equation 2.76

The terms $\frac{\partial u}{\partial \xi}, \frac{\partial u}{\partial \eta}, \frac{\partial v}{\partial \xi}, \frac{\partial v}{\partial \eta}, \frac{\partial T}{\partial \xi}$ and $\frac{\partial T}{\partial \eta}$ now must be calculated with forward, rearward, or central differentiation depending on the vector they are going to be used into. What happens is the following. The terms of equations Equation 2.72 to Equation 2.76 form part of the vectors *F* and *G*, and at the same time they form part of F_1 and G_1, which are differentiated differently.

With this reasoning, it becomes clear that two variants of the vectors F and G must be calculated, which will be called F_{F_1}, F_{G_1}, G_{F_1} and G_{G_1}. Each variant is designated to be used in the formation of one of the vectors F_1 or G_1, and is formed with variables differentiated accordingly to where they are going to be used.

The best way to visualize this concept is by looking at an example, in this case the viscosity term τ_{xx}. These equations show explicitly the differentiation used for this term when calculating F_1 and G_1 in the predictor step.

$$\left.{\tau_{xx}}^{t}_{i,j}\right|_{F_1} = \frac{2}{3}\mu^t_{i,j}\left[2\left(\frac{\partial\xi}{\partial x}\left(\frac{u^t_{i,j}-u^t_{i-1,j}}{\Delta\xi}\right)+\frac{\partial\eta}{\partial x}\left(\frac{u^t_{i,j+1}-u^t_{i,j-1}}{2\,\Delta\eta}\right)\right) - \left(\frac{\partial\xi}{\partial y}\left(\frac{v^t_{i,j}-v^t_{i-1,j}}{\Delta\xi}\right)+\frac{\partial\eta}{\partial y}\left(\frac{v^t_{i,j+1}-v^t_{i,j-1}}{2\,\Delta\eta}\right)\right)\right]$$

Equation 2.77

$$\left.{\tau_{xx}}^{t}_{i,j}\right|_{G_1} = \frac{2}{3}\mu^t_{i,j}\left[2\left(\frac{\partial\xi}{\partial x}\left(\frac{u^t_{i+1,j}-u^t_{i-1,j}}{2\,\Delta\xi}\right)+\frac{\partial\eta}{\partial x}\left(\frac{u^t_{i,j}-u^t_{i,j-1}}{\Delta\eta}\right)\right) - \left(\frac{\partial\xi}{\partial y}\left(\frac{v^t_{i+1,j}-v^t_{i-1,j}}{2\,\Delta\xi}\right)+\frac{\partial\eta}{\partial y}\left(\frac{v^t_{i,j}-v^t_{i,j-1}}{\Delta\eta}\right)\right)\right]$$

Equation 2.78

As said, the expressions can get confusing, but the compliance with the differentiation requisite is important for the second order accuracy of the method. In order to alleviate the section of extremely large expressions, not all the equivalents to Equation 2.77 and Equation 2.78 with the other variables are going to be shown. Just for comparison, appreciate how the differentiation used changes in the corrector step.

$$\left.{\bar{\tau}_{xx}}^{t}_{i,j}\right|_{\bar{F}_1} = \frac{2}{3}\bar{\mu}^t_{i,j}\left[2\left(\frac{\partial\xi}{\partial x}\left(\frac{\bar{u}^t_{i+1,j}-\bar{u}^t_{i,j}}{\Delta\xi}\right)+\frac{\partial\eta}{\partial x}\left(\frac{\bar{u}^t_{i,j+1}-\bar{u}^t_{i,j-1}}{2\,\Delta\eta}\right)\right) - \left(\frac{\partial\xi}{\partial y}\left(\frac{\bar{v}^t_{i+1,j}-\bar{v}^t_{i,j}}{\Delta\xi}\right)+\frac{\partial\eta}{\partial y}\left(\frac{\bar{v}^t_{i,j+1}-\bar{v}^t_{i,j-1}}{2\,\Delta\eta}\right)\right)\right]$$

Equation 2.79

$$\left.{\bar{\tau}_{xx}}^{t}_{i,j}\right|_{\bar{G}_1} = \frac{2}{3}\bar{\mu}^t_{i,j}\left[2\left(\frac{\partial\xi}{\partial x}\left(\frac{\bar{u}^t_{i+1,j}-\bar{u}^t_{i-1,j}}{2\,\Delta\xi}\right)+\frac{\partial\eta}{\partial x}\left(\frac{\bar{u}^t_{i,j+1}-\bar{u}^t_{i,j}}{\Delta\eta}\right)\right) - \left(\frac{\partial\xi}{\partial y}\left(\frac{\bar{v}^t_{i+1,j}-\bar{v}^t_{i-1,j}}{2\,\Delta\xi}\right)+\frac{\partial\eta}{\partial y}\left(\frac{\bar{v}^t_{i,j+1}-\bar{v}^t_{i,j}}{\Delta\eta}\right)\right)\right]$$

Equation 2.80

The correct implementation of these expressions has been one of the most difficult parts of the project, and its correctness is of vital importance. Despite not all the expressions appearing in this theoretical chapter, the reader can check their implementation by taking a look at the sections “`Encode variables for predictor step`” and “`Encode variables for corrector step`“ of the script that appears at the end of subchapter 3.2.

3 Numerical resolution

To codify the technique and perform the computation required to obtain the results, a programming language must be chosen to develop the code and then run it. In this project the program of choice has been MATLAB®, due to several reasons. Among them, that it is a program in which a considerable amount of focus is put into during the Aerospace Degree, and so it didn't require learning a new language. Also, the University of Seville has educational licenses and it can be easily accessed.

There have been a few scripts written to achieve the numerical resolution. However, they are structured as is going to be explained.

The base script is the one called "MainBasic.m". It represents the bedrock of the coding and is the one who calls the other scripts, that are defined as functions. Those functions are the responsible of setting the starting conditions, the boundary conditions, and assess the results, for instance to check the validity of the results and represent the plots. "MainBasic.m", as well as almost all the other scripts, have had multiple versions, denoted by the adding of, for example, "_v1" or "_v3" to the tittle.

The script is structured in sections, and the following subchapters are going to explain the high-level functioning of the script section by section. The different subscripts, called "auxiliary functions", vary depending on the problem that is going to be solved, thus will be treated in the next chapter for each application.

Finally, the chapter will end by explaining the modifications of the code that are needed to implement the non-uniform grid transformations by the method described in the subchapter 2.3. This chapter is mostly written to provide a comprehensive guide about how to understand and read the code if any person wants to take this project as base for a future project.

3.1 The Program for Uniform Grids

Before starting the MacCormack's technique to advance the simulation in time, the whole problem and a lot of variables must be set. The purpose of the sections before the MacCormack's technique begins is just that.

After deleting all current plots any pre-existing variables in the work space, the script "MainBasic.m" begins with the first section called "Properties of the air", in which the basic parameters of the air are set, such as the Prandtl number, the gas constant R, the ratio of specific heats γ, and the reference values at sea level T_0 and μ_0 needed for the Sutherland's law. Then c_v and c_p are calculated from the values of R and γ.

The second section is called "Entry air data", which specifies the parameters of the incoming air like Mach number and temperature. Following it begins the section "Geometry and grid definition". It determines the number of points in the grid, the security factor for the time step and also the dimensions of the computational domain.

The most important of the set-up sections is the "Initial Conditions" section. It calls the auxiliary function "Initial_Conditions_Application.m", that must return the value of the density, vertical and horizontal velocity, temperature and pressure at all grid points at the simulation starting time. The values of the internal energy e, viscosity μ and the heat transfer coefficient k are calculated from this data.

Due to the evolution of the program, the values of the predicted and the corrected variables, which are represented with the addition of "_p" and "_c" to the variables respectively, have to be set in the first iteration. So, they are simply made equal to the initial conditions. After the initial conditions are finished, the preparations end by allocating variables to speed up the code and specifying the maximum time and iterations for the simulation.

The advance in time is achieved by a loop "while", that checks in each iteration if the number of iterations or the time are greater than the maximums specified, if convergence has been achieved, or if a custom condition expressed by a "flag" variable has occurred. Before the loop, the time, iterations, convergence and flag are set to zero. All the stopping conditions are optional, and they can be disabled at will.

Inside the loop, the first section is dedicated to the calculation of the time step. A mathematically proven criterion has not been demonstrated jet, however, following the recommendation made by John Anderson in chapter 10 of [Ref 1], the Courant-Friedrichs-Lewy (CFL) criterion has been used to determine the time step which has been empirically proven enough to preserve a convergent behaviour.

$$(\Delta t_{CFL})_{i,j} = \left[\frac{|u_{i,j}|}{\Delta x} + \frac{|v_{i,j}|}{\Delta y} + a_{i,j}\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}} + 2v'_{i,j}\left(\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}\right)\right]^{-1}$$

Equation 3.1. Courant-Friedrichs-Lewy criterion.

Where

$$v'_{i,j} = max\left[\frac{\frac{4}{3}\mu_{i,j}(\gamma\mu_{i,j}/Pr)}{\rho_{i,j}}\right]$$

Equation 3.2

The time step is the minimum of the values of Δt_{CFL}, with a security factor that ensures the numerical stability that was set in the set-up phase.

$$\Delta t = K \cdot min(\Delta t_{CFL})_{i,j}$$

Equation 3.3

As to apply the boundary conditions, the variables have to be decoded after each step. The implementation of the predictor and corrector steps has multiple sections, and the structure of these sections is the following:

1. Calculation of the variables for the predictor step with the equations Equation 2.27 to Equation 2.32.

2. Formation of the vectors *U, F, G* and *S* as stated in the expressions Equation 2.6 to Equation 2.9.

3. Calculation of the predicted vector U_p with the Equation 2.26. The predicted variables have as common distinction the adding of "_p" to their names.

4. Decoding of the predicted primitive variables with the procedure explained at the beginning of section 2.2.

5. Enforcement of the boundary conditions on the predicted primitive variables.

This enforcement is not done directly in the "MainBasic.m" script. It is done by calling an auxiliary function called "Boundary_Conditions_Application.m", which changes with each application. This, together with the "Initial_Conditions_Application.m", allows great flexibility with the coding, as the main program is general and for each application only the specific initial and boundary condition scripts have to be written.

6. Calculation of the variables for the corrector step with equations equivalent to Equation 2.27 to Equation 2.32. Now, also new predicted values of viscosity and heat transfer coefficient must be calculated to use them in the formation of the "*conservative form*" vectors.

7. Formation of the predicted vectors *U, F, G* and *S* with the expressions Equation 2.6 to Equation 2.9, only that now the predicted values of the variables are used.

8. Calculation of the corrected vector U_c with the Equation 2.33. The corrected variables have as common distinction the adding of "_c" to their names.

9. Decoding of the corrected primitive variables with the procedure explained at the beginning of section 2.2.

10. Enforcement of the boundary conditions on the corrected primitive variables with the same "Boundary_Conditions_Application.m" function used in the predictor step.

11. Reformation of the vector U_c with the boundary conditions already applied.

12. Combination of the steps to achieve the final results with Equation 2.34.

The last part encompasses several sections with different purposes. The first section checks the convergence of the code by comparing the new obtained density at time $t + \Delta t$ with the density at time t from the previous iteration in each computational point. If the maximum difference is less than 10^{-8} the convergence variable is activated and, if the option is enabled, the program exits the loop at the end of the current iteration. As a comment, this condition could potentially be improved by substituting the static number with an expression that takes into consideration the time step.

The next section decodes the obtained vector $U^{t+\Delta t}$, erasing the old variables from the previous iteration in the process and after that the time and iterations are updated.

In the end, the current time and time step are displayed to allow a visual check of the correct execution of the program. This slightly slows the program, but not much, and is very helpful when dealing with a new script that may or may not work at the first time. Optionally, it can be enabled the plotting of a figure to see the evolution of the variables. However, this greatly slows the execution, and should only be used in exceptional situations.

Lastly, the variable “flag” can be used to stop the program if a custom condition has been met and the option is enabled. In the code for this project, it checks if the viscosity has any complex component, as this is signal that the simulation has somehow broken.

When the program ends the loop because one of the stopping conditions has been met, a check of the validity of the results has been implemented by integrating the mass flow on the boundaries of the computational domain. If the difference between the mass that enters the domain and the mass than exits it is less than one percent, the solution is considered valid. If the value is more than that, a warning is displayed and the amount of mass differential in percentage with respect to the mass inflow is also displayed.

Main program for rectangular and uniform grids “MainBasic.m”

```
clear all; close all; clc;
tic

%%%%%%%%%%% MAIN PROGRAM OF THE MACCORMACK'S METHOD %%%%%%%%%%

% FOR UNIFORM RECTANGULAR GRIDS

%----------------------------------------------------------------------
%% Air properties

% Prandlt number
Pr = 0.71;
% Specific heats ratio
gam = 1.4;
% Ideal gas constant
R = 287;
% Heat coefficients
c_v = R/(gam-1);
c_p = gam*c_v;
% Reference values
T0 = 298;
nu0 = 1.849e-5;

%----------------------------------------------------------------------
%% Air flow stream data

Minf = 4; % Choose the desired Mach number
Tinf = 298;
rhoinf = 1.225;
pinf = rhoinf*R*Tinf;
nuinf = nu0*(Tinf/T0).^(3/2).*((T0+110)./(Tinf+110));
uinf = Minf*sqrt(gam*R*Tinf);

%----------------------------------------------------------------------
%% Geometry and grid fineness

Nx = 70; Ny = 70; % Choose the adequate grid fineness
m = Nx-1; n = Ny-1;
K = 0.3; % Choose the adequate safe factor

LHOR = 0.00001; % Modify to fit the desire dimensions
Reinf = rhoinf*uinf*LHOR/nuinf;
delta = 5*LHOR/sqrt(Reinf);
LVER = 5*delta; % or LVER = fix value; % Choose at will

dx = LHOR /(Nx-1);
dy = LVER /(Ny-1);

%----------------------------------------------------------------------
%% Initial conditions

% Real variables
% Choose the correct auxiliary function depending on the application
[rho,u,v,T,p] = Initial_Conditions(rhoinf,uinf,Tinf,R,Nx,Ny);

e = c_v*T;
nu = nu0*T.*sqrt(T)/T0^(3/2).*((T0+110)./(T+110));
```

```
k = nu*c_p/Pr;

% Predictor variables
rho_p = rho;
u_p = u;
v_p = v;
T_p = T;
p_p = p;
e_p = e;
nu_p = nu;
k_p = k;

% Corrector variables
rho_c = rho;
u_c = u;
v_c = v;
T_c = T;
p_c = p;
e_c = e;
nu_c = nu;
k_c = k;

%---------------------------------------------------------------------
%% Maximum time and iterations

tmax = 4e-7;
itermax = 5000;

%---------------------------------------------------------------------
%% Allocate Variables

U = zeros(Nx,Ny,4);
F = zeros(Nx,Ny,4);
G = zeros(Nx,Ny,4);
S = zeros(Nx,Ny,4);
U_p = zeros(size(U));
U_c = zeros(size(U));
dudx = zeros(size(u));
dudy = zeros(size(u));
dvdx = zeros(size(u));
dvdy = zeros(size(u));
dTdx = zeros(size(u));
dTdy = zeros(size(u));

%---------------------------------------------------------------------
%% MacCormack's Method advance

t = 0;
iter = 0;
conver = 0;
flag = 0;

% Enable and disable at will the conditions to stop the loop
while t<tmax && flag==0 && conver==0 && iter<itermax

    %-----------------------------------------------------------------
    %% Time step calculation

    M = u./sqrt(gam*R*T);
    v_prim = max( max( (4*gam*nu.*nu)./(3.*rho*Pr) ) );
    dt = 1./(abs(u)/dx + abs(v)/dy + sqrt(gam*R*T)*sqrt(1/dx^2+1/dy^2)
+ 2*v_prim*(1/dx^2+1/dy^2));
    dt = K*min( min (dt) );
    %-----------------------------------------------------------------
    %% Encode variables for predictor step

    V = sqrt(u.*u + v.*v);

    % DR means Rearward Difference
    % DC means Central Difference
    % DP means Forward Difference

    % tauxx x:DR y:DC

    dudx(1,:) = ( u(2,:) - u(1,:) )/dx;
    dudx(2:Nx,:) = ( u(2:Nx,:) - u(1:Nx-1,:) )/dx;

    dvdy(:,1) = ( v(:,2) - v(:,1) )/dy;
    dvdy(:,2:n) = ( v(:,3:n+1) - v(:,1:n-1) )/(2*dy);
    dvdy(:,Ny) = ( v(:,Ny) - v(:,Ny-1) )/dy;

    tauxx = 2/3*nu.*( 2*dudx - dvdy );

    % tauyy x:DC y:DR

    dudx(1,:) = ( u(2,:) - u(1,:) )/dx;
    dudx(2:m,:) = ( u(3:m+1,:) - u(1:m-1,:) )/(2*dx);
    dudx(Nx,:) = ( u(Nx,:) - u(Nx-1,:) )/dx;

    dvdy(:,1) = ( v(:,2) - v(:,1) )/dy;
    dvdy(:,2:Ny) = ( v(:,2:Ny) - v(:,1:Ny-1) )/dy;

    tauyy = 2/3*nu.*( 2*dvdy - dudx );

    % tauxy_F x:DR y:DC

    dudy(:,1) = ( u(:,2) - u(:,1) )/dy;
    dudy(:,2:n) = ( u(:,3:n+1) - u(:,1:n-1) )/(2*dy);
    dudy(:,Ny) = ( u(:,Ny) - u(:,Ny-1) )/dy;

    dvdx(1,:) = ( v(2,:) - v(1,:) )/dx;
    dvdx(2:Nx,:) = ( v(2:Nx,:) - v(1:Nx-1,:) )/dx;

    tauxy_F = nu.*( dudy + dvdx );

    % tauxy_G x:DC y:DR

    dudy(:,1) = ( u(:,2) - u(:,1) )/dy;
    dudy(:,2:Ny) = ( u(:,2:Ny) - u(:,1:Ny-1) )/dy;

    dvdx(1,:) = ( v(2,:) - v(1,:) )/dx;
    dvdx(2:m,:) = ( v(3:m+1,:) - v(1:m-1,:) )/(2*dx);
    dvdx(Nx,:) = ( v(Nx,:) - v(Nx-1,:) )/dx;

    tauxy_G = nu.*( dudy + dvdx );
    % dTdx x:DR

    dTdx(1,:) = ( T(2,:) - T(1,:) )/dx;
    dTdx(2:Nx,:) = ( T(2:Nx,:) - T(1:Nx-1,:) )/dx;

    % dTdy y:DR
    dTdy(:,1) = ( T(:,2) - T(:,1) )/dy;
    dTdy(:,2:Ny) = ( T(:,2:Ny) - T(:,1:Ny-1) )/dy;

    %-----------------------------------------------------------------
    %% Predictor step: x:DP y:DP

    U(:,:,1) = rho;
    U(:,:,2) = rho.*u;
    U(:,:,3) = rho.*v;
    U(:,:,4) = rho.*(e+V.*V/2);

    F(:,:,1) = rho.*u;
    F(:,:,2) = rho.*u.*u + p - tauxx;
    F(:,:,3) = rho.*v.*u - tauxy_F;
    F(:,:,4) = rho.*(e+V.*V/2).*u + p.*u - k.*dTdx - u.*tauxx -
v.*tauxy_F;

    G(:,:,1) = rho.*v;
    G(:,:,2) = rho.*u.*v - tauxy_G;
    G(:,:,3) = rho.*v.*v + p - tauyy;
    G(:,:,4) = rho.*(e+V.*V/2).*v + p.*v - k.*dTdy - u.*tauxy_G -
v.*tauyy;

    U_p(2:m,2:n,:) = U(2:m,2:n,:) - dt*(F(3:m+1,2:n,:)-
F(2:m,2:n,:))/dx - dt*(G(2:m,3:n+1,:)-G(2:m,2:n,:))/dy +
dt*S(2:m,2:n,:);

    %-----------------------------------------------------------------
    %% Decode variables of predictor step

    rho_p(2:m,2:n) = U_p(2:m,2:n,1);
    u_p(2:m,2:n) = U_p(2:m,2:n,2)./U_p(2:m,2:n,1);
    v_p(2:m,2:n) = U_p(2:m,2:n,3)./U_p(2:m,2:n,1);
    e_p(2:m,2:n) = U_p(2:m,2:n,4)./U_p(2:m,2:n,1) -
(u_p(2:m,2:n).*u_p(2:m,2:n) + v_p(2:m,2:n).*v_p(2:m,2:n))/2;
    T_p(2:m,2:n) = e_p(2:m,2:n)/c_v;
    p_p(2:m,2:n) = R*rho_p(2:m,2:n).*T_p(2:m,2:n);

    %-----------------------------------------------------------------
    %% Boundary conditions of predictor step

    % Choose the correct auxiliary function depending on the
application
    [rho_p,u_p,v_p,T_p,p_p] =
Boundary_Conditions(rho_p,u_p,v_p,T_p,p_p,Nx,Ny);

    e_p(Nx,2:Ny-1) = c_v*T_p(Nx,2:Ny-1);

    %-----------------------------------------------------------------
    %% Encode variables for corrector step

    V_p = sqrt(u_p.*u_p + v_p.*v_p);
    nu_p = nu0*(T_p/T0).^(3/2).*((T0+110)./(T_p+110));
```

```
k_p = nu_p*c_p/Pr;

% tauxx x:DP y:DC

dudx(1:m,:) = ( u_p(2:m+1,:) - u_p(1:m,:) )/dx;
dudx(Nx,:) = ( u_p(Nx,:) - u_p(Nx-1,:) )/dx;

dvdy(:,1) = ( v_p(:,2) - v_p(:,1) )/dy;
dvdy(:,2:n) = ( v_p(:,3:n+1) - v_p(:,1:n-1) )/(2*dy);
dvdy(:,Ny) = ( v_p(:,Ny) - v_p(:,Ny-1) )/dy;

tauxx_p = 2/3*nu_p.*( 2*dudx - dvdy );

% tauyy x:DC y:DP

dudx(1,:) = ( u_p(2,:) - u_p(1,:) )/dx;
dudx(2:m,:) = ( u_p(3:m+1,:) - u_p(1:m-1,:) )/(2*dx);
dudx(Nx,:) = ( u_p(Nx,:) - u_p(Nx-1,:) )/dx;

dvdy(:,1:n) = ( v_p(:,2:n+1) - v_p(:,1:n) )/dy;
dvdy(:,Ny) = ( v_p(:,Ny) - v_p(:,Ny-1) )/dy;

tauyy_p = 2/3*nu_p.*( 2*dvdy - dudx );

% tauxy_F x:DP y:DC

dudy(:,1) = ( u_p(:,2) - u_p(:,1) )/dy;
dudy(:,2:n) = ( u_p(:,3:n+1) - u_p(:,1:n-1) )/(2*dy);
dudy(:,Ny) = ( u_p(:,Ny) - u_p(:,Ny-1) )/dy;

dvdx(1:m,:) = ( v_p(2:m+1,:) - v_p(1:m,:) )/dx;
dvdx(Nx,:) = ( v_p(Nx,:) - v_p(Nx-1,:) )/dx;

tauxy_F_p = nu_p.*( dudy + dvdx );

% tauxy_G x:DC y:DP

dudy(:,1:n) = ( u_p(:,2:n+1) - u_p(:,1:n) )/dy;
dudy(:,Ny) = ( u_p(:,Ny) - u_p(:,Ny-1) )/dy;

dvdx(1,:) = ( v_p(2,:) - v_p(1,:) )/dx;
dvdx(2:m,:) = ( v_p(3:m+1,:) - v_p(1:m-1,:) )/(2*dx);
dvdx(Nx,:) = ( v_p(Nx,:) - v_p(Nx-1,:) )/dx;

tauxy_G_p = nu_p.*( dudy + dvdx );

% dTdx x:DP

dTdx(1:m,:) = ( T_p(2:m+1,:) - T_p(1:m,:) )/dx;
dTdx(Nx,:) = ( T_p(Nx,:) - T_p(Nx-1,:) )/dx;

% dTdy y:DP
dTdy(:,1:n) = ( T_p(:,2:n+1) - T_p(:,1:n) )/dy;
dTdy(:,Ny) = ( T_p(:,Ny) - T_p(:,Ny-1) )/dy;

%------------------------------------------------------------------
%% Corrector step: x:DR y:DR
```

```
    U_p(:,:,1) = rho_p;
    U_p(:,:,2) = rho_p.*u_p;
    U_p(:,:,3) = rho_p.*v_p;
    U_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2);

    F_p(:,:,1) = rho_p.*u_p;
    F_p(:,:,2) = rho_p.*u_p.*u_p + p_p - tauxx_p;
    F_p(:,:,3) = rho_p.*v_p.*u_p - tauxy_F_p;
    F_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*u_p + p_p.*u_p - k_p.*dTdx -
u_p.*tauxx_p - v_p.*tauxy_F_p;

    G_p(:,:,1) = rho_p.*v_p;
    G_p(:,:,2) = rho_p.*u_p.*v_p - tauxy_G_p;
    G_p(:,:,3) = rho_p.*v_p.*v_p + p_p - tauyy_p;
    G_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*v_p + p_p.*v_p - k_p.*dTdy -
u_p.*tauxy_G_p - v_p.*tauyy_p;

    U_c(2:m,2:n,:) = U(2:m,2:n,:) - dt*(F_p(2:m,2:n,:)-F_p(1:m-
1,2:n,:))/dx - dt*(G_p(2:m,2:n,:)-G_p(2:m,1:n-1,:))/dy +
dt*S(2:m,2:n,:);

    %-----------------------------------------------------------
    %% Decode variables of corrector step

    rho_c(2:m,2:n) = U_c(2:m,2:n,1);
    u_c(2:m,2:n) = U_c(2:m,2:n,2)./U_c(2:m,2:n,1);
    v_c(2:m,2:n) = U_c(2:m,2:n,3)./U_c(2:m,2:n,1);
    e_c(2:m,2:n) = U_c(2:m,2:n,4)./U_c(2:m,2:n,1) -
(u_c(2:m,2:n).*u_c(2:m,2:n) + v_c(2:m,2:n).*v_c(2:m,2:n))/2;
    T_c(2:m,2:n) = e_c(2:m,2:n)/c_v;
%     p_c(2:m,2:n) = R*rho_c(2:m,2:n).*T_c(2:m,2:n); % not necessary
decoding

    %-----------------------------------------------------------
    %% Boundary conditions of corrector step

    [rho_c,u_c,v_c,T_c,p_c] =
Bounday_Conditions(rho_c,u_c,v_c,T_c,p_c,Nx,Ny);

    e_c(Nx,2:Ny-1) = c_v*T_c(Nx,2:Ny-1);

    %-----------------------------------------------------------
    %% Encode variables for result and calculation of result

    U_c(:,:,1) = rho_c;
    U_c(:,:,2) = rho_c.*u_c;
    U_c(:,:,3) = rho_c.*v_c;
    U_c(:,:,4) = rho_c.*(e_c + (u_c.*u_c + v_c.*v_c)/2);

    U = (U_p + U_c)/2;

    %-----------------------------------------------------------
    %% Check for convergence

    if max( max( abs( U(:,:,1) - rho ) ) )>1e-8
        conver = 0;
    else
        conver = 1;
    end

    %------------------------------------------------------------------
    %% Decode final variables

    rho = U(:,:,1);
    u = U(:,:,2)./U(:,:,1);
    v = U(:,:,3)./U(:,:,1);
    e = U(:,:,4)./U(:,:,1) - (u.*u + v.*v)/2;
    T = e./c_v;
    p = R*rho.*T;
    p(:,1) = 2*p(:,2) - p(:,3);

    [rho,u,v,T,p] = Boundary_Conditions(rho,u,v,T,p,Nx,Ny);

    nu = nu0*(T/T0).^(3/2).*((T0+110)./(T+110));
    k = nu*c_p/Pr;

    % Update time and iteration counts
    iter = iter + 1;
    t = t + dt;

    % Print at screen to check for correct execution
    display([t dt])
    if max(max(abs(imag(nu))))>0 && flag==0
        tbreak = t;
        flag = 1;
    end

%     mesh(1:Ny,1:Nx,real(p))
%     pause(0.01)

end

%----------------------------------------------------------------------
%% Check the validity of the results by calculating the mass inflow
and outflow balance

Mass_in = trapz(0:dy:LVER,u(1,:).*rho(1,:));
Mass_out = trapz(0:dy:LVER,u(Nx,:).*rho(Nx,:));
if abs( ( Mass_in - Mass_out ) / Mass_in*100 ) <= 1
    display('The solution seems valid checking the mass balance')
else
    display('The solution seems to violate the mass balance, it is not
valid')
end

%----------------------------------------------------------------------
%% Viscosity force at the wall

f_pared = nu(:,1).*( u(:,2) - u(:,1) )/dy;
F_pared = trapz(0:dx:LHOR,f_pared);
C_d = F_pared/0.5/rhoinf/uinf^2/LHOR;

F_press = trapz(0:dx:LHOR,p(:,1));

%----------------------------------------------------------------------
%% Plots and ending
```

```
plot(1:Nx,p(:,1))
figure(2)
mesh(0:dy:LVER,0:dx:LHOR,real(p))
display(conver)
time = toc

% Additional plots and calculations using the results may be done at
% different scripts
```

3.2 Non-Uniform Grid Modifications

This subchapter will explain the modifications that has been made to the code in order to implement the non-uniform grid transformations as they were explained in the subchapter 2.3. The new main program is now called “MainGridTrans_DirectMetrics.m”.

The first difference is that the set-up section “Geometry and grid definition” is now much more complex. It makes use of an auxiliary, application specific function, which must return the coordinates x and y of the grid points, the Jacobian at each point, and the direct and inverse metrics. In principle, with the relations given by Equation 2.57, only one set of metrics could be used. However, as it will be seen later, the inverse metrics are necessary at one point of the coding. Using the x and y coordinates, the distance between points Δx and Δy must be calculated too to use them in the Courant-Friedrichs-Lewy criterion.

As it has been said before, the fact that both vectors *F* and *G* form part of the new F_1 and G_1 forces the formation of two variants of *F* and *G*, to comply with the opposite differentiation needed for the second order accuracy. Thus, the vectors F_{F_1}, F_{G_1}, G_{F_1} and G_{G_1} are formed, each using the adequate variables. Then, the vectors F_{F_1} and G_{F_1} are used in the Equation 2.70, together with the inverse metrics, to form F_1. The same happens with F_{G_1} and G_{G_1} that are used to form G_1 with Equation 2.71.

An important commentary has to be made here. In the formation of U_1 the Jacobian “*J*” is used, and it is calculated using the inverse metrics. When exact analytical expressions are used, the mixing of both direct and inverse metrics seems to be irrelevant. However, if the metrics are approximately calculated with central differentiation, as it is the case in most of the applications in this project, it exists a small error between the two sets. This very small error **adds-up** iteration after iteration and ends up making the results inexact. To avoid that, the expressions used to calculate F_1 and G_1 have been Equation 2.70 and Equation 2.71, which use inverse metrics, instead of Equation 2.68 and Equation 2.69, which because of using direct metrics produced this very subtle error.

Main program for non-uniform grids “MainGridTrans_DirectMetrics.m”

```
clear all; close all; clc;
tic

%%%%%%%%%%% MAIN PROGRAM OF THE MACCORMACK'S METHOD %%%%%%%%%%

% FOR NON-UNIFORM GRID TRANSFORMATIONS

%----------------------------------------------------------------------
%% Air properties

% Prandlt number
Pr = 0.71;
% Specific heats ratio
gam = 1.4;
% Ideal gas constant
R = 287;
% Heat coefficients
c_v = R/(gam-1);
c_p = gam*c_v;
% Reference values
T0 = 298;
nu0 = 1.849e-5;

%----------------------------------------------------------------------
%% Air flow stream data

Minf = 4; % Choose the desired Mach number
Tinf = 298;
rhoinf = 1.225;
pinf = rhoinf*R*Tinf;
nuinf = nu0*(Tinf/T0).^(3/2).*((T0+110)./(Tinf+110));
uinf = Minf*sqrt(gam*R*Tinf);

%----------------------------------------------------------------------
%% Geometry and grid fineness

Nxi = 100; Neta = 150; % Choose the adequate grid fineness
m = Nxi-1; n = Neta-1;
K = 0.1; % Choose the adequate safe factor

LHOR = 0.00001; % Modify to fit the desire dimensions
Reinf = rhoinf*uinf*LHOR/nuinf;
delta = 5*LHOR/sqrt(Reinf);
LVER = 1.6345e-05; % or LVER = value*delta; % Choose at will

[x,y,dxi,deta,dxidx,dxidy,detadx,detady,J,dxdxi,dxdeta,dydxi,dydeta] =
Geometry_Function(Nxi,Neta,LHOR,LVER);

delta_x(1,:) = x(2,:) - x(1,:);
delta_x(2:m,:) = ( x(3:m+1,:) - x(1:m-1,:) )/2;
delta_x(Nxi,:) = x(Nxi,:) - x(Nxi-1,:);

delta_y(:,1) = y(:,2) - y(:,1);
delta_y(:,2:n) = ( y(:,3:n+1) - y(:,1:n-1) )/2;
delta_y(:,Neta) = y(:,Neta) - y(:,Neta-1);
```

```
%-----------------------------------------------------------------------
%% Initial conditions

% Real variables
% Choose the correct auxiliary function depending on the application
[rho,u,v,T,p] = Initial_Conditions(rhoinf,uinf,Tinf,R,Nxi,Neta);

e = c_v*T;
nu = nu0*T.*sqrt(T)/T0^(3/2).*((T0+110)./(T+110));
k = nu*c_p/Pr;

% Predictor variables
rho_p = rho;
u_p = u;
v_p = v;
T_p = T;
p_p = p;
e_p = e;
nu_p = nu;
k_p = k;

% Corrector variables
rho_c = rho;
u_c = u;
v_c = v;
T_c = T;
p_c = p;
e_c = e;
nu_c = nu;
k_c = k;

%-----------------------------------------------------------------------
%% Maximum time and iterations

tmax = 2.5e-07; %1.5e-7;
itermax = 5000;

%-----------------------------------------------------------------------
%% Allocate Variables

U_p = zeros(Nxi,Neta,4);
U_c = zeros(Nxi,Neta,4);
S = zeros(Nxi,Neta,4);

%-----------------------------------------------------------------------
%% MacCormack's Method advance

t = 0;
iter = 0;
conver = 0;
flag = 0;

% Enable and disable at will the conditions to stop the loop
while t<tmax && flag==0 && conver==0 && iter<itermax

    %-------------------------------------------------------------------
    %% Time step calculation

    M = u./sqrt(gam*R*T);
    v_prim = max( max( (4*gam*nu.*nu)./(3.*rho*Pr) ) );
    dt = 1./( abs(u)./delta_x + abs(v)./delta_y +
sqrt(gam*R*T).*sqrt(1./delta_x.^2+1./delta_y.^2) +
2*v_prim.*(1./delta_x.^2+1./delta_y.^2) );
    dt = K*min( min (dt) );

    %-----------------------------------------------------------------
    %% Encode variables for predictor step

    V = sqrt(u.*u + v.*v);

    % DR means Rearward Difference
    % DC means Central Difference
    % DP means Forward Difference

    % xi:DR
    dudxi_R(1,:) = ( u(2,:) - u(1,:) )/dxi;
    dudxi_R(2:Nxi,:) = ( u(2:Nxi,:) - u(1:Nxi-1,:) )/dxi;

    dvdxi_R(1,:) = ( v(2,:) - v(1,:) )/dxi;
    dvdxi_R(2:Nxi,:) = ( v(2:Nxi,:) - v(1:Nxi-1,:) )/dxi;

    dTdxi_R(1,:) = ( T(2,:) - T(1,:) )/dxi;
    dTdxi_R(2:Nxi,:) = ( T(2:Nxi,:) - T(1:Nxi-1,:) )/dxi;

    % xi:DC
    dudxi_C(1,:) = ( u(2,:) - u(1,:) )/dxi;
    dudxi_C(2:m,:) = ( u(3:m+1,:) - u(1:m-1,:) )/(2*dxi);
    dudxi_C(Nxi,:) = ( u(Nxi,:) - u(Nxi-1,:) )/dxi;

    dvdxi_C(1,:) = ( v(2,:) - v(1,:) )/dxi;
    dvdxi_C(2:m,:) = ( v(3:m+1,:) - v(1:m-1,:) )/(2*dxi);
    dvdxi_C(Nxi,:) = ( v(Nxi,:) - v(Nxi-1,:) )/dxi;

    dTdxi_C(1,:) = ( T(2,:) - T(1,:) )/dxi;
    dTdxi_C(2:m,:) = ( T(3:m+1,:) - T(1:m-1,:) )/(2*dxi);
    dTdxi_C(Nxi,:) = ( T(Nxi,:) - T(Nxi-1,:) )/dxi;

    % eta:DR
    dudeta_R(:,1) = ( u(:,2) - u(:,1) )/deta;
    dudeta_R(:,2:Neta) = ( u(:,2:Neta) - u(:,1:Neta-1) )/deta;

    dvdeta_R(:,1) = ( v(:,2) - v(:,1) )/deta;
    dvdeta_R(:,2:Neta) = ( v(:,2:Neta) - v(:,1:Neta-1) )/deta;

    dTdeta_R(:,1) = ( T(:,2) - T(:,1) )/deta;
    dTdeta_R(:,2:Neta) = ( T(:,2:Neta) - T(:,1:Neta-1) )/deta;

    % eta:DC
    dudeta_C(:,1) = ( u(:,2) - u(:,1) )/deta;
    dudeta_C(:,2:n) = ( u(:,3:n+1) - u(:,1:n-1) )/(2*deta);
    dudeta_C(:,Neta) = ( u(:,Neta) - u(:,Neta-1) )/deta;

    dvdeta_C(:,1) = ( v(:,2) - v(:,1) )/deta;
    dvdeta_C(:,2:n) = ( v(:,3:n+1) - v(:,1:n-1) )/(2*deta);
    dvdeta_C(:,Neta) = ( v(:,Neta) - v(:,Neta-1) )/deta;

    dTdeta_C(:,1) = ( T(:,2) - T(:,1) )/deta;
    dTdeta_C(:,2:n) = ( T(:,3:n+1) - T(:,1:n-1) )/(2*deta);
    dTdeta_C(:,Neta) = ( T(:,Neta) - T(:,Neta-1) )/deta;

    % tauxx
    tauxx_F1 = 2/3*nu.*( 2*( dxidx.*dudxi_R + detadx.*dudeta_C ) - (
dxidy.*dvdxi_R + detady.*dvdeta_C ) );
    tauxx_G1 = 2/3*nu.*( 2*( dxidx.*dudxi_C + detadx.*dudeta_R ) - (
dxidy.*dvdxi_C + detady.*dvdeta_R ) );

    % tauyy
    tauyy_F1 = 2/3*nu.*( 2*( dxidy.*dvdxi_R + detady.*dvdeta_C ) - (
dxidx.*dudxi_R + detadx.*dudeta_C ) );
    tauyy_G1 = 2/3*nu.*( 2*( dxidy.*dvdxi_C + detady.*dvdeta_R ) - (
dxidx.*dudxi_C + detadx.*dudeta_R ) );

    % tauxy
    tauxy_F1 = nu.*( ( dxidy.*dudxi_R + detady.*dudeta_C ) + (
dxidx.*dvdxi_R + detadx.*dvdeta_C ) );
    tauxy_G1 = nu.*( ( dxidy.*dudxi_C + detady.*dudeta_R ) + (
dxidx.*dvdxi_C + detadx.*dvdeta_R ) );

    % dTdx
    dTdx_F1 = dxidx.*dTdxi_R + detadx.*dTdeta_C;
    dTdx_G1 = dxidx.*dTdxi_C + detadx.*dTdeta_R;

    % dTdy
    dTdy_F1 = dxidy.*dTdxi_R + detady.*dTdeta_C;
    dTdy_G1 = dxidy.*dTdxi_C + detady.*dTdeta_R;

    %------------------------------------------------------------------
    %% Predictor step:  xi:DP  eta:DP

    U(:,:,1) = rho;
    U(:,:,2) = rho.*u;
    U(:,:,3) = rho.*v;
    U(:,:,4) = rho.*(e+V.*V/2);

    F_F1(:,:,1) = rho.*u;
    F_F1(:,:,2) = rho.*u.*u + p - tauxx_F1;
    F_F1(:,:,3) = rho.*v.*u - tauxy_F1;
    F_F1(:,:,4) = rho.*(e+V.*V/2).*u + p.*u - k.*dTdx_F1 - u.*tauxx_F1
- v.*tauxy_F1;

    G_F1(:,:,1) = rho.*v;
    G_F1(:,:,2) = rho.*u.*v - tauxy_F1;
    G_F1(:,:,3) = rho.*v.*v + p - tauyy_F1;
    G_F1(:,:,4) = rho.*(e+V.*V/2).*v + p.*v - k.*dTdy_F1 - u.*tauxy_F1
- v.*tauyy_F1;

    F_G1(:,:,1) = rho.*u;
    F_G1(:,:,2) = rho.*u.*u + p - tauxx_G1;
    F_G1(:,:,3) = rho.*v.*u - tauxy_G1;
    F_G1(:,:,4) = rho.*(e+V.*V/2).*u + p.*u - k.*dTdx_G1 - u.*tauxx_G1
- v.*tauxy_G1;

    G_G1(:,:,1) = rho.*v;
    G_G1(:,:,2) = rho.*u.*v - tauxy_G1;
    G_G1(:,:,3) = rho.*v.*v + p - tauyy_G1;
    G_G1(:,:,4) = rho.*(e+V.*V/2).*v + p.*v - k.*dTdy_G1 - u.*tauxy_G1
- v.*tauyy_G1;

    for i=1:4
        U1(:,:,i) = J.*U(:,:,i);
        F1(:,:,i) = F_F1(:,:,i).*dydeta - G_F1(:,:,i).*dxdeta;
        G1(:,:,i) = - F_G1(:,:,i).*dydxi + G_G1(:,:,i).*dxdxi;
    end

    U1_p(2:m,2:n,:) = U1(2:m,2:n,:) - dt*(F1(3:m+1,2:n,:)-
F1(2:m,2:n,:))/dxi - dt*(G1(2:m,3:n+1,:)-G1(2:m,2:n,:))/deta +
dt*S(2:m,2:n,:);

    %-----------------------------------------------------------------
    %% Decode variables of predictor step

    for i=1:4
        U_p(2:m,2:n,i) = U1_p(2:m,2:n,i)./J(2:m,2:n);
    end

    rho_p(2:m,2:n) = U_p(2:m,2:n,1);
    u_p(2:m,2:n) = U_p(2:m,2:n,2)./U_p(2:m,2:n,1);
    v_p(2:m,2:n) = U_p(2:m,2:n,3)./U_p(2:m,2:n,1);
    e_p(2:m,2:n) = U_p(2:m,2:n,4)./U_p(2:m,2:n,1) -
(u_p(2:m,2:n).*u_p(2:m,2:n) + v_p(2:m,2:n).*v_p(2:m,2:n))/2;
    T_p(2:m,2:n) = e_p(2:m,2:n)/c_v;
    p_p(2:m,2:n) = R*rho_p(2:m,2:n).*T_p(2:m,2:n);

    %-----------------------------------------------------------------
    %% Boundary conditions of predictor step

    % Choose the correct auxiliary function depending on the
application
    [rho_p,u_p,v_p,T_p,p_p] =
Boundary_Conditions(rho_p,u_p,v_p,T_p,p_p,Nxi,Neta);

    e_p(Nxi,2:Neta-1) = c_v*T_p(Nxi,2:Neta-1);

    %-----------------------------------------------------------------
    %% Encode variables for corrector step

    V_p = sqrt(u_p.*u_p + v_p.*v_p);
    nu_p = nu0*(T_p/T0).^(3/2).*((T0+110)./(T_p+110));
    k_p = nu_p*c_p/Pr;

    % xi:DP
    dudxi_P_p(1:m,:) = ( u_p(2:m+1,:) - u_p(1:m,:) )/dxi;
    dudxi_P_p(Nxi,:) = ( u_p(Nxi,:) - u_p(Nxi-1,:) )/dxi;

    dvdxi_P_p(1:m,:) = ( v_p(2:m+1,:) - v_p(1:m,:) )/dxi;
    dvdxi_P_p(Nxi,:) = ( v_p(Nxi,:) - v_p(Nxi-1,:) )/dxi;

    dTdxi_P_p(1:m,:) = ( T_p(2:m+1,:) - T_p(1:m,:) )/dxi;
    dTdxi_P_p(Nxi,:) = ( T_p(Nxi,:) - T_p(Nxi-1,:) )/dxi;

    % xi:DC
    dudxi_C_p(1,:) = ( u_p(2,:) - u_p(1,:) )/dxi;
    dudxi_C_p(2:m,:) = ( u_p(3:m+1,:) - u_p(1:m-1,:) )/(2*dxi);
    dudxi_C_p(Nxi,:) = ( u_p(Nxi,:) - u_p(Nxi-1,:) )/dxi;
    dvdxi_C_p(1,:) = ( v_p(2,:) - v_p(1,:) )/dxi;
    dvdxi_C_p(2:m,:) = ( v_p(3:m+1,:) - v_p(1:m-1,:) )/(2*dxi);
    dvdxi_C_p(Nxi,:) = ( v_p(Nxi,:) - v_p(Nxi-1,:) )/dxi;

    dTdxi_C_p(1,:) = ( T_p(2,:) - T_p(1,:) )/dxi;
    dTdxi_C_p(2:m,:) = ( T_p(3:m+1,:) - T_p(1:m-1,:) )/(2*dxi);
    dTdxi_C_p(Nxi,:) = ( T_p(Nxi,:) - T_p(Nxi-1,:) )/dxi;

    % eta:DP
    dudeta_P_p(:,1:n) = ( u_p(:,2:n+1) - u_p(:,1:n) )/deta;
    dudeta_P_p(:,Neta) = ( u_p(:,Neta) - u_p(:,Neta-1) )/deta;

    dvdeta_P_p(:,1:n) = ( v_p(:,2:n+1) - v_p(:,1:n) )/deta;
    dvdeta_P_p(:,Neta) = ( v_p(:,Neta) - v_p(:,Neta-1) )/deta;

    dTdeta_P_p(:,1:n) = ( T_p(:,2:n+1) - T_p(:,1:n) )/deta;
    dTdeta_P_p(:,Neta) = ( T_p(:,Neta) - T_p(:,Neta-1) )/deta;

    % eta:DC
    dudeta_C_p(:,1) = ( u_p(:,2) - u_p(:,1) )/deta;
    dudeta_C_p(:,2:n) = ( u_p(:,3:n+1) - u_p(:,1:n-1) )/(2*deta);
    dudeta_C_p(:,Neta) = ( u_p(:,Neta) - u_p(:,Neta-1) )/deta;

    dvdeta_C_p(:,1) = ( v_p(:,2) - v_p(:,1) )/deta;
    dvdeta_C_p(:,2:n) = ( v_p(:,3:n+1) - v_p(:,1:n-1) )/(2*deta);
    dvdeta_C_p(:,Neta) = ( v_p(:,Neta) - v_p(:,Neta-1) )/deta;

    dTdeta_C_p(:,1) = ( T_p(:,2) - T_p(:,1) )/deta;
    dTdeta_C_p(:,2:n) = ( T_p(:,3:n+1) - T_p(:,1:n-1) )/(2*deta);
    dTdeta_C_p(:,Neta) = ( T_p(:,Neta) - T_p(:,Neta-1) )/deta;

    % tauxx
    tauxx_F1_p = 2/3*nu.*( 2*( dxidx.*dudxi_P_p + detadx.*dudeta_C_p )
- ( dxidy.*dvdxi_P_p + detady.*dvdeta_C_p ) );
    tauxx_G1_p = 2/3*nu.*( 2*( dxidx.*dudxi_C_p + detadx.*dudeta_P_p )
- ( dxidy.*dvdxi_C_p + detady.*dvdeta_P_p ) );

    % tauyy
    tauyy_F1_p = 2/3*nu.*( 2*( dxidy.*dvdxi_P_p + detady.*dvdeta_C_p )
- ( dxidx.*dudxi_P_p + detadx.*dudeta_C_p ) );
    tauyy_G1_p = 2/3*nu.*( 2*( dxidy.*dvdxi_C_p + detady.*dvdeta_P_p )
- ( dxidx.*dudxi_C_p + detadx.*dudeta_P_p ) );

    % tauxy
    tauxy_F1_p = nu.*( ( dxidy.*dudxi_P_p + detady.*dudeta_C_p ) + (
dxidx.*dvdxi_P_p + detadx.*dvdeta_C_p ) );
    tauxy_G1_p = nu.*( ( dxidy.*dudxi_C_p + detady.*dudeta_P_p ) + (
dxidx.*dvdxi_C_p + detadx.*dvdeta_P_p ) );

    % dTdx
    dTdx_F1_p = dxidx.*dTdxi_P_p + detadx.*dTdeta_C_p;
    dTdx_G1_p = dxidx.*dTdxi_C_p + detadx.*dTdeta_P_p;

    % dTdy
    dTdy_F1_p = dxidy.*dTdxi_P_p + detady.*dTdeta_C_p;
    dTdy_G1_p = dxidy.*dTdxi_C_p + detady.*dTdeta_P_p;

   %---------------------------------------------------------
```

```
    %% Corrector step: xi:DR eta:DR

    U_p(:,:,1) = rho_p;
    U_p(:,:,2) = rho_p.*u_p;
    U_p(:,:,3) = rho_p.*v_p;
    U_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2);

    F_F1_p(:,:,1) = rho_p.*u_p;
    F_F1_p(:,:,2) = rho_p.*u_p.*u_p + p_p - tauxx_F1_p;
    F_F1_p(:,:,3) = rho_p.*v_p.*u_p - tauxy_F1_p;
    F_F1_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*u_p + p_p.*u_p - 
k_p.*dTdx_F1_p - u_p.*tauxx_F1_p - v_p.*tauxy_F1_p;

    G_F1_p(:,:,1) = rho_p.*v_p;
    G_F1_p(:,:,2) = rho_p.*u_p.*v_p - tauxy_F1_p;
    G_F1_p(:,:,3) = rho_p.*v_p.*v_p + p_p - tauyy_F1_p;
    G_F1_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*v_p + p_p.*v_p - 
k_p.*dTdy_F1_p - u_p.*tauxy_F1_p - v_p.*tauyy_F1_p;

    F_G1_p(:,:,1) = rho_p.*u_p;
    F_G1_p(:,:,2) = rho_p.*u_p.*u_p + p_p - tauxx_G1_p;
    F_G1_p(:,:,3) = rho_p.*v_p.*u_p - tauxy_G1_p;
    F_G1_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*u_p + p_p.*u_p - 
k_p.*dTdx_G1_p - u_p.*tauxx_G1_p - v_p.*tauxy_G1_p;

    G_G1_p(:,:,1) = rho_p.*v_p;
    G_G1_p(:,:,2) = rho_p.*u_p.*v_p - tauxy_G1_p;
    G_G1_p(:,:,3) = rho_p.*v_p.*v_p + p_p - tauyy_G1_p;
    G_G1_p(:,:,4) = rho_p.*(e_p+V_p.*V_p/2).*v_p + p_p.*v_p - 
k_p.*dTdy_G1_p - u_p.*tauxy_G1_p - v_p.*tauyy_G1_p;

    for i=1:4
        F1_p(:,:,i) = F_F1_p(:,:,i).*dydeta - G_F1_p(:,:,i).*dxdeta;
        G1_p(:,:,i) = - F_G1_p(:,:,i).*dydxi + G_G1_p(:,:,i).*dxdxi;
    end

    U1_c(2:m,2:n,:) = U1(2:m,2:n,:) - dt*(F1_p(2:m,2:n,:)-F1_p(1:m-
1,2:n,:))/dxi - dt*(G1_p(2:m,2:n,:)-G1_p(2:m,1:n-1,:))/deta + 
dt*S(2:m,2:n,:);

    %-----------------------------------------------------------
    %% Decode variables of corrector step

    for i=1:4
        U_c(2:m,2:n,i) = U1_c(2:m,2:n,i)./J(2:m,2:n);
    end

    rho_c(2:m,2:n) = U_c(2:m,2:n,1);
    u_c(2:m,2:n) = U_c(2:m,2:n,2)./U_c(2:m,2:n,1);
    v_c(2:m,2:n) = U_c(2:m,2:n,3)./U_c(2:m,2:n,1);
    e_c(2:m,2:n) = U_c(2:m,2:n,4)./U_c(2:m,2:n,1) - 
(u_c(2:m,2:n).*u_c(2:m,2:n) + v_c(2:m,2:n).*v_c(2:m,2:n))/2;
    T_c(2:m,2:n) = e_c(2:m,2:n)/c_v;
%     p_c(2:m,2:n) = R*rho_c(2:m,2:n).*T_c(2:m,2:n); % not necessary 
decoding

    %-----------------------------------------------------------
    %% Boundary conditions of corrector step
    [rho_c,u_c,v_c,T_c,p_c] =
Condiciones_Contorno_Rampa_ascendente(rho_c,u_c,v_c,T_c,p_c,Nxi,Neta);

    e_c(Nxi,2:Neta-1) = c_v*T_c(Nxi,2:Neta-1);

    %-------------------------------------------------------------------
    %% Encode variables for result and calculation of result

    U_c(:,:,1) = rho_c;
    U_c(:,:,2) = rho_c.*u_c;
    U_c(:,:,3) = rho_c.*v_c;
    U_c(:,:,4) = rho_c.*(e_c + (u_c.*u_c + v_c.*v_c)/2);

    U = (U_p + U_c)/2;

    %-------------------------------------------------------------------
    %% Check for convergence

    if max( max( abs( U(:,:,1) - rho ) ) )>1e-8
        conver = 0;
    else
        conver = 1;
    end

    %-------------------------------------------------------------------
    %% Decode final variables

    rho = U(:,:,1);
    u = U(:,:,2)./U(:,:,1);
    v = U(:,:,3)./U(:,:,1);
    e = U(:,:,4)./U(:,:,1) - (u.*u + v.*v)/2;
    T = e./c_v;
    p = R*rho.*T;
    p(:,1) = 2*p(:,2) - p(:,3);

    [rho,u,v,T,p] = Boundary_Conditions(rho,u,v,T,p,Nxi,Neta);

    nu = nu0*(T/T0).^(3/2).*((T0+110)./(T+110));
    k = nu*c_p/Pr;

    % Update time and iteration counts
    iter = iter + 1;
    t = t + dt;

    % Print at screen to check for correct execution
    display([t,dt])
    if max(max(abs(imag(T))))>0 && flag==0
        tbreak = t;
        flag = 1;
    end

%     mesh(y,x,p)
%     pause(0.001)

end

%-----------------------------------------------------------------------
```

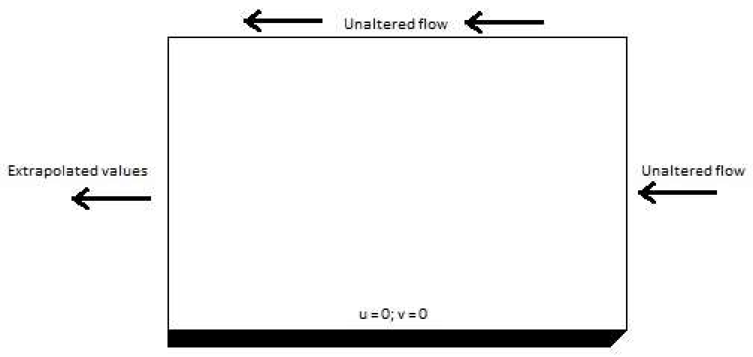
```
%% Check the validity of the results by calculating the mass inflow
and outflow balance

Mass_in = trapz( y(1,:) , u(1,:).*rho(1,:) );
Mass_out = trapz( y(Nxi,:) , u(Nxi,:).*rho(Nxi,:) );
if abs( ( Mass_in - Mass_out ) / Mass_in*100 ) <= 1
    display('The solution seems valid checking the mass balance')
else
    display('The solution seems to violate the mass balance, it is not
valid')
end

%-----------------------------------------------------------------------
%% Viscosity force at the wall

f_pared = nu(:,1).*( u(:,2) - u(:,1) )/dy;
F_pared = trapz(0:dx:LHOR,f_pared);
C_d = F_pared/0.5/rhoinf/uinf^2/LHOR;

F_press = trapz(0:dx:LHOR,p(:,1));

%-----------------------------------------------------------------------
%% Plots and ending

plot( x(:,1) , p(:,1) )
figure(2)
mesh(0:dy:LVER,0:dx:LHOR,real(p))
disp = [( Mass_in - Mass_out ) / Mass_in*100 conver]
time = toc

% Additional plots and calculations using the results may be done at
% different scripts
```

4 Applications

4.1 Proof of Concept: The Flat Plate

Before any further using of the method, it is necessary the testing of the program that has been created in order to check the correct implementation of the technique. This testing has to be done with a problem about which other results already exist, provided by other authors, so they can be compared to the ones in this project.

In this case the best problem to do so is the flat plate. In this problem the computational domain is comprised of one rectangle, bellow which is the flat plate. It composes the lower boundary of the domain. In the upper boundary, the stream of air is supposed to be unaltered, and in the right boundary the air enters the domain also unaltered, as with supersonic flows the flat plate cannot alter the flow upstream.



The dimension of the domain is 10^{-5} m in the horizontal direction. In the vertical direction it is set by calculating the approximated thickness of the boundary layer, estimated by the Reynolds number, and multiplying it by a factor, in this case set to five.

$$Re_{HOR\,\infty} = \frac{\rho_\infty U_\infty L_{HOR}}{\mu_\infty}; \quad \delta = \frac{5\, L_{HOR}}{\sqrt{Re_{HOR\,\infty}}}$$

Equation 4.1

$$L_{VER} = 5 \cdot \delta$$

The boundary conditions that have been implemented are the ones expressed earlier, together with a no slip condition in the flat plate surface and the calculation by extrapolation of the pressure at the plate surface and of all the variables at the left side of the domain. However, this boundary conditions have two important variants.

4.1.1 Constant Temperature Plate

In the first one the temperature of the plate is supposed to remain unaltered. The results for a Mach 2 flow, with 70 x 70 grid points are shown here:

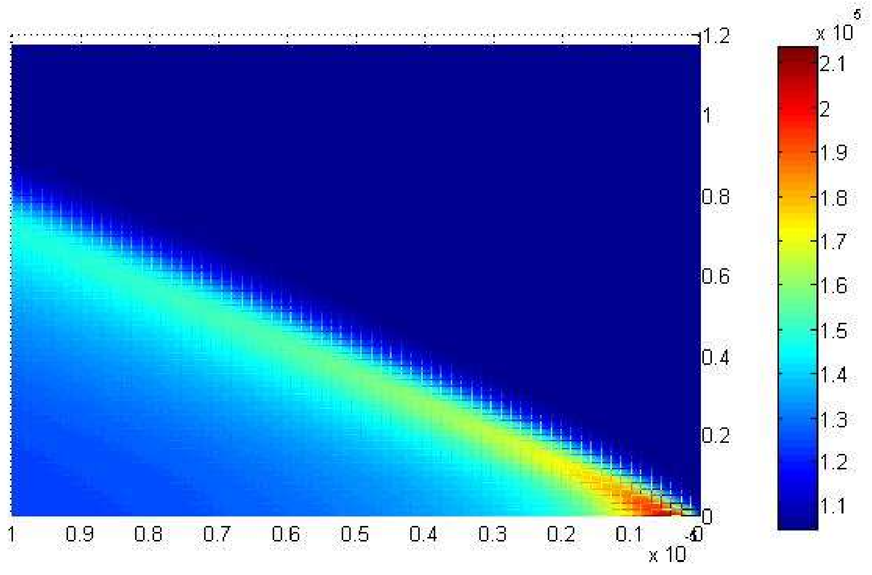


Figure 4.2. Pressure (Pa) in flat plate, Mach 2, grid 70 x 70. Two-dimensional view.

Figure 4.2 shows the pressure in the domain when stationary flow is reached. The hotter colours represent higher values of pressure, and the blue ones represent lower values. It can clearly be seen the presence of a shock wave at the point in which the flow contacts the plate.

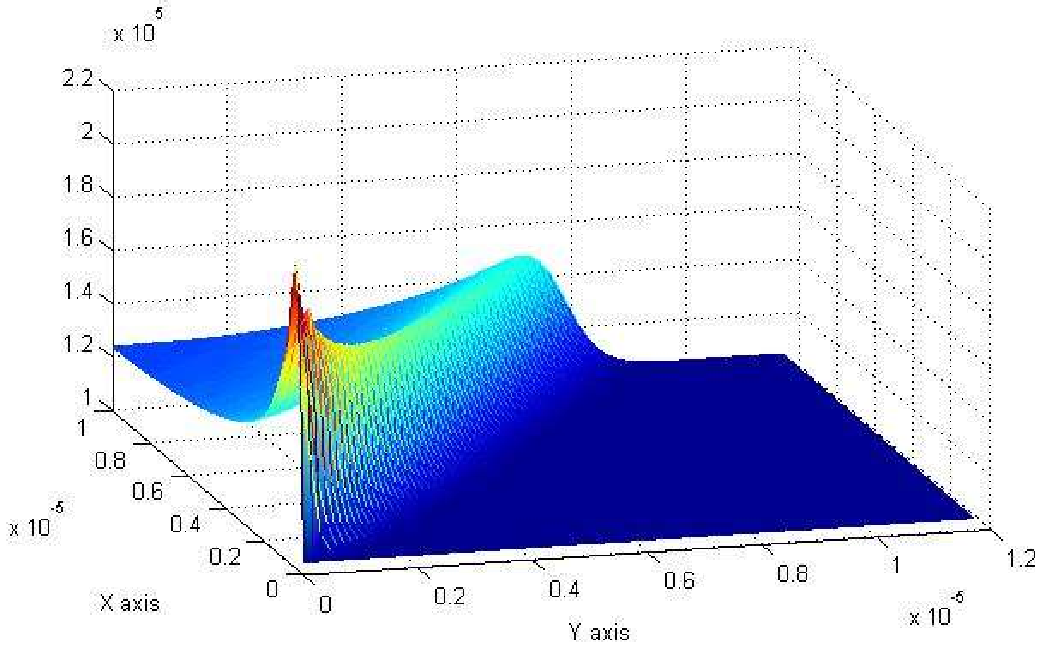


Figure 4.1. Pressure in flat plate, Mach 2, grid 70 x 70. Three-dimensional view.

In this three-dimensional representation the increase of pressure is even more evident. The shock wave is created at the beginning of the plate, the leading-edge, where the pressure reaches two times the atmospheric pressure, and then travels backwards and upwards, with an angle with respect to the horizontal direction. In Figure 4.2 it can be seen that at the end of the horizontal domain the shock wave reaches a height of 0.8×10^{-5}.

The velocity field is obviously different, represented in Figure 4.4. It also suffers the effect of the shock wave, which reduces the velocity of the air. Despite that, the effect of the boundary layer, caused by the no-slip condition, near to the plate is much more important.

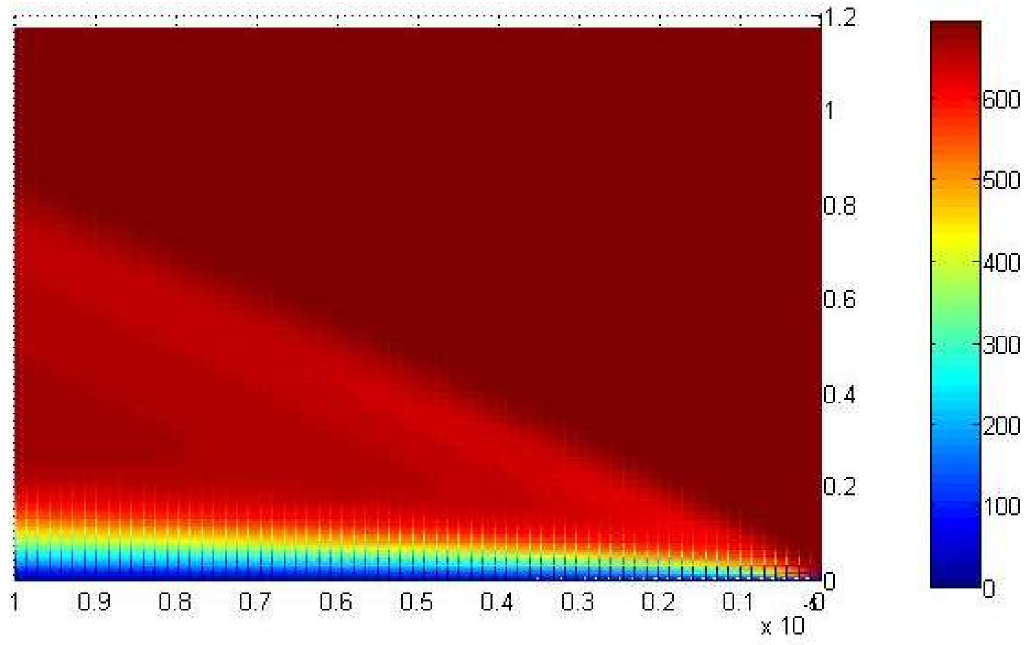


Figure 4.4. Velocity (m/s) in flat plate, Mach 2, grid 70 x 70. Two-dimensional view.

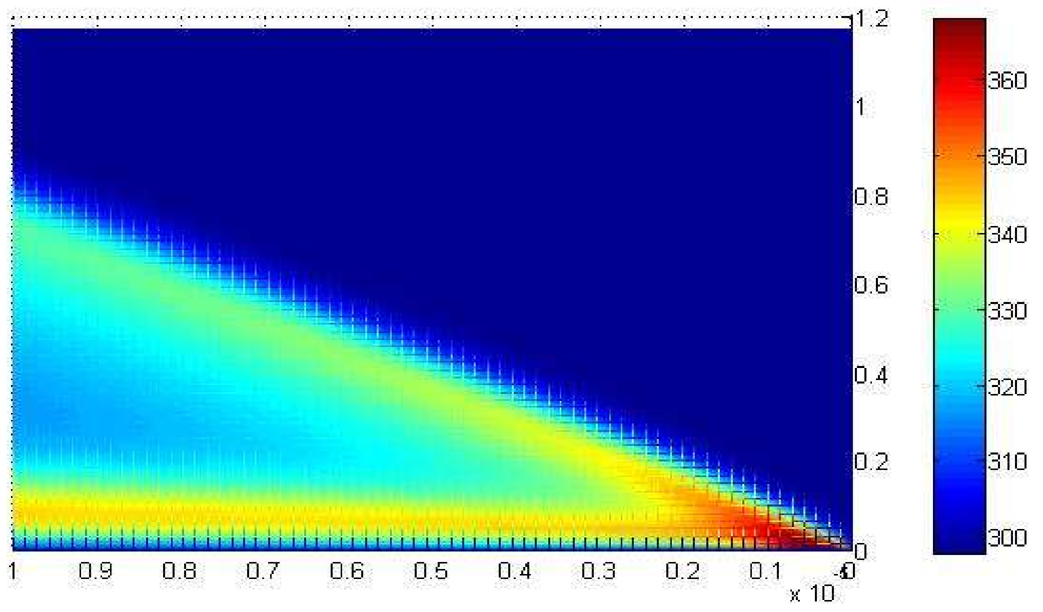


Figure 4.3. Temperature (K) in flat plate, Mach 2, grid 70 x 70. Two-dimensional view.

The temperature field also suffers the effects of the boundary layer. Figure 4.3 indicates that the air is heated a lot at the leading-edge point. Then, there are two differentiated regions of hot air. The first is the air heated by the shock wave. The second is the boundary layer region, due to fiction and possibly because of convection from the hotter leading-edge, but this latter factor is just a theory. In the middle the temperature of the air is greater than in the exterior but lower than in these two regions.

4.1.1.1 Comparison with previous results

As said previously, one of the aims when simulating the flat plate is to compare it with previous results to check the validity of the program. In John D. Anderson's book "Computational Fluid Dynamics" [Ref 1], in chapter 10, this very same problem is solved, and a few results and figures are offered so they can be compared with the results of this project.

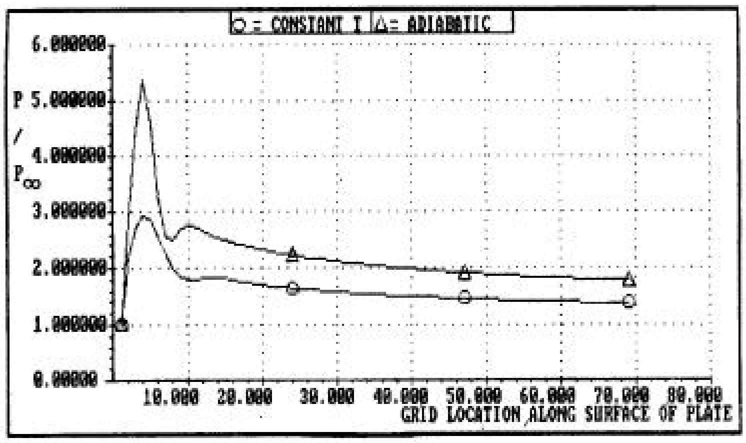


Figure 4.5. John D. Anderson's "Computational Fluid Dynamics" [Ref 1] results. Pressure along surface of plate with grid 70x70 and Mach 4.

This figure represents the surface pressure that he obtained in the plate with a Mach number of 4 and a grid of 70x70 points. If these conditions are replicated the results are represented in the following figure.

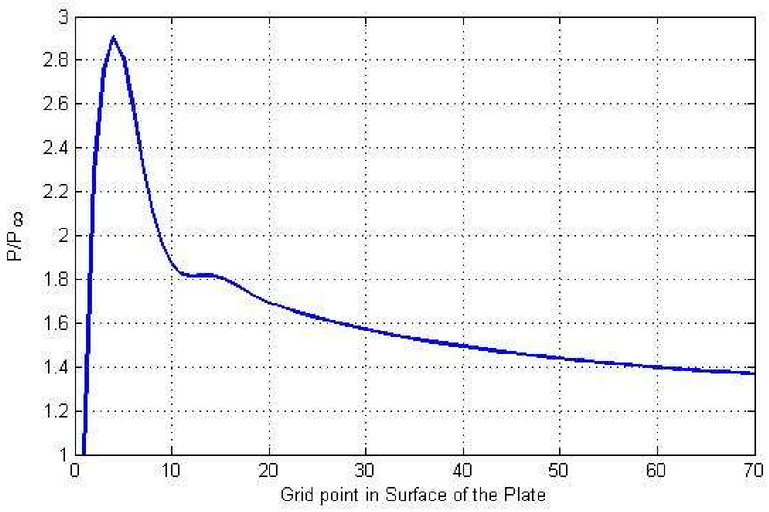


Figure 4.6. Surface pressure, Mach 4, Grid 70x70.

Looking at the constant temperature results obtained by John Anderson, the curve below in Figure 4.5, and the Figure 4.6, it can be seen that they are identical. The pressure spike peaks in around 2.9 times the outside pressure in both figures, then between the points 10 and 15 it appears a region in which pressure stabilizes and even increases slightly, and at the end of the plate the pressure is almost 1.4 times the outside pressure.

To ensure that the results of the program developed are valid other figures are going to be compared too. The next is the pressure profile in the vertical direction at the trailing edge.

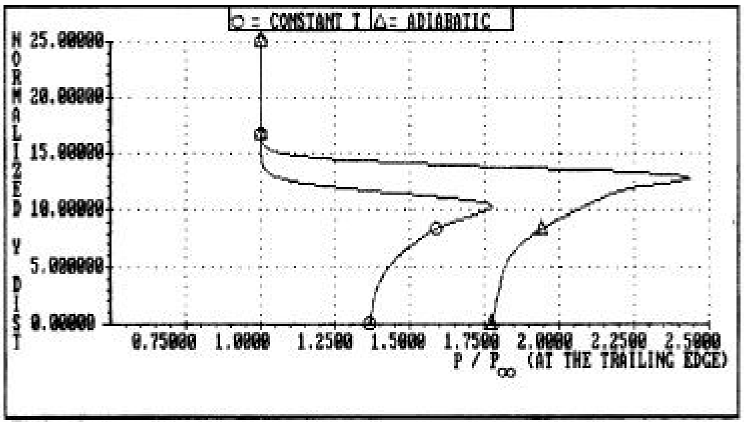


Figure 4.8. John D. Anderson's "Computational Fluid Dynamics" [Ref 1] results. Pressure profile at trailing edge with grid 70x70 and Mach 4.

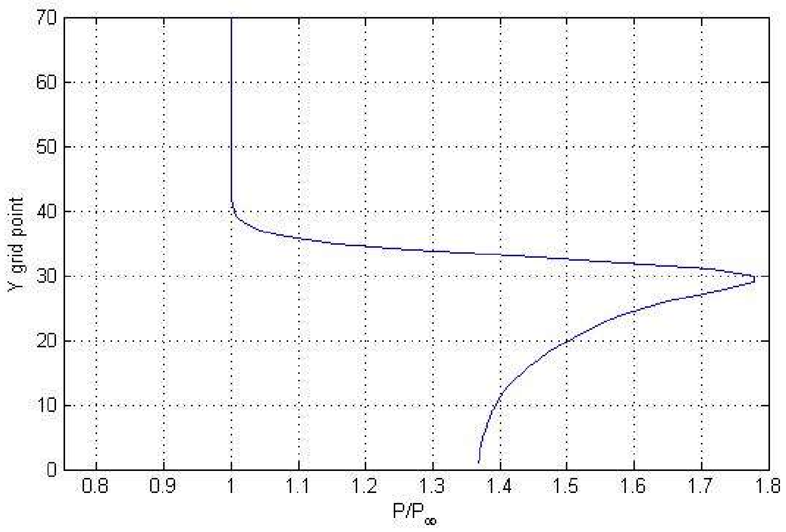


Figure 4.7. Trailing edge pressure profile, Mach 4, Grid 70x70.

John Anderson's result is shown in Figure 4.8 and the obtained with the program in Figure 4.7, and they are again identical. The pressure starts rising at 60 % of the total height, peaks at around 40 % with a value of almost 1.8 and ends at a value of around 1.37.

This same trailing edge profile analysis can be made with the temperature. The results are also

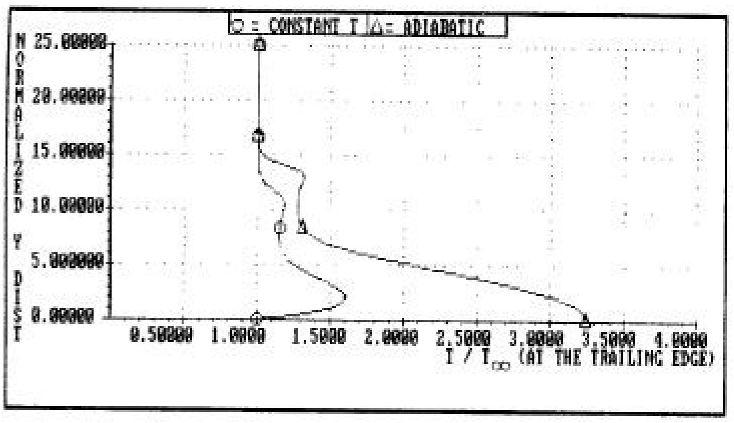


Figure 4.9. John D. Anderson's "Computational Fluid Dynamics" [Ref 1] results. Temperature profile at trailing edge with grid 70x70 and Mach 4.

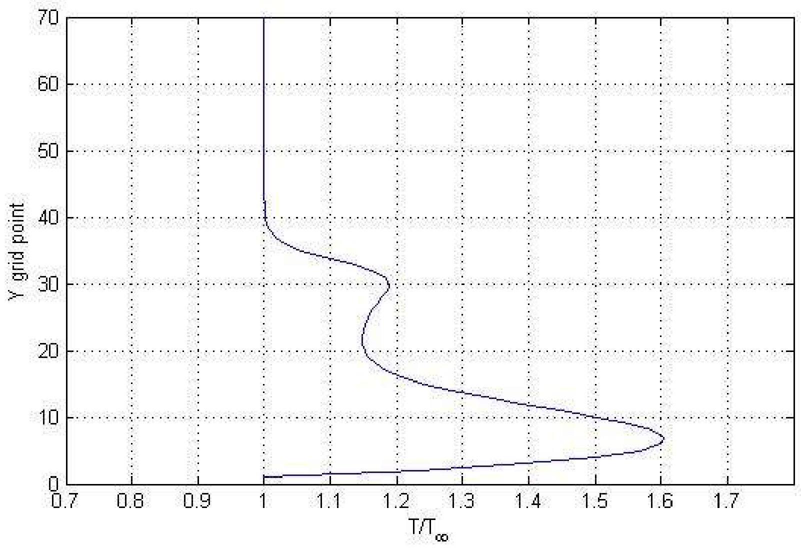


Figure 4.10. Trailing edge temperature profile, Mach 4, Grid 70x70.

the same, shown in figures Figure 4.10 and Figure 4.9. As it was said for Figure 4.3, there are two regions of heated air, represented here by the two spikes in temperature in the profile, one close to the plate and other at around 40 % of the height. The first spike reaches around 1.2 times the outside temperature and the inner spike reaches 1.6 times it.

The last comparison is going to be the velocity profile. This is going to be made by the trailing edge Mach number profiles, shown in figures Figure 4.12 and Figure 4.11. From the (0,0) point, which represents the no slip condition, the air accelerates to the exterior velocity of Mach 4. The presence of the shock wave is also noticeable at around 40 % of the profile, provoking an initial drop in velocity.

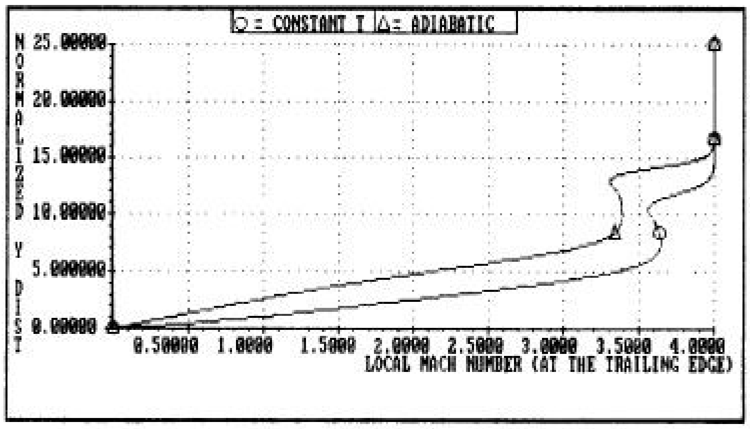


Figure 4.12. John D. Anderson's "Computational Fluid Dynamics" [Ref 1] results. Mach number profile at trailing edge with grid 70x70 and Mach 4.

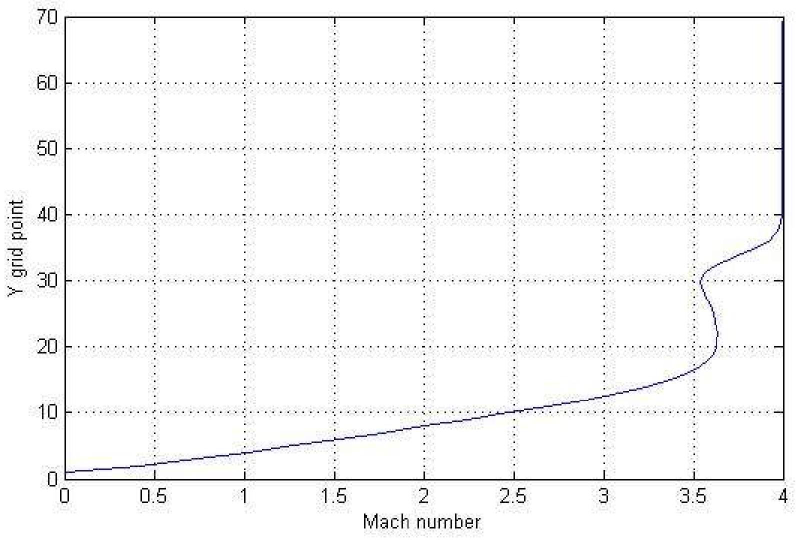


Figure 4.11. Trailing edge temperature profile, Mach 4, Grid 70x70.

As a final conclusion it can be said that the code programmed obtains valid results and is ready to analyse more complex situations.

4.1.1.2 Dependency with the grid points

One of the tests that can be made now is for example how the results vary with the fineness of the grid. So, several simulations where made of the flat plate with Mach 2, only varying the number of grid points, for the results to be compared.

What has been found is that at the trailing edge the results are almost the same. For instance, the pressure profiles are shown in Figure 4.14. Simply the shock wave moves up slightly.

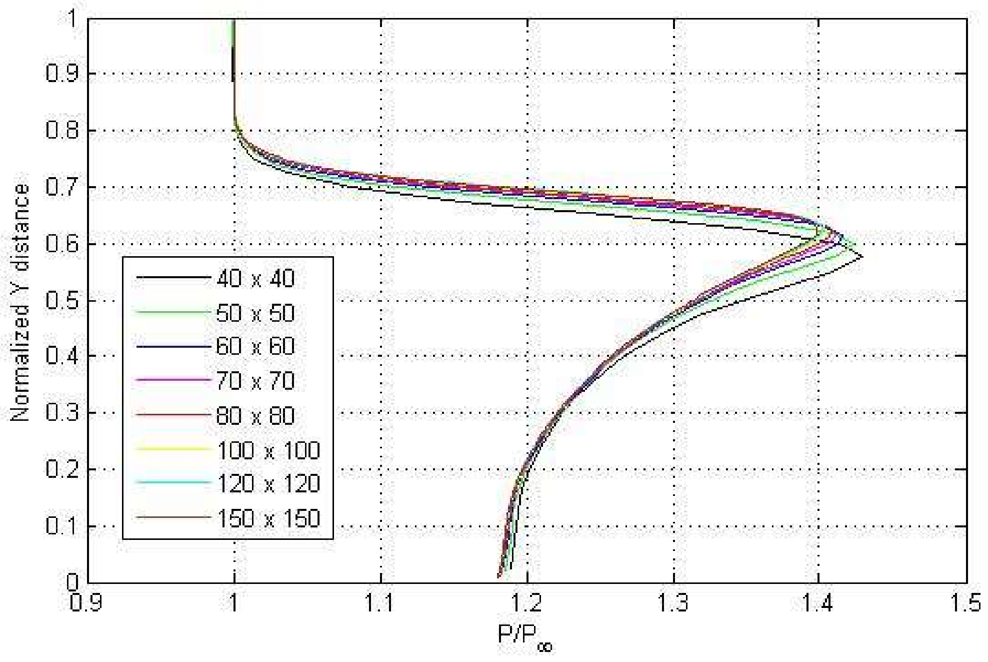


Figure 4.14. Trailing edge pressure profiles, Mach 2, dependency with the grid points.

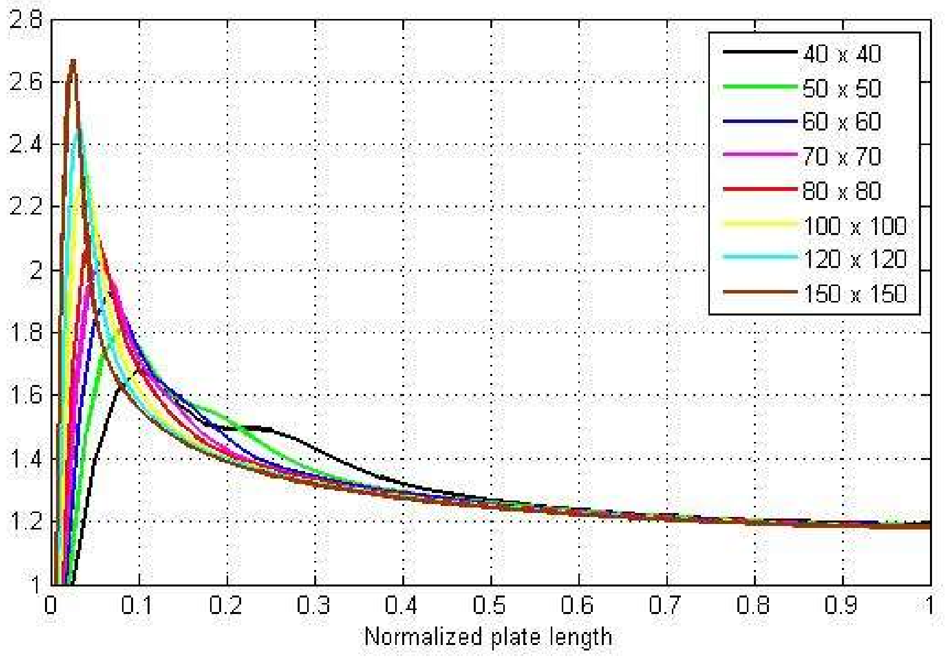


Figure 4.13. Surface pressure, Mach 2, dependency with the grid points.

But at the leading edge the results change significantly. The pressure spike becomes more intense, starts earlier and is thinner with the increase in number points. The resistance coefficient rises a little bit, and the total vertical force, in Newtons, is almost unchanged.

	40 x 40	50 x 50	60 x 60	70 x 70	80 x 80	100x100	120x120	150x150
C_d	0.0569	0.0589	0.0604	0.0616	0.0626	0.0643	0.0655	0.0670
Vertical force (N)	-1.383	-1.386	-1.388	-1.389	-1.389	-1.390	-1.390	-1.389

4.1.1.3 Dependency with the Mach number

It is also interesting to make the analysis of how the Mach number affects the results. For instance, looking at Figure 4.15 and comparing it with Figure 4.1, it is clear that the shock wave has been made narrower. Also, the pressure spike is much greater, and other calculations like the forces change too.

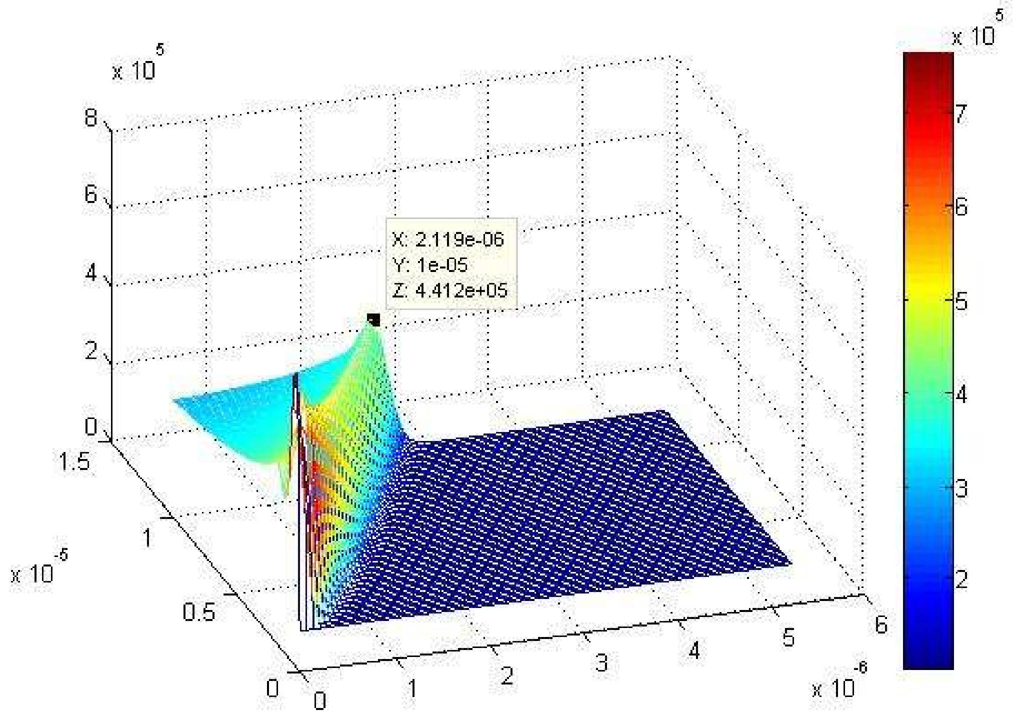


Figure 4.15. Pressure (Pa) field, Grid 70 x 70 points, Mach number 10.

The maximum value of the pressure in the shock wave at the trailing edge rises with the Mach number according to the Figure 4.18. Also, as mentioned before, the point of this maximum gets closer to the flat plate, in what seems to be an asymptotic behaviour according to Figure 4.17.

The pressure at the plate is considerably altered by the Mach number, shown in Figure 4.16.

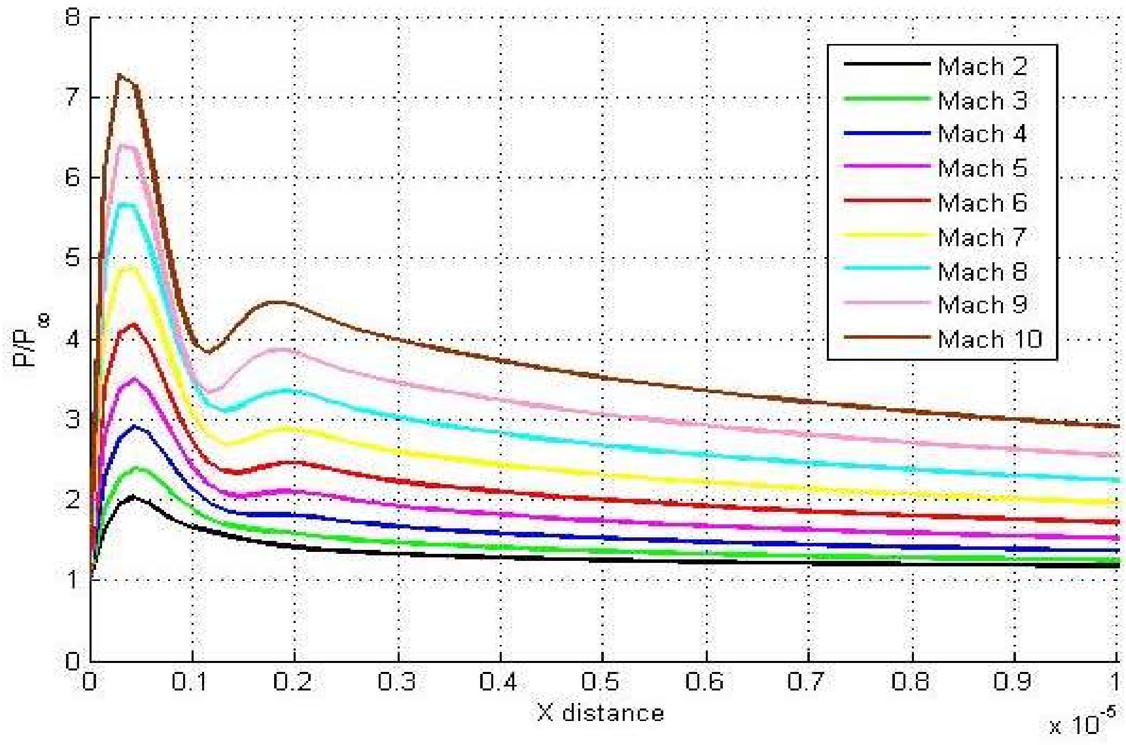


Figure 4.16. Pressure at the surface of the plate

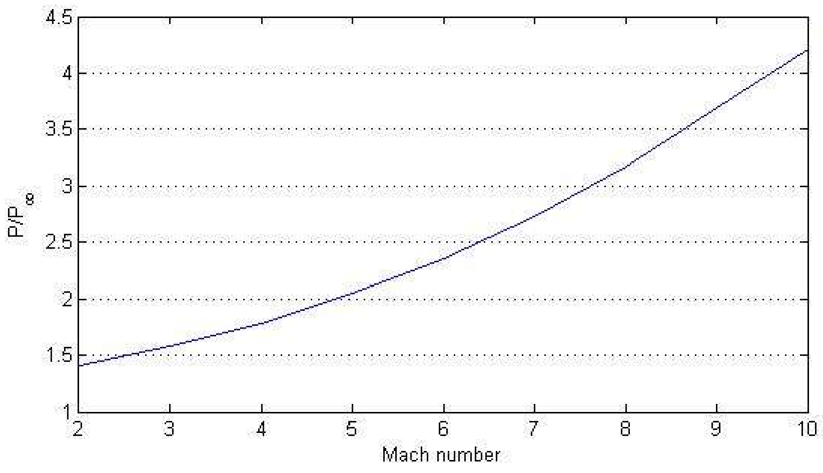


Figure 4.18. Trailing edge maximum shock wave pressure value.

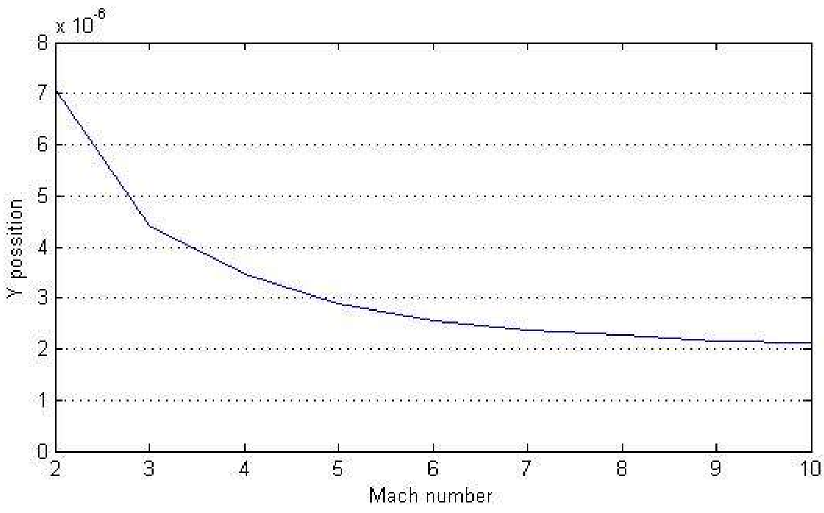


Figure 4.17. Trailing edge maximum shock wave pressure position.

The resistance coefficient decreases, even though the actual friction force increases, due to the increase in the external velocity that is used in the normalization.

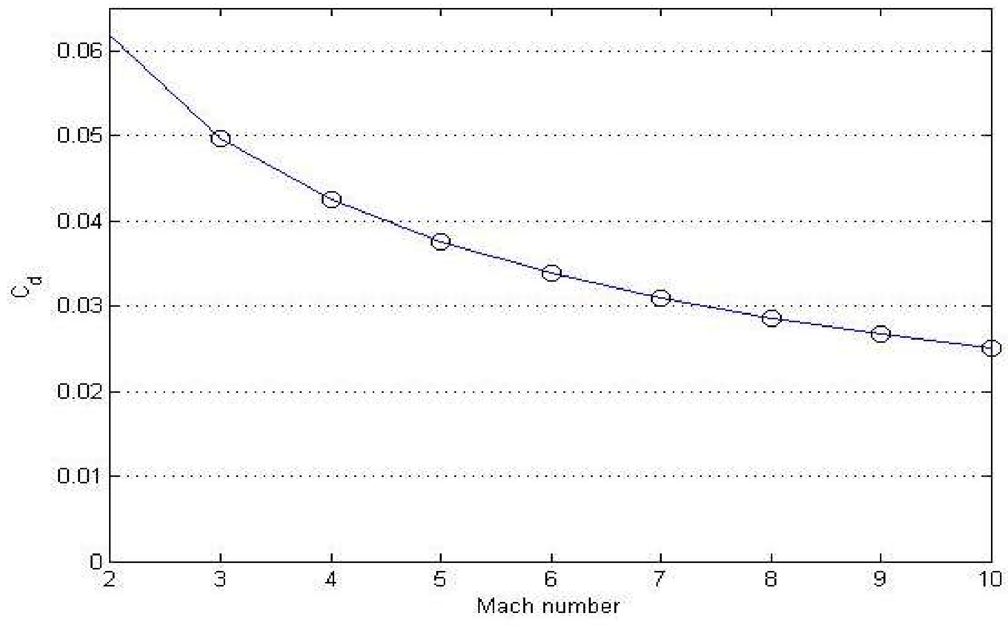


Figure 4.20. Resistance coefficient.

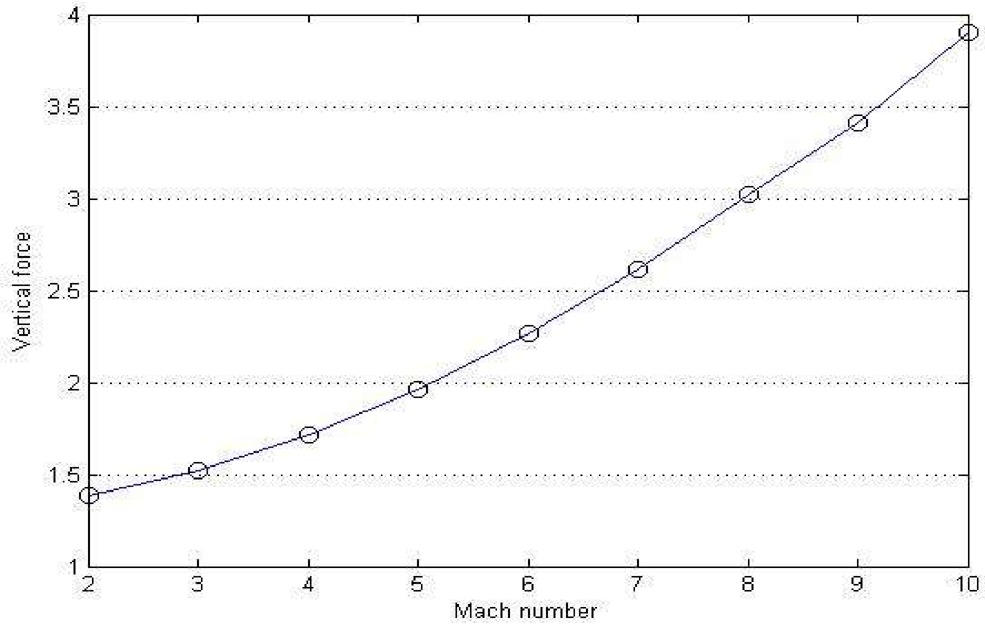


Figure 4.19. Downward vertical force (N).

The downward vertical force increases due to the increase in the overall pressure on the plate surface.

4.1.2 Adiabatic Plate

The second variant of the boundary conditions is the case of the adiabatic plate. This time, the temperature of the plate is not constant, instead there is no heat transfer crossing through it. The results obtained by John Anderson for the adiabatic plate have already been shown in figures Figure 4.5, Figure 4.8, Figure 4.9 and Figure 4.12.

For this section not to be too large only the case of Mach number 4 and 70 x 70 grid points is going to be discussed to compare it with the results obtained in [Ref 1].

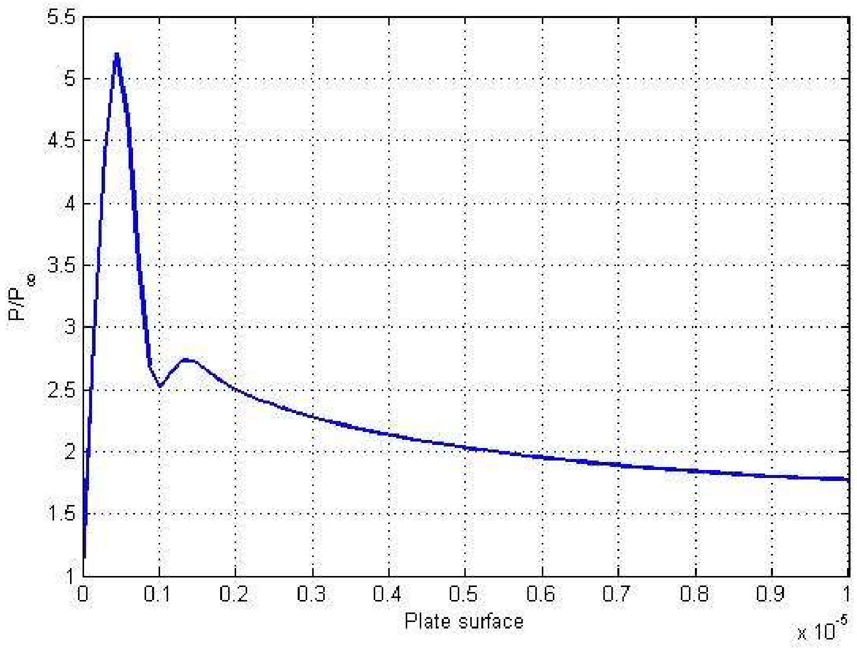


Figure 4.21. Surface pressure for adiabatic plate, Mach 4, Grid 70 x 70 points.

To begin with, the surface pressure obtained by John Anderson in Figure 4.5 and the one obtained in this project are identical. The pressure peak is slightly higher than five time the outside value and the behaviour, with an increase in pressure around the grid point 10 and a value of about 2.8 times the outside pressure.

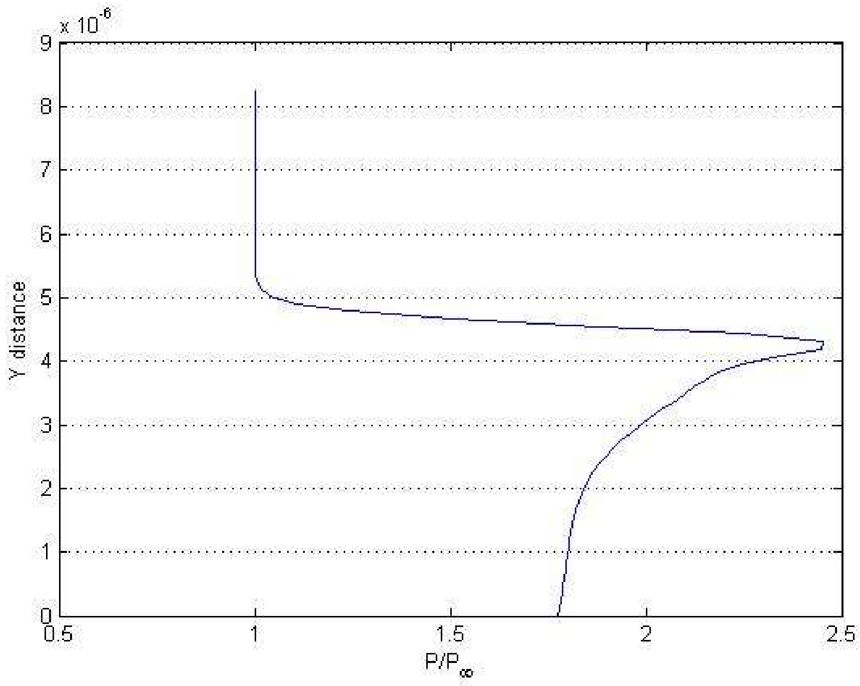


Figure 4.22. Trailing edge pressure profile for adiabatic plate, Mach 4, Grid 70 x 70.

The trailing edge pressure profile is also the same in Figure 4.22 than in Figure 4.8, the temperature profile is the same in Figure 4.23 than in Figure 4.9, and finally the Mach number profile is the same in Figure 4.24 than in Figure 4.12.

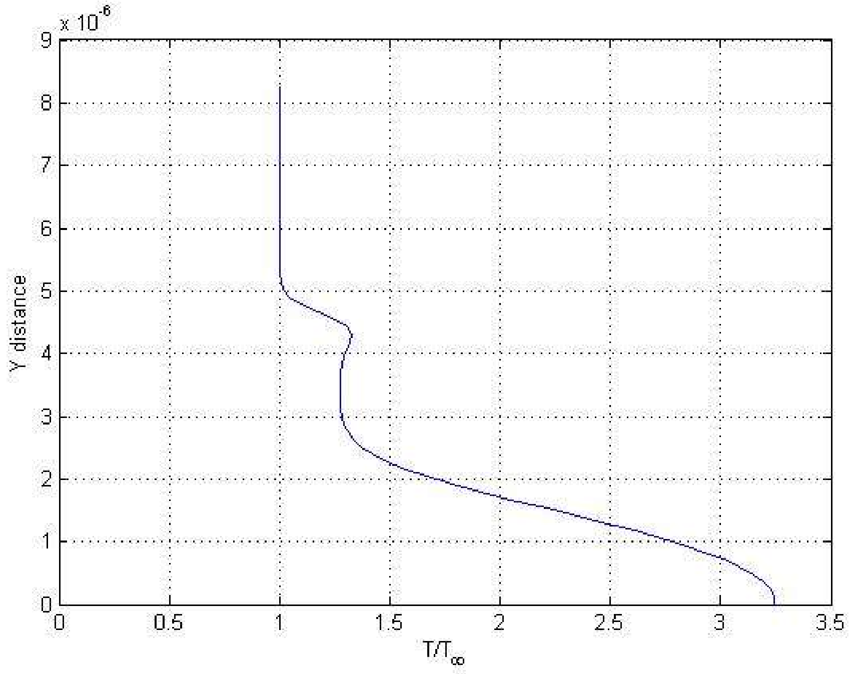


Figure 4.23. Trailing edge temperature profile for adiabatic plate, Mach 4, Grid 70 x 70.

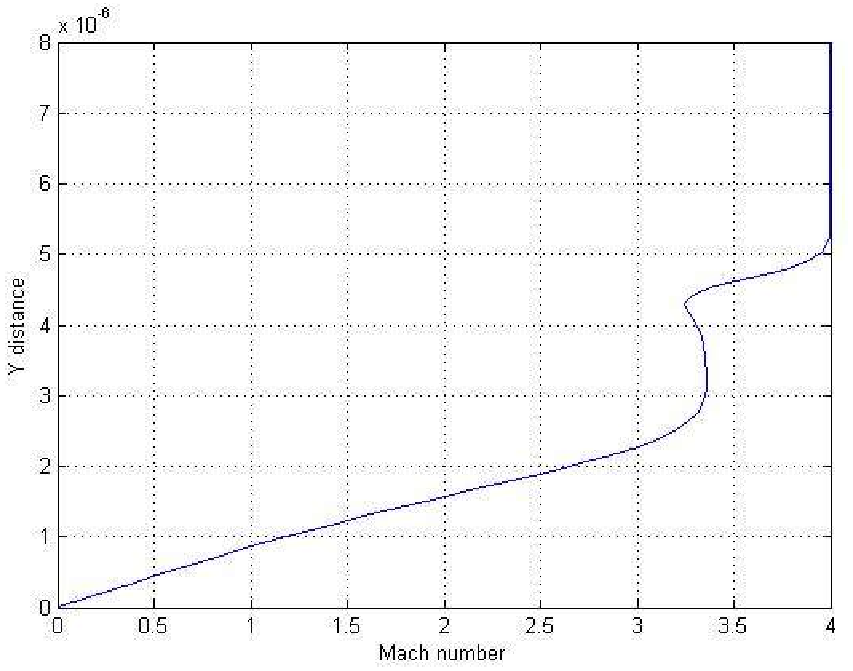


Figure 4.24. Trailing edge Mach number profile for adiabatic plate, Mach 4, Grid 70 x 70.

To close the adiabatic plate case the temperature field is displayed in Figure 4.25. The air is firstly heated by the shock wave and then heats more and more when approaching the surface.

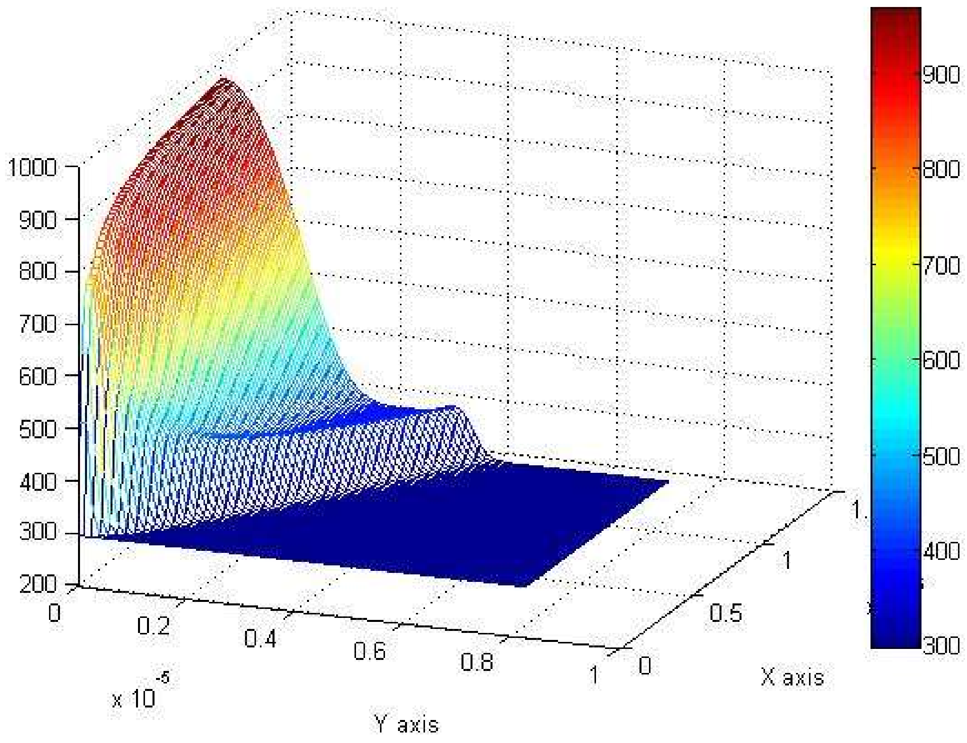


Figure 4.25. Temperature (K) field for adiabatic plate.

4.1.3 Flat plate with compressed grid

Not only the program used for rectangular and uniform grids needs testing, also the used for non-uniform grids does. To achieve this what has been done is solving the flat plate but now using a compressed grid. In particular, the grid transformation is given by the following relations:

$$x = \frac{e^{\xi} - 1}{f}$$

Equation 4.2

$$y = \frac{e^{\eta} - 1}{f}$$

Equation 4.3

Theses relations create a compressed grid around the point (0,0), where f is a corrector factor that determines the severity of this compression. Obtaining the analytical direct and inverse metrics from Equation 4.2 and Equation 4.3 is easily achieved:

$$\frac{\partial x}{\partial \xi} = \frac{e^{\xi}}{f}\ ; \frac{\partial y}{\partial \eta} = \frac{e^{\eta}}{f}\ ; \frac{\partial \xi}{\partial x} = \frac{f}{fx+1}\ ; \frac{\partial \eta}{\partial y} = \frac{f}{fy+1}$$

With the rest been equal to zero. As a demonstration, the Figure 4.26 shows the grid generated with this method for a factor of 10^6 and 15 x 15 points.

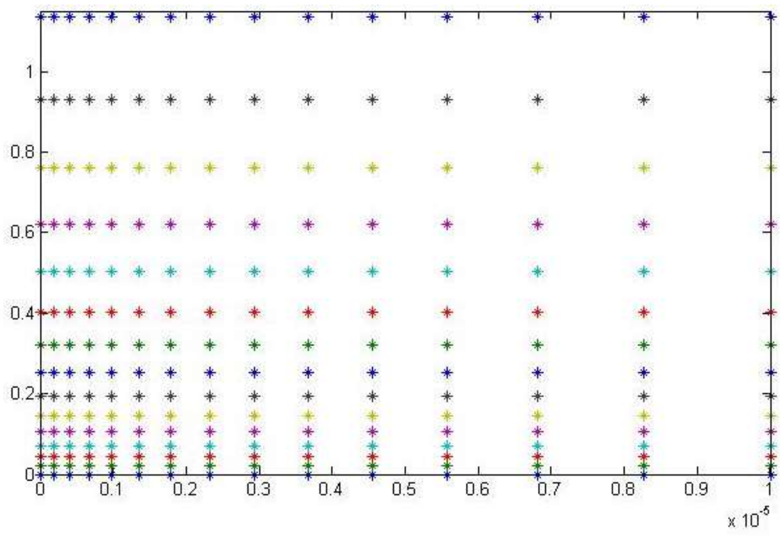


Figure 4.26. Compressed grid of 15x15 points and f = 10^6

The only thing that is left to say is that with this grid the results obtained by the "MainGridTrans_DirectMetrics.m" program have been exactly the same as the obtained by the uniform grid, so they are not going to be repeated here. By checking that the results coincide, the correct functioning of the program developed for non-uniform grid transformations has been ensured.

4.1.4 Auxiliary functions for the flat plate

Initial Conditions Function (both for uniform and non-uniform grid)

```
function [rho,u,v,T,p] =
Initial_Conditions_FlatPlate(rhoinf,uinf,Tinf,R,Nxi,Neta)

rho(1:Nxi,1:Neta) = rhoinf;
u(1:Nxi,1) = 0;
u(1:Nxi,2:Neta) = uinf;
v(1:Nxi,1:Neta) = 0;
T(1:Nxi,1:Neta) = Tinf;
p(1:Nxi,1:Neta) = R*rho.*T;
```

Boundary Conditions Function (both for uniform and non-uniform grid)

```
function [rho,u,v,T,p] =
Boundary_Conditions_FlatPlate(rho,u,v,T,p,Nxi,Neta)

% CC at the right side
rho(Nxi,2:Neta-1) = 2*rho(Nxi-1,2:Neta-1) - rho(Nxi-2,2:Neta-1);
u(Nxi,2:Neta-1) = 2*u(Nxi-1,2:Neta-1) - u(Nxi-2,2:Neta-1);
v(Nxi,2:Neta-1) = 2*v(Nxi-1,2:Neta-1) - v(Nxi-2,2:Neta-1);
T(Nxi,2:Neta-1) = 2*T(Nxi-1,2:Neta-1) - T(Nxi-2,2:Neta-1);
p(Nxi,2:Neta-1) = 2*p(Nxi-1,2:Neta-1) - p(Nxi-2,2:Neta-1);

% CC at the plate surface
```

```
rho(2:Nxi,1) = 2*rho(2:Nxi,2) - rho(2:Nxi,3);
p(2:Nxi,1) = 2*p(2:Nxi,2) - p(2:Nxi,3);
T(2:Nxi,1) = T(2:Nxi,2);
```

Geometry Function (only for compressed grid)

```
function
[x,y,dxi,deta,dxidx,dxidy,detadx,detady,J,dxdxi,dxdeta,dydxi,dydeta] =
Geometry_CompressedGrid(Nxi,Neta,LHOR,LVER)

f = 10^6;

xi_LHOR = log(f*LHOR+1);
eta_LVER = log(f*LVER+1);

dxi = xi_LHOR/(Nxi-1);
deta = eta_LVER/(Neta-1);

for j=1:Neta
    xi(:,j) = 0:dxi:xi_LHOR;
end
for i=1:Nxi
    eta(i,:) = 0:deta:eta_LVER;
end

for i=1:Nxi
    for j=1:Neta
        x(i,j) = (exp(xi(i,j)) - 1)/f;
        y(i,j) = (exp(eta(i,j)) - 1)/f;
    end
end

dxidx = f./(f*x+1);
dxidy = 0;
detadx = 0;
detady = f./(f*y+1);

dxdxi = exp(xi)/f;
dxdeta = 0;
dydxi = 0;
dydeta = exp(eta)/f;

J = dxdxi.*dydeta - dydxi.*dxdeta;
```

4.2 Forward Step

One of the interesting modifications that can be done to the flat plate is the case in which it exists an adiabatic step facing the direction of the flow. In this case the air creates a strong pressure increase in contact with the vertical wall and pushes back the solid surface. The geometry includes a flat plate before the step that, as it is going to be seen, creates its own shock wave before reaching the wall. The computational domain has been delimited to 10^{-5} in the horizontal direction and to $1.6 \cdot 10^{-5}$ in the vertical direction.

Inside the "solid step" there are grid points, as this problem has been solved with the code that requires a uniform and rectangular grid. What has been done is adequately modifying the initial and boundary conditions to enforce the variables inside the solid square in each time step, thus isolating it from the rest of the flow field. In the walls of the step a no slip condition has been coded so no flow permeates the wall.

The position and height of the wall can be modified at will and, in this case, it always starts at 70% of the horizontal distance, what means $0.7 \cdot 10^{-5}$. The height is going to be variated to see the effect it has over the results. See for instance the figure below, in which the pressure is

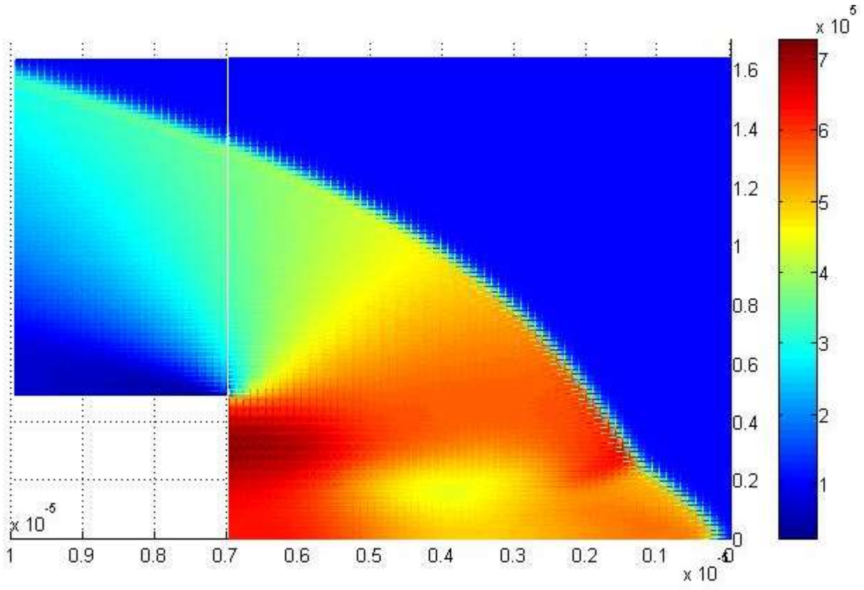


Figure 4.27. Pressure (Pa) in forward step, Mach 2, step (3e-6 x 4.8e-6). Two-dimensional view.

represented for a Mach 2 flow, with a step height of 30% of the vertical domain, $4.8 \cdot 10^{-6}$.

In this example the flat plate creates its own shock wave, but the relatively slow velocity of the flow makes the effect of the step almost erase it. In the next example, the same geometry but at Mach 3, showed in three-dimensional view to better appreciate the figure, the effects of the step are much more compressed close to it.

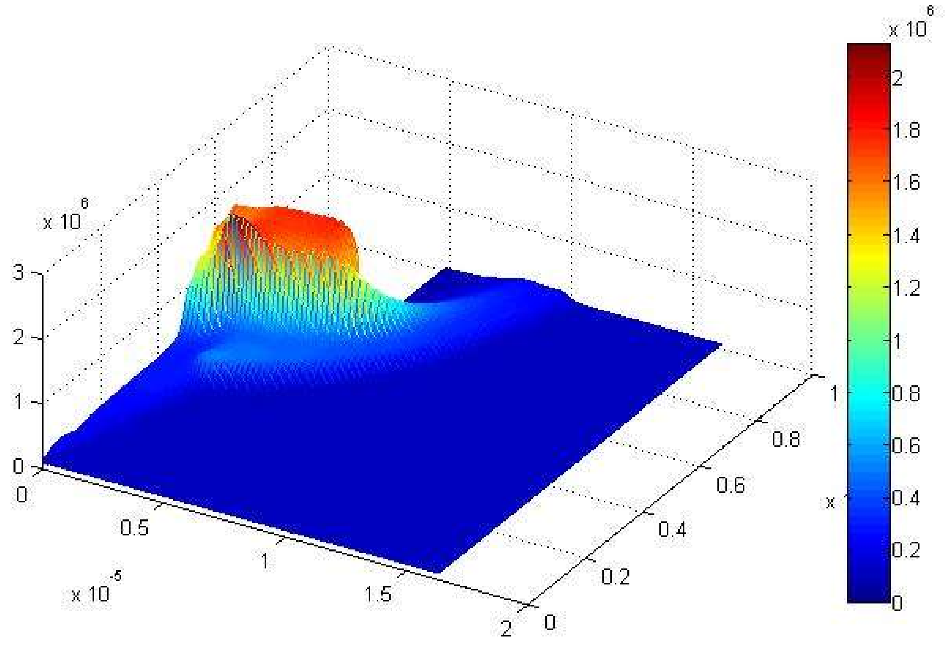


Figure 4.28. Pressure (Pa) in forward step, Mach 3, step (3e-6 x 4.8e-6). Three-dimensional view.

Now looking at the colour bar next to the figures Figure 4.27 and Figure 4.28 notice the great difference in pressure that the different Mach numbers create over the vertical wall. In Figure 4.28, at the beginning of the flat plate it is appreciated the pressure increase equivalent to Figure 4.16 at Mach 3, but still it is very small in comparison with the pressure near the wall.

It also starts to be seen certain interaction between the flat plate and the vertical wall. It starts next to the plate, like if the flow is deflected away from the inner corner by an oblique shock wave. This effect is much better seen by increasing the step height even more, to 50% ($8 \cdot 10^{-6}$), shown in Figure 4.30 for Mach 2.

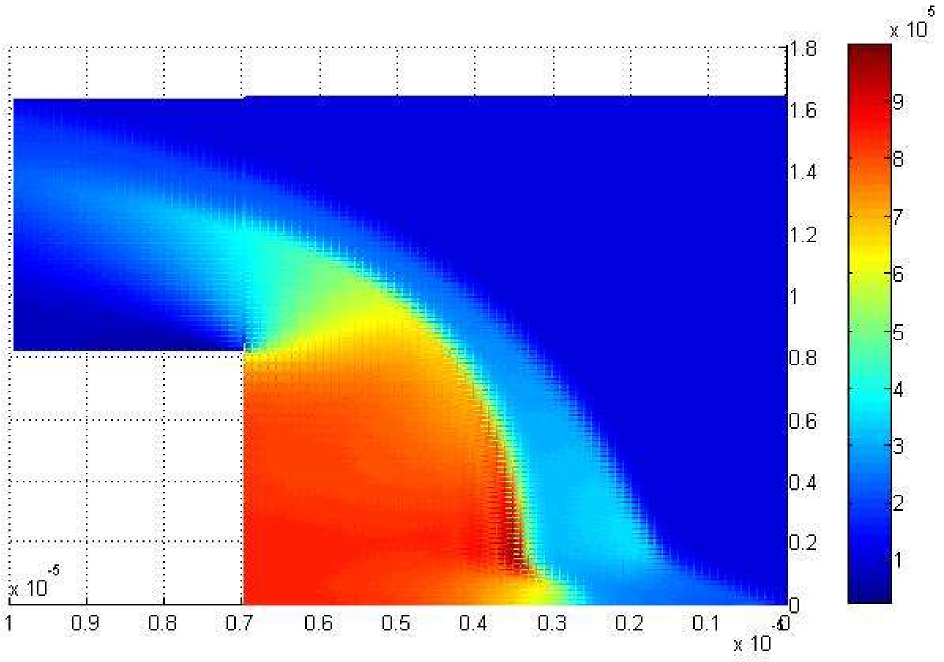


Figure 4.30. Pressure in forward step, Mach 2, step (3e-6 x 8e-6). Two-dimensional view.

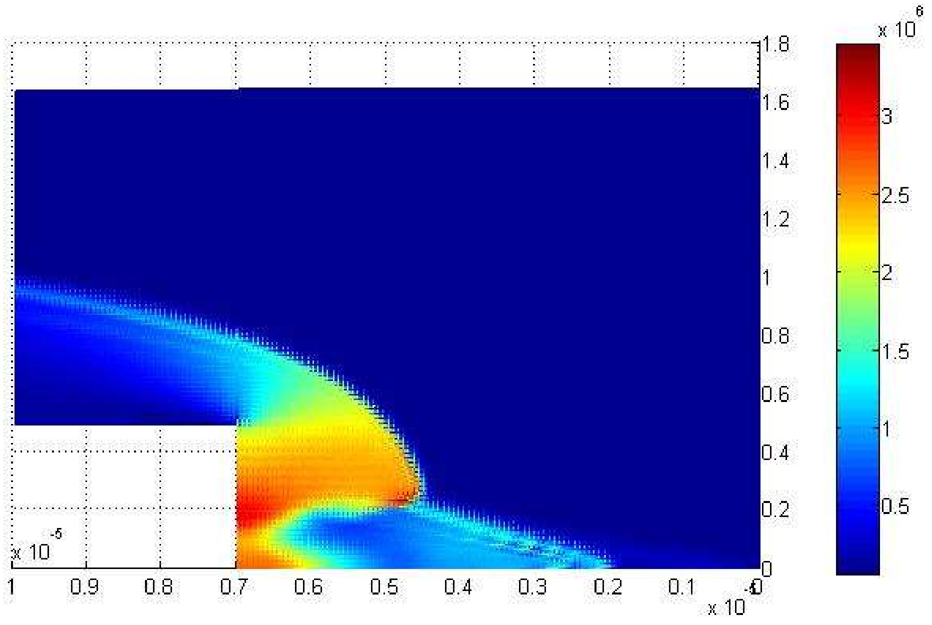


Figure 4.29. Pressure in forward step, Mach 4, step (3e-6 x 4.8e-6). Two-dimensional view.

When the Mach number is increased, this oblique wave mixes with the step effects. Instead what is strongly seen is a recirculating air region that is created near the corner of the step, shown in Figure 4.29.

The mean pressure in the vertical wall for the steps of $4.8 \cdot 10^{-6}$ and $8 \cdot 10^{-6}$ is shown in Figure 4.33. The mean pressure is almost unaffected by the step height, but greatly increases when increasing the Mach number.

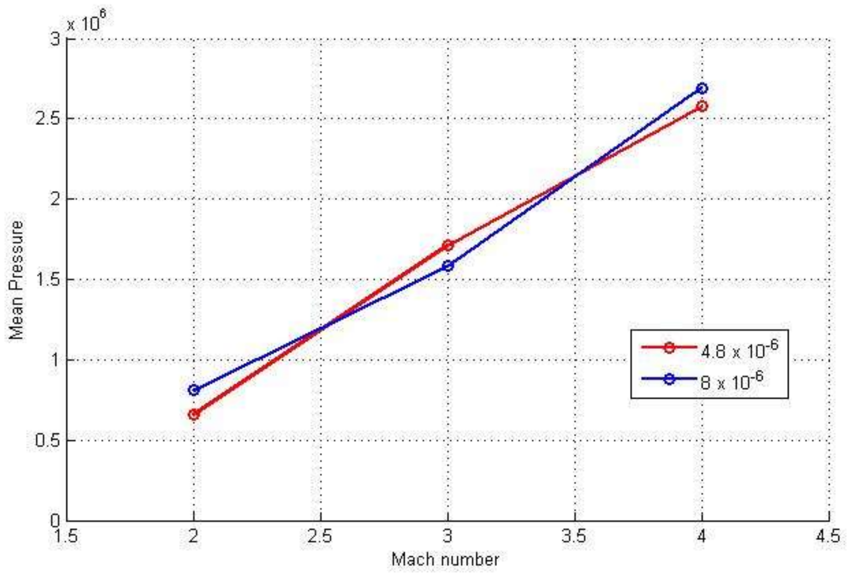


Figure 4.33. Mean pressure for forward step. Dependency with step height and Mach number.

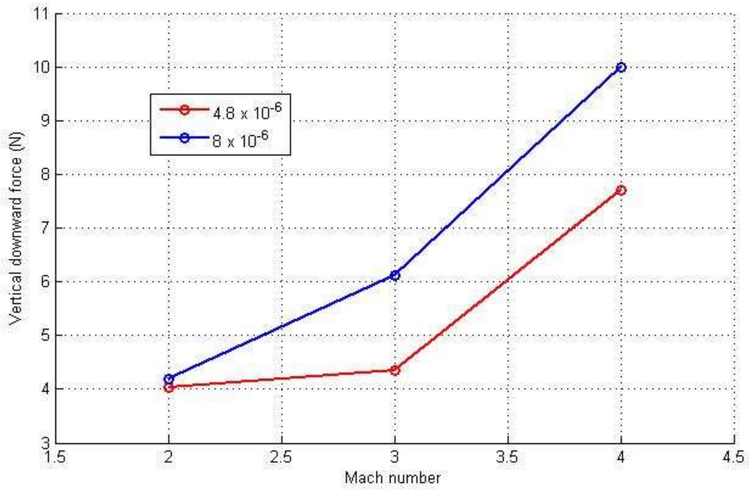


Figure 4.31. Downward vertical force for forward step. Dependency with step height and Mach number.

By comparing Figure 4.19 to Figure 4.31 it is safe to say that the downward vertical force increases due to the presence of the step, which makes sense looking at the high pressure

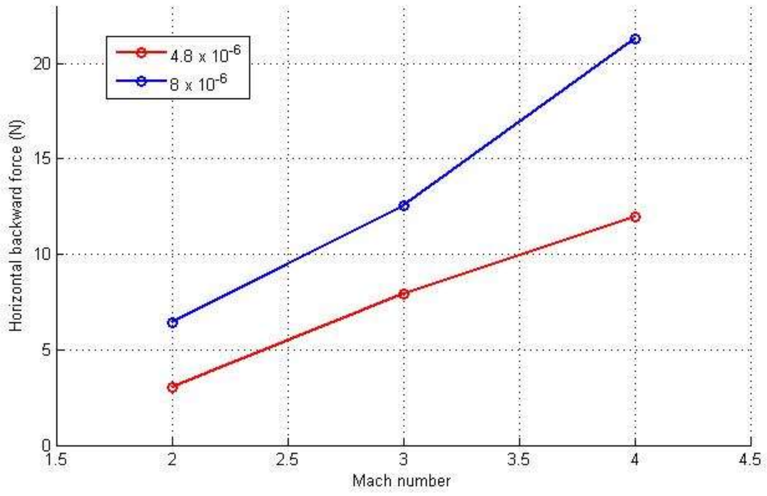


Figure 4.32. Backward horizontal force for forward step. Dependency with step height and Mach number.

region created that is pushing the surface not only backwards but also downward.

When analysing the backward force, the Figure 4.32 shows that it increases with the Mach number, and obviously with the step height, as there is more surface opposing the incoming flow. For comparison, the friction force for the flat plate at Mach 2 was 0.18 Newtons, so friction forces are neglectable compared with the pressure ones.

One last remark is to mention that the temperature field in this case is very similar to the pressure field. For instance, the temperature field for Mach 4 flow with $4.8 \cdot 10^{-6}$ step height is shown in Figure 4.34. It is to be compared with the pressure of Figure 4.29, and the similarities are evident. The only clear difference is the regions very close to the surfaces, in which the temperature falls to accommodate with the constant temperature surfaces set in the boundary conditions.

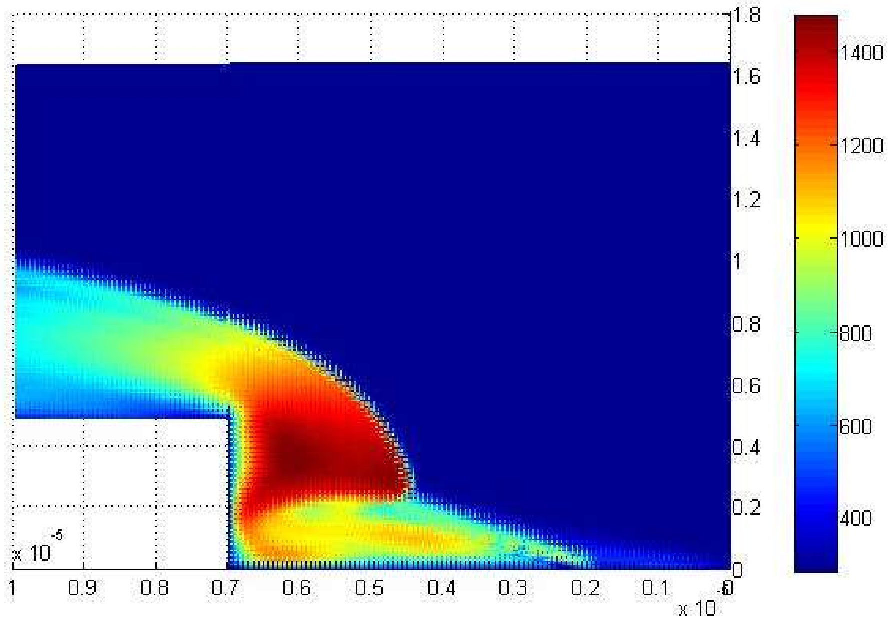


Figure 4.34. Temperature (K) in forward step, Mach 4, step (3e-6 x 4.8e-6). Two-dimensional view.

4.2.1 Auxiliary functions for the forward step

Initial Conditions Function

```
function [rho,u,v,T,p] =
Initial_Conditions_ForwardStep(rhoinf,uinf,Tinf,R,Nx,Ny)

% Step X direction beginning
Nxs = floor(0.7*Nx);

% Step height
Nya = floor(0.5*Ny);

rho(1:Nx,1:Ny) = rhoinf;
u(1:Nx,2:Ny) = uinf;
v(1:Nx,1:Ny) = 0;
T(1:Nx,1:Ny) = Tinf;
p(1:Nx,1:Ny) = R*rho.*T;
```

```
u(1:Nx,1) = 0;

u(Nxs:Nx,1:Nya) = 0;
v(Nxs:Nx,1:Nya) = 0;
T(Nxs:Nx,1:Nya) = Tinf;
```

Boundary Conditions Function

```
function [rho,u,v,T,p] =
Boundary_Conditions_ForwardStep(rho,u,v,T,p,Nx,Ny)

% Step X direction beginning
Nxs = floor(0.7*Nx);

% Step height
Nya = floor(0.5*Ny);

% Force variables at the solid square
u(Nxs:Nx,1:Nya) = 0;
v(Nxs:Nx,1:Nya) = 0;

T(Nxs:Nx,1:Nya) = 298;
p(Nxs+1:Nx,1:Nya-1) = 1.0477e+05;

% CC at the right side
rho(Nx,1:Ny-1) = 2*rho(Nx-1,1:Ny-1) - rho(Nx-2,1:Ny-1);
u(Nx,1:Ny-1) = 2*u(Nx-1,1:Ny-1) - u(Nx-2,1:Ny-1);
v(Nx,1:Ny-1) = 2*v(Nx-1,1:Ny-1) - v(Nx-2,1:Ny-1);
T(Nx,1:Ny-1) = 2*T(Nx-1,1:Ny-1) - T(Nx-2,1:Ny-1);
p(Nx,1:Ny-1) = 2*p(Nx-1,1:Ny-1) - p(Nx-2,1:Ny-1);

% CC at the solid surface
rho(2:Nxs,1) = 2*rho(2:Nxs,2) - rho(2:Nxs,3);
p(2:Nxs,1) = 2*p(2:Nxs,2) - p(2:Nxs,3);

% CC at the Nxs: vertical wall
rho(Nxs,1:Nya) = 2*rho(Nxs-1,1:Nya) - rho(Nxs-2,1:Nya);
p(Nxs,1:Nya) = 2*p(Nxs-1,1:Nya) - p(Nxs-2,1:Nya);

% CC at the Nya: horizontal wall
rho(Nxs+1:Nx,Nya) = 2*rho(Nxs+1:Nx,Nya+1) - rho(Nxs+1:Nx,Nya+2);
p(Nxs+1:Nx,Nya) = 2*p(Nxs+1:Nx,Nya+1) - p(Nxs+1:Nx,Nya+2);
```

4.3 Rearward Step

The complementary problem is to see what happens in the opposite situation, when the step is now facing rearwards to the incoming flow. In this geometry, the flat plate above the solid square creates a shock wave just as the flat plate without the step would do. However, after that, the pressure falls significantly behind the step. The geometry has been delimited to 10^{-5} in the horizontal direction and to $1.5 \cdot 10^{-5}$ in the vertical direction.

This problem is less interesting than the previous one as the only meaningful thing that is to be measured is the pressure behind the step. The more interesting behaviour, that discusses the formation of the Prandtl-Meyer Expansion waves, is when the step becomes a slope and constitutes the next application.

The pressure field for a Mach 2 flow with a step height of 30% of the total vertical dimension, $4.5 \cdot 10^{-6}$, is shown in Figure 4.35.

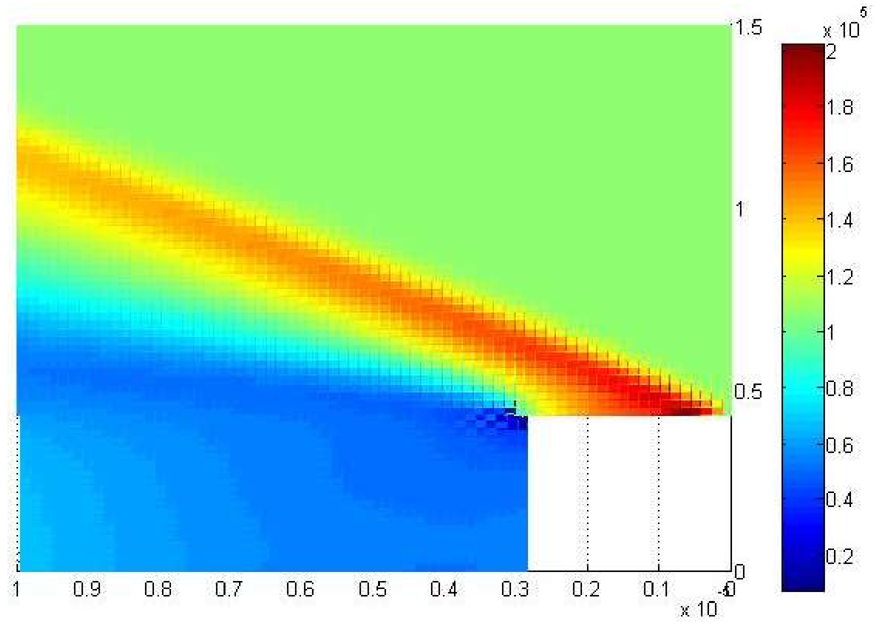


Figure 4.35. Pressure (Pa) in rearward step, Mach 2, step (3e-6 x 4.5e-6). Two-dimensional view.

The pressure increase in the shock wave created by the flat plate above the solid square is identical to the normal flat plate. However, after that the pressure falls in an expansion wave to a value lower than the external pressure.

The peak of low pressure happens just at the corner of the step, where it exists a point that almost reaches zero (and it has been a nightmare for the numerical stability of the program). After the peak, the pressure starts to recover both when advancing down close to the surface or back following the flow direction. It is seen clearly in Figure 4.36, that represents the pressure for a Mach 4 flow with a rearward step of height $3 \cdot 10^{-6}$.

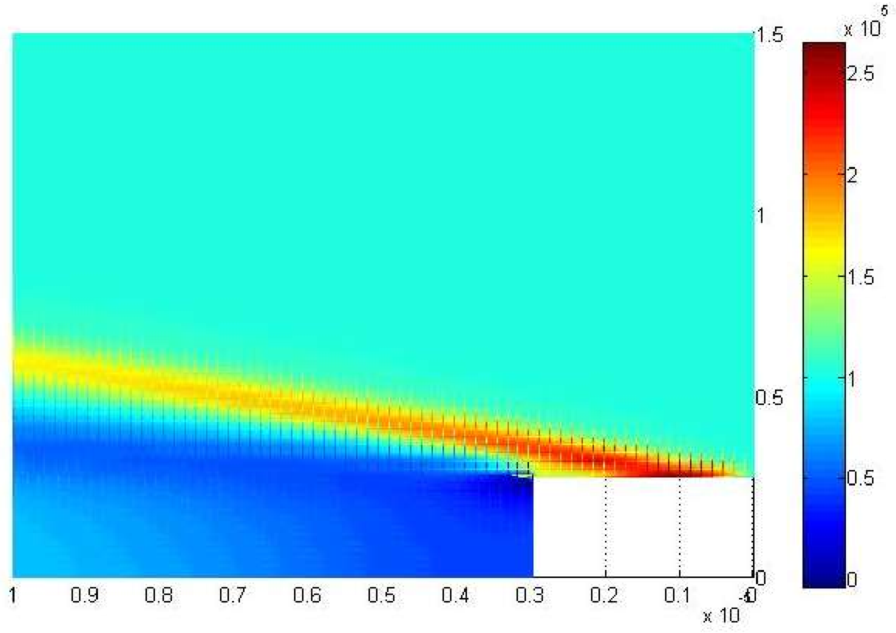


Figure 4.36. Pressure (Pa) in rearward step, Mach 4, step (3e-6 x 3e-6). Two-dimensional view.

The evolution of the mean pressure in the vertical wall is displayed in Figure 4.37. As it is showed by the two lines, now the mean pressure at the wall, in contrast with the forward step (Figure 4.33), highly depends on the step height. When the step is higher the overall mean pressure is also higher. That is probably because for the lower step the peak of low pressure represents more proportionally to the total vertical surface.

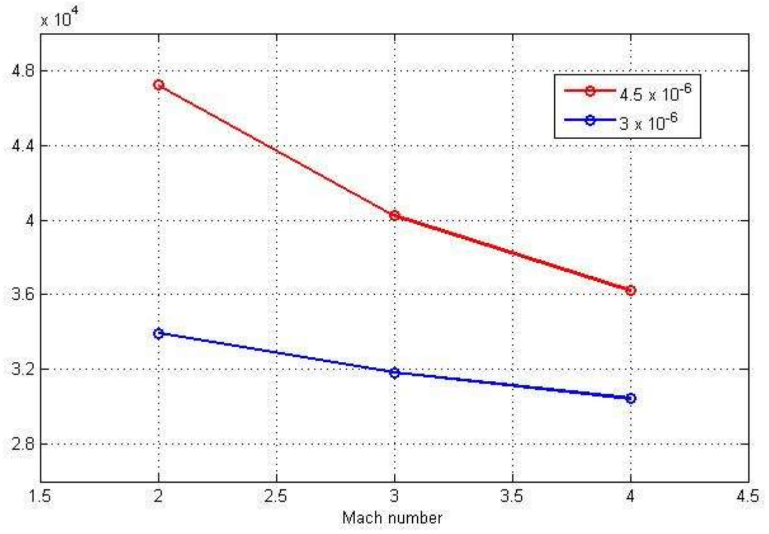


Figure 4.37. Mean pressure in (Pa) for rearward step. Dependency with step height and Mach number.

Also, a dependency is appreciated regarding the Mach number. With higher flow velocities, the vacuum effect grows behind the wall. This effect is stronger for higher steps and seems to have diminishing behaviour with the increase in Mach.

The last piece of information that is needed to be discussed is the temperature field. Now it is not similar to the pressure field as it was for the forward step. Instead, the air is heated in two different regions, as it is seen in Figure 4.38 for a Mach 2 flow with a $3.5 \cdot 10^{-6}$ step. The first region of hot air is caused by the shock wave, but after it, it appears another region in which the air is heated by friction.

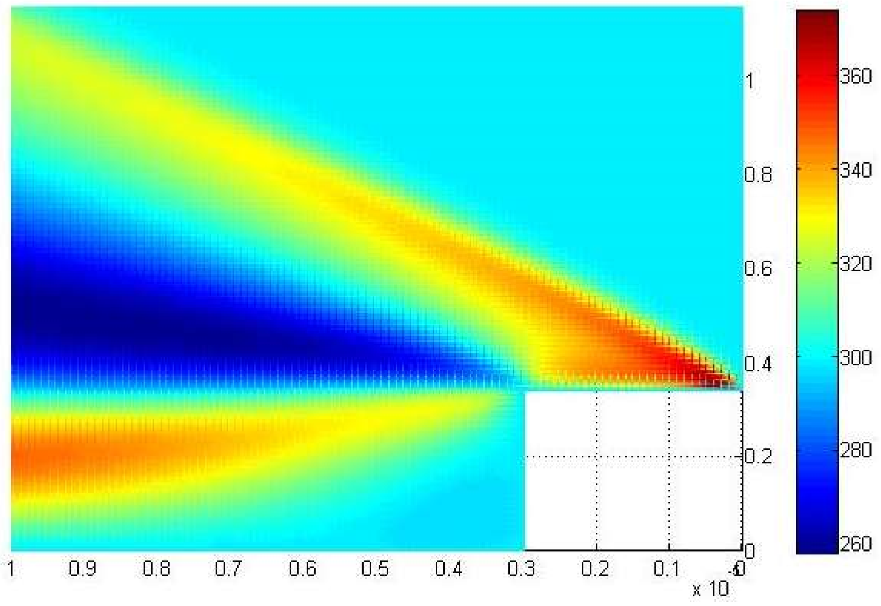


Figure 4.38. Temperature (K) in rearward step, Mach 2, step (3e-6 x 3.5e-6). Two-dimensional view.

The proof of that can be obtained by taking a look at the velocity field. In Figure 4.39 it is seen that the heated region coincides with great velocity gradients caused by friction.

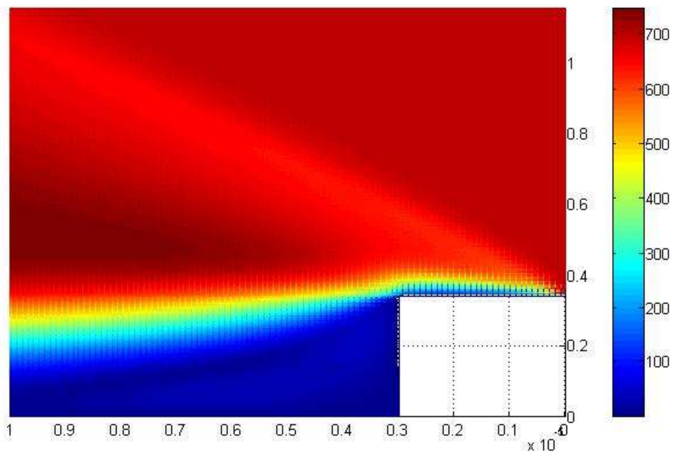


Figure 4.39 Velocity (m/s) in rearward step, Mach 2, step (3e-6 x 3.5e-6). Two-dimensional view.

Also, it is appreciated that a recirculating air region is created at the proximity of the bottom corner of the vertical wall, as isolating and analysing only the horizontal velocity field, showed in Figure 4.40, reveals that this component reaches negative values and has a circular behaviour.

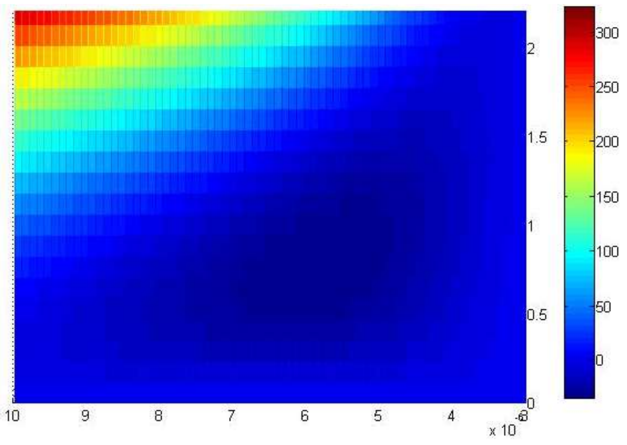
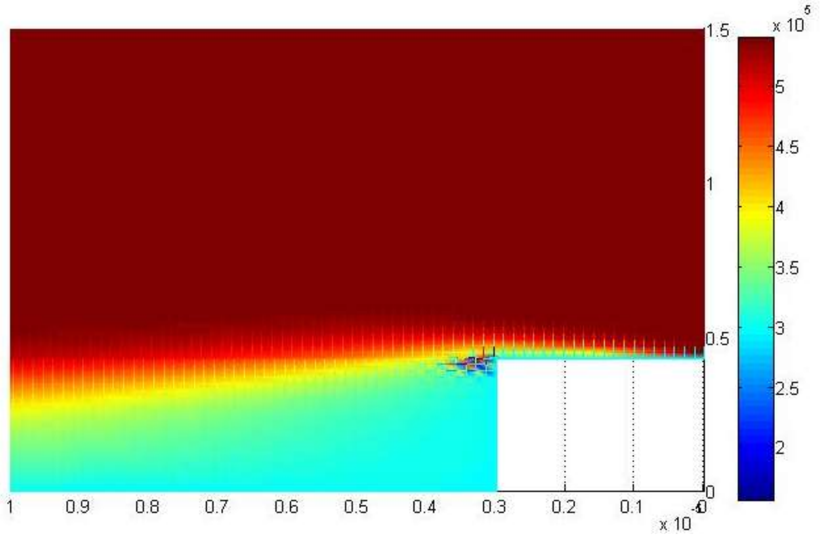


Figure 4.40. Zoom at the bottom corner, horizontal velocity (m/s) in rearward step, Mach 2, step (3e-6 x 3.5e-6). Two-dimensional view.

The rearward step is a good application to test a demonstrated fact; that the stagnation enthalpy is not affected in a shock wave. The stagnation enthalpy has the following expression:

$$h_0 = c_p T + \frac{V^2}{2}$$

As it is shown in the figure below, the presence of the shock wave seen in Figure 4.35 does not affect the stagnation enthalpy. Only the viscosity effects in the high velocity gradients do.



Stagnation enthalpy in rearward step, Mach 2, step (3e-6 x 4.5e-6).

4.3.1 Auxiliary functions for the rearward step

Initial Conditions Function

```
function [rho,u,v,T,p] =
Initial_Conditions_RearwardStep(rhoinf,uinf,Tinf,R,Nx,Ny)

% Step X direction beginning
Nxs = floor(0.3*Nx);

% Step height
Nya = floor(0.2*Ny);

u(1:Nx,1:Ny) = uinf;
u(1:Nx,1:Nya) = 0;

v(1:Nx,1:Ny) = 0;

rho(1:Nx,1:Ny) = rhoinf;
T(1:Nx,1:Ny) = Tinf;
p(1:Nx,1:Ny) = R*rho.*T;
```

Boundary Conditions Function

```
function [rho,u,v,T,p] =
Boundary_Conditions_RearwardStep(rho,u,v,T,p,Nx,Ny,Tinf,pinf)

% Step X direction beginning
Nxs = floor(0.3*Nx);

% Step height
Nya = floor(0.2*Ny);

% Force variables in the solid square
u(1:Nxs,1:Nya) = 0;
v(1:Nxs,1:Nya) = 0;

T(1:Nxs,1:Nya) = Tinf;
p(1:Nxs-1,1:Nya-1) = pinf;

% CC at the right side
rho(Nx,1:Ny-1) = 2*rho(Nx-1,1:Ny-1) - rho(Nx-2,1:Ny-1);
u(Nx,1:Ny-1) = 2*u(Nx-1,1:Ny-1) - u(Nx-2,1:Ny-1);
v(Nx,1:Ny-1) = 2*v(Nx-1,1:Ny-1) - v(Nx-2,1:Ny-1);
T(Nx,1:Ny-1) = 2*T(Nx-1,1:Ny-1) - T(Nx-2,1:Ny-1);
p(Nx,1:Ny-1) = 2*p(Nx-1,1:Ny-1) - p(Nx-2,1:Ny-1);

% CC at the solid surface
rho(Nxs:Nx,1) = 2*rho(Nxs:Nx,2) - rho(Nxs:Nx,3);
%rho_p(2:Nx,1) = rho(2:Nx,1) - dt/2/dy*( 3*rho(2:Nx,1).*u(2:Nx,1) -
4*rho(2:Nx,2).*u(2:Nx,2) + rho(2:Nx,3).*u(2:Nx,3));
p(Nxs:Nx,1) = 2*p(Nxs:Nx,2) - p(Nxs:Nx,3);

% CC at the Nxs: vertical wall
rho(Nxs,1:Nya) = 2*rho(Nxs+1,1:Nya) - rho(Nxs+2,1:Nya);
p(Nxs,1:Nya) = 2*p(Nxs+1,1:Nya) - p(Nxs+2,1:Nya);

% CC at the Nya: horizontal wall
rho(1:Nxs,Nya) = 2*rho(1:Nxs,Nya+1) - rho(1:Nxs,Nya+2);
p(1:Nxs,Nya) = 2*p(1:Nxs,Nya+1) - p(1:Nxs,Nya+2);
```

4.4 Rearward Slope and The Prandtl-Meyer Expansion Waves

It is time to use the non-uniform grid transformation for the first time, turning the rearward vertical wall into a slope. In this problem the horizontal and vertical dimensions of the computational domain have been set to 10^{-5} and $1.63 \cdot 10^{-5}$ respectively. The beginning and height of the slope can be modified at will, and the slope falls from the indicated point till the end of the computational domain.

Firstly, it is going to be studied the case in which still exist a portion of flat plate above the solid surface. In this case the flat plate creates the typical shock wave that has been seen in previous applications. This pressure field showed in Figure 4.41 for a 35º slope with Mach 3 shows precisely that.

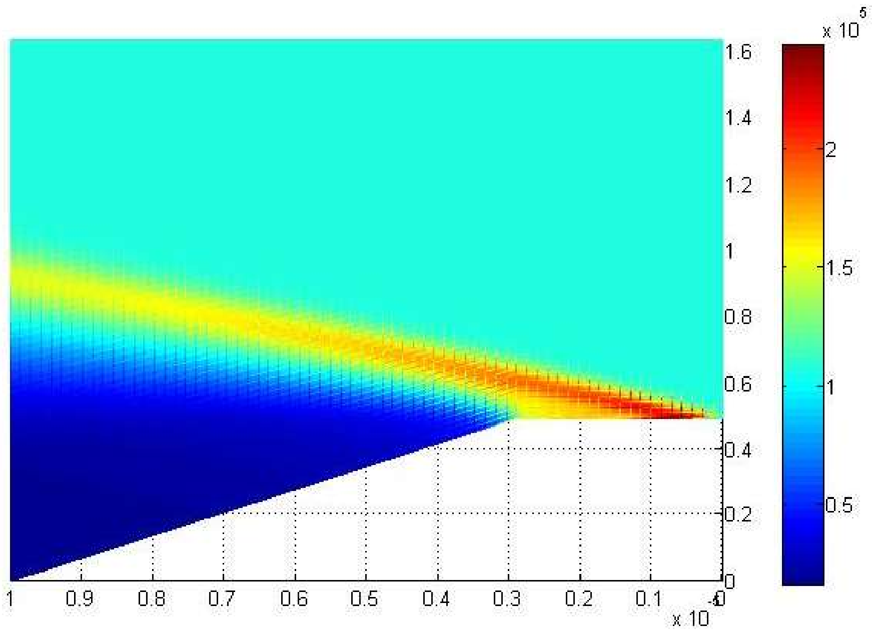


Figure 4.41. Pressure (Pa) in rearward slope, Mach 3, Slope angle 35º. Two-dimensional view.

The temperature field on the other hand, Figure 4.43, reveals that the second region of heated air, at least in this configuration, remains attached to the slope. That may be an indication that the recirculating air region has not been formed. In addition, the temperature field reveals that there exists a region in which the temperature falls below the external value. This is probably because there the air is accelerated due to the expansion wave to velocities higher than the incoming flow.

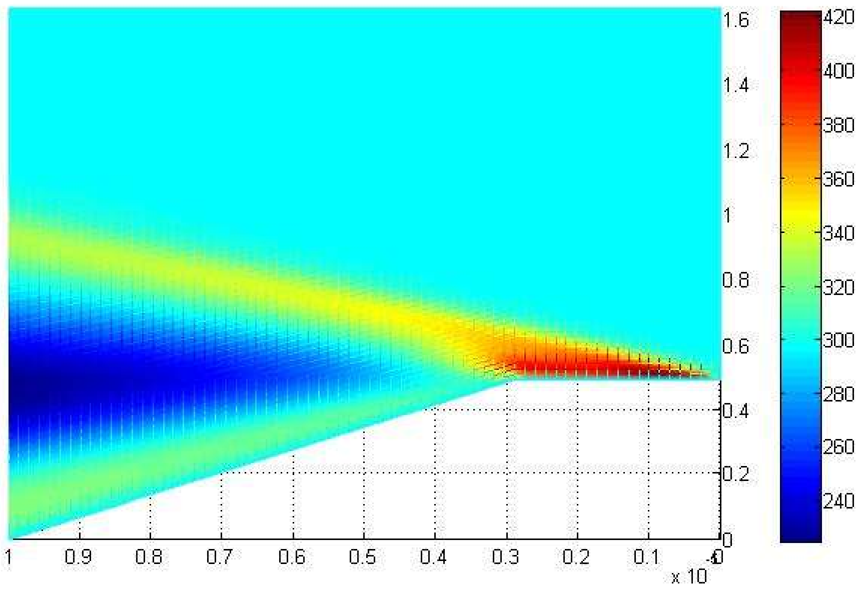


Figure 4.43. Temperature (K) in rearward slope, Mach 3, Slope angle 35º. Two-dimensional view.

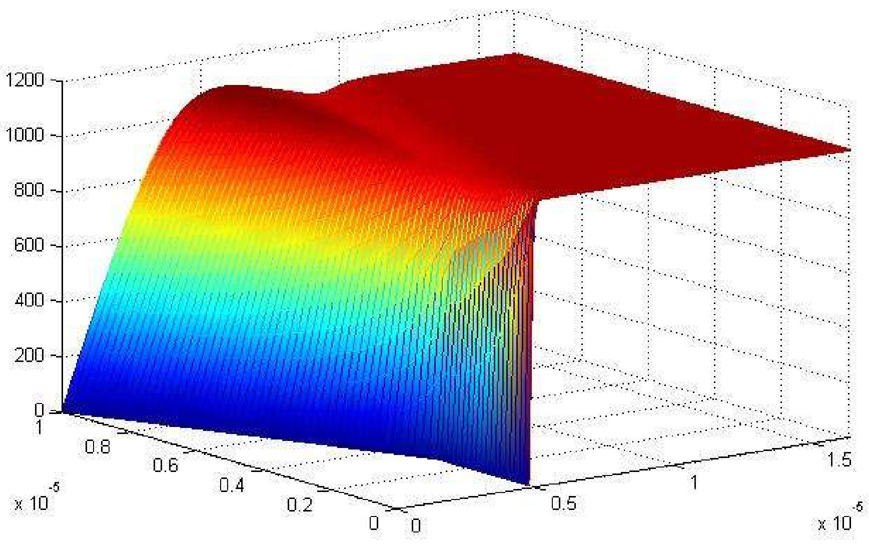


Figure 4.42. Velocity (m/s) in rearward slope, Mach 3, Slope angle 35º. Three-dimensional view.

Looking at the velocity field, Figure 4.42, it is confirmed that, as it was hinted by the temperature, there is no recirculating area. Also, it is seen that at the area in which the temperature falls, the velocity of the air is accelerated to even a greater value than the external flow.

It is therefore interesting to see the surface pressure distribution and its dependency with the slope angle and the Mach number. The Figure 4.44 shows the pressure surface profiles for different angles of the slope.

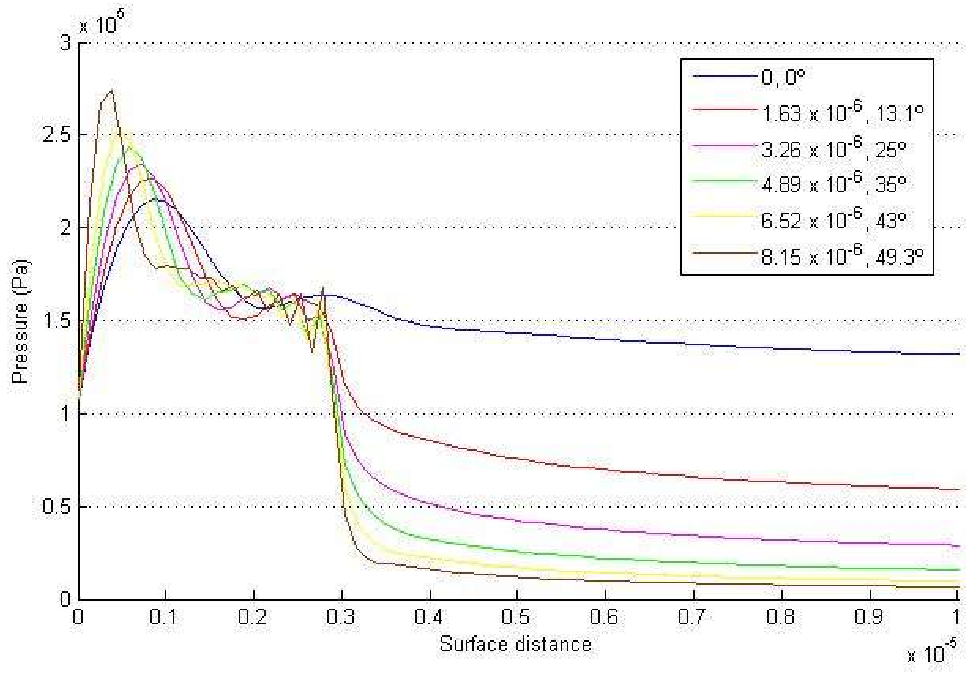


Figure 4.44. Pressure at surface for rearward slope, Mach 3, dependency with slope angle.

The first curve corresponds to the flat plate and coincides with the results of section 4.1. It is seen that the presence of a minimum slope already affects greatly to the results after the slope corner. Then, the increase in the slope angle decreases the pressure in a diminishing fashion with the angle. The behaviour of the pressure at the surface after the step is asymptotical and is not expected to differ much more after the end of the computational domain.

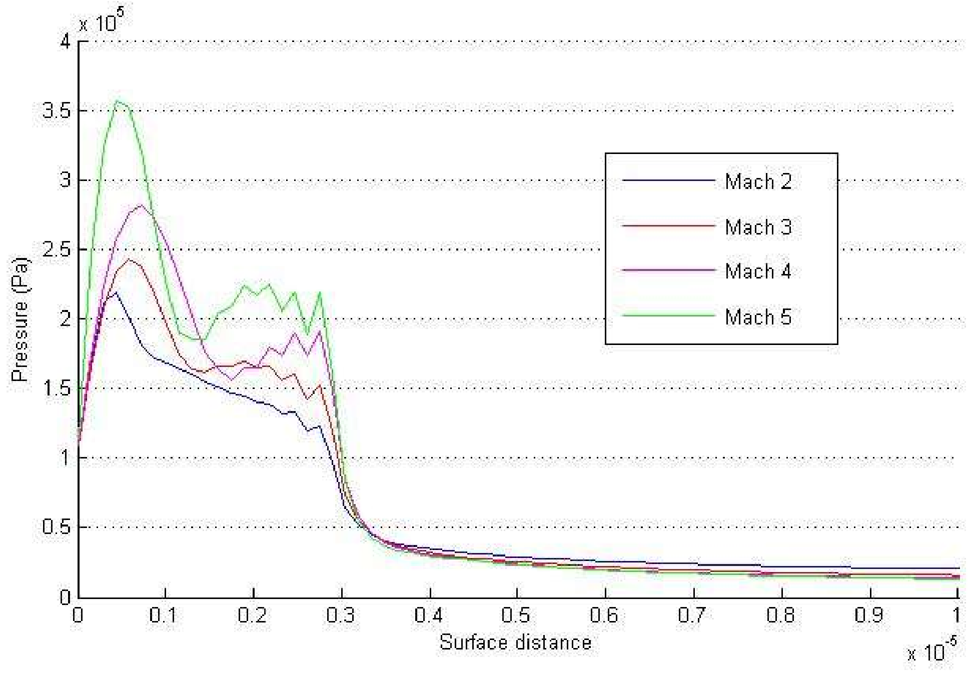


Figure 4.45. Pressure at surface for rearward slope, slope angle 35º, dependency with Mach number.

When varying the Mach number, what is observed is that the pressure peak at the beginning is accentuated, as was expected due to Figure 4.16, and what happens after that is that, at least for an angle as big as 35 degrees, the pressure after the corner does not depend much on the Mach number.

4.4.1 Prandtl-Meyer expansion waves

The most interesting part of this application is when the flat plate surface is eliminated, letting only the slope. In this case, if the slope angle is big enough, there is no shock wave created. It is interesting to investigate the transition between the behaviour of a flat plate and this other.

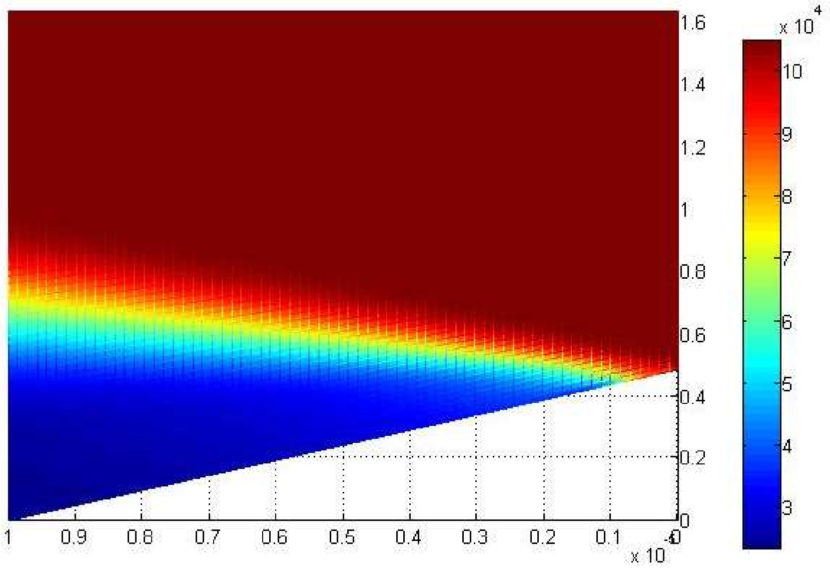


Figure 4.46. Pressure (Pa) in rearward slope, Mach 3, Slope angle 26º, no flat plate.

This Figure 4.46 represents the pressure for a Mach three flow with an angle of 26º. Now the higher values of pressure correspond to the unperturbed flow, and the plate only creates an expansion wave. It is known that it is an expansion wave because it decreases the pressure of the air and accelerates it.

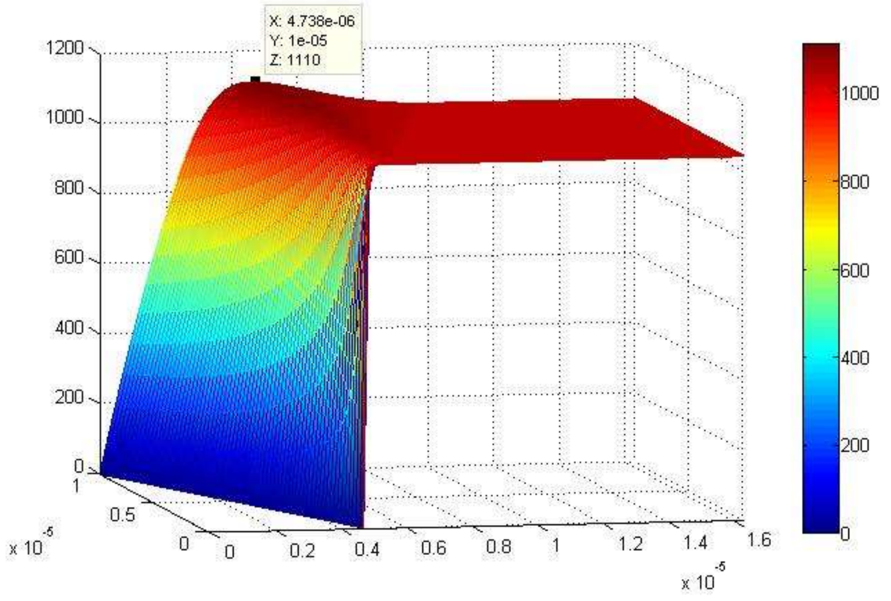


Figure 4.47. Velocity (m/s) in rearward slope, Mach 3, Slope angle 26º, no flat plate.

Looking at Figure 4.47 it is seen that the air accelerates over 100 m/s with respect to the unperturbed air stream in the expansion wave. The elimination of the shock wave also removes the heated air area created by it. Instead, the air cools in the expansion wave. Close to the surface the air heats again due to friction effects, as it corresponds to high velocity gradients observed in Figure 4.47.

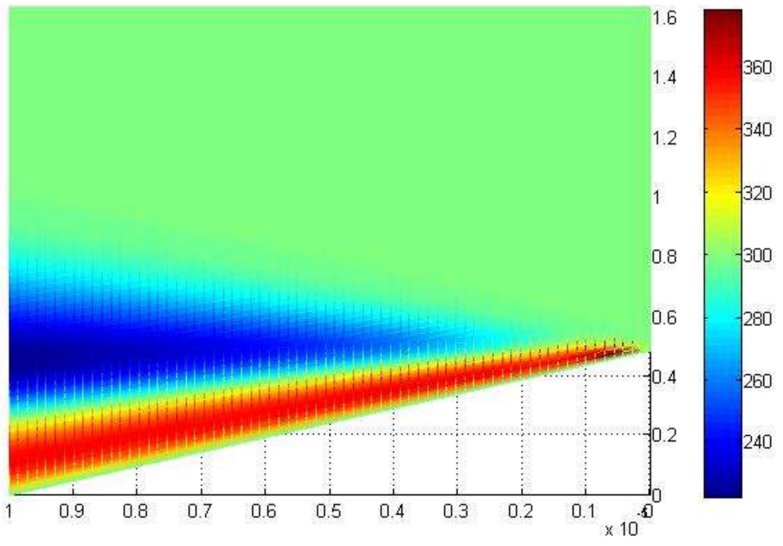


Figure 4.48. Temperature (K) in rearward slope, Mach 3, Slope angle 26º, no flat plate.

Validating our results is very difficult as no exactly similar numerical experiment has been found, but it can be tried by comparing them with the results obtained by John Anderson, again in "*Computational Fluid Dynamics*" [Ref 1], in chapter 8. There, the analytical results for the expansion wave problem is demonstrated.

The set-up and the problem in general that is solved in this chapter is quite different, so differences in the results are to be expected. Specifically, the problem solved in the reference has a free slip condition at the surface, and is treated with a step marching method, so its dimensions are unreachable by our code, of over 10 meters. However, recreating the 5,352 degree angle and the Mach 2 flow in this project's code, we obtain the following results.

The density at the surface at end of the computational domain is $1.071\ \mathrm{kg}/m^3$, that compared with the $0.992\ \mathrm{kg}/m^3$ of the analytical solution represents a 7.3% difference. Counting on the differences in problems mentioned, it is reasonable to present the results obtained as not far from reality. Making the same comparison it is obtained an 8.7% difference in pressure value, $0.81 \cdot 10^5$ Pa compared to the analytical $0.734 \cdot 10^5$ Pa, again reasonably far away given the differences in set up.

When the slope angle is small a shock wave is created anyway, similarly to what happened to the flat plate. See for instance the pressure field when the angle is lowered in Figure 4.49 and compare it to Figure 4.46.

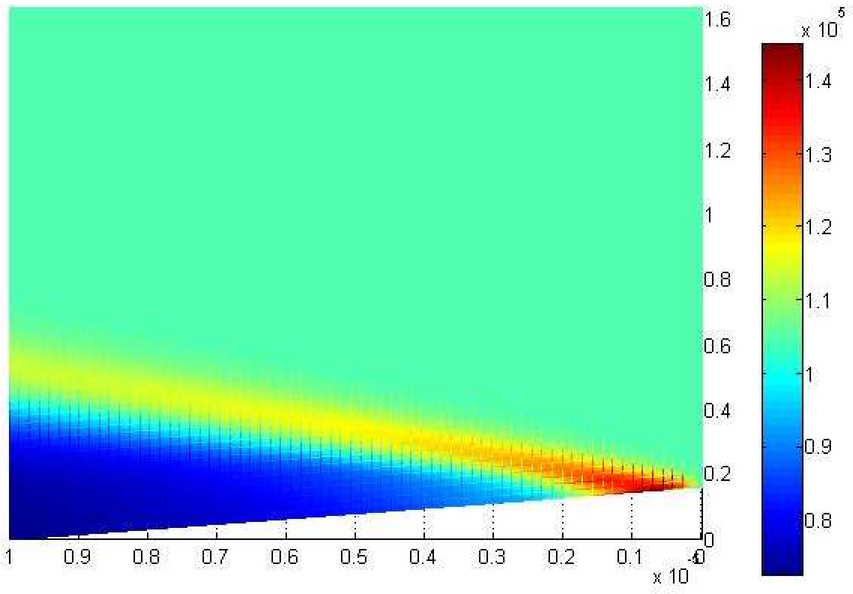


Figure 4.49. Pressure (Pa) in rearward slope, Mach 3, Slope angle 9.25º, no flat plate.

Thus, the transition happens between these two values of angle. Looking at the surface pressure for different angles shown in Figure 4.50 it is seen that the pressure peak at the

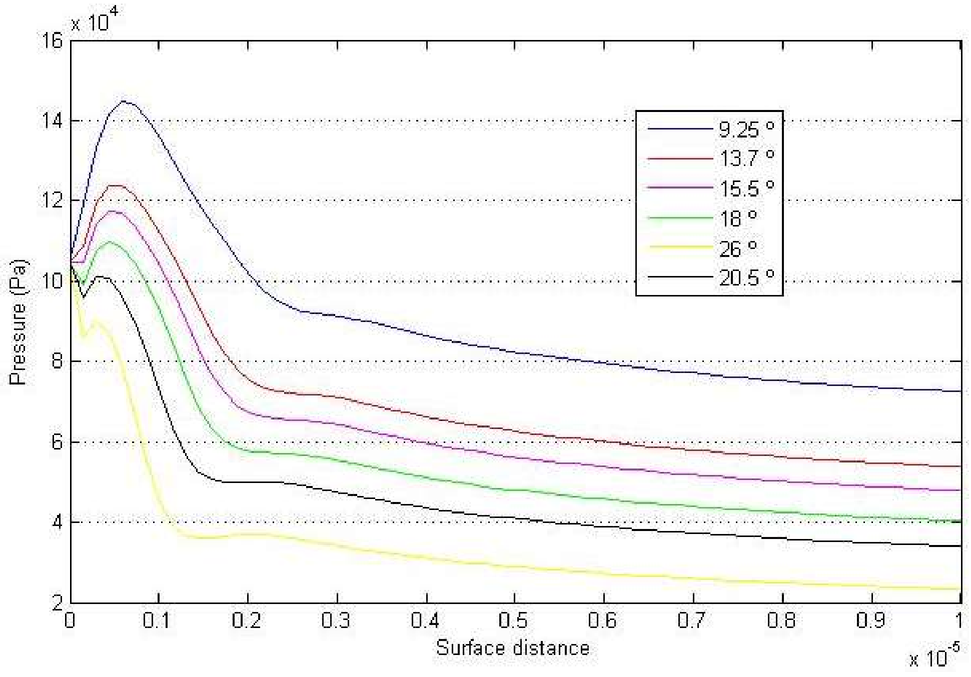


Figure 4.50. Pressure at surface for rearward slope, Mach 3, dependency with slope angle, no flat plate.

beginning of the surface is reduced with the angle increase. With 18 degree angle it can be considered insignificant and with 20 degrees it has already disappeared completely.

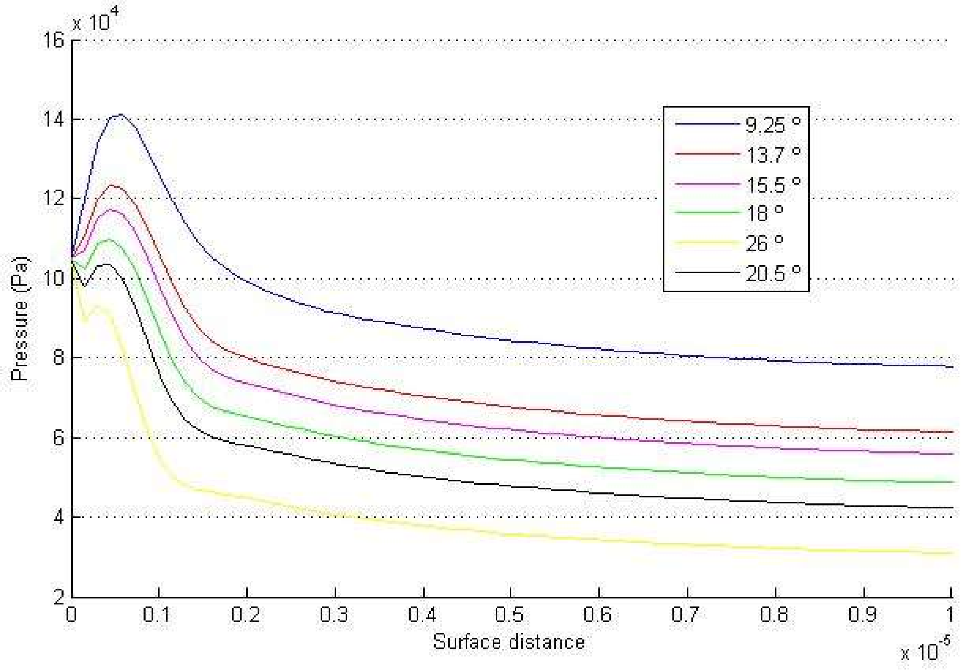


Figure 4.52. Pressure at surface for rearward slope, Mach 2, dependency with slope angle, no flat plate.

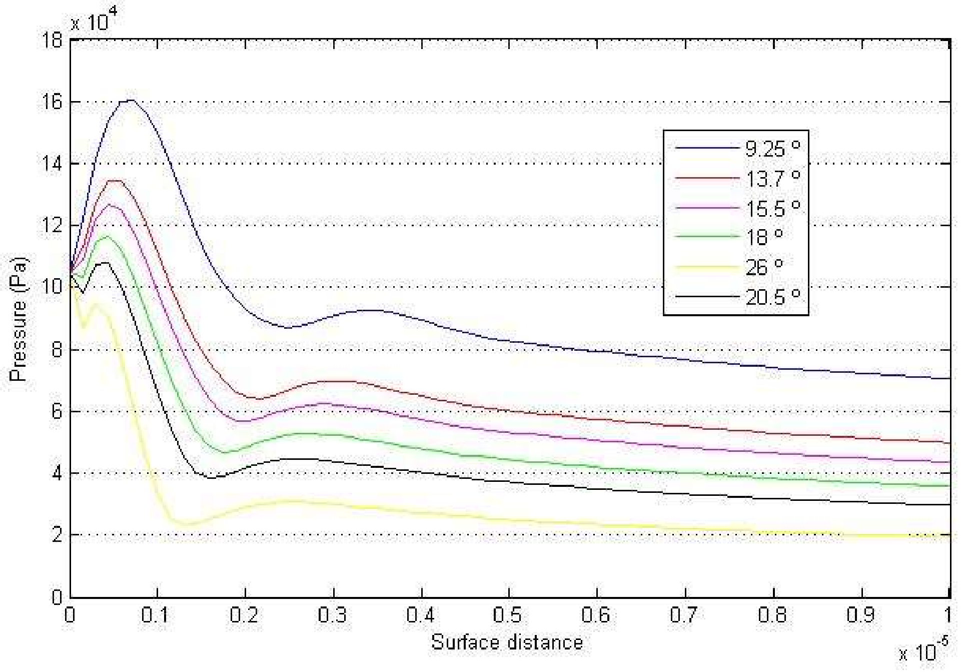


Figure 4.51. Pressure at surface for rearward slope, Mach 4, dependency with slope angle, no flat plate.

By comparing Figure 4.50 with the equivalent plots at different Mach numbers shown in Figure 4.52 and Figure 4.51, at Mach 2 and Mach 4 respectively, it is seen that the behaviour almost does not change with the Mach number, and the pure expansion wave keeps happening at 20 degrees angle approximately.

Finally, when analysing the pressure profiles dependency with the Mach number, it is seen in Figure 4.54 that a suction peak is created and accentuated when increasing the Mach near the start of the slope. This behaviour is similar to the one seen in section 4.3 in the upper corner of the rearward step, and as it did, has caused serious problems when dealing with the numerical stability of the code.

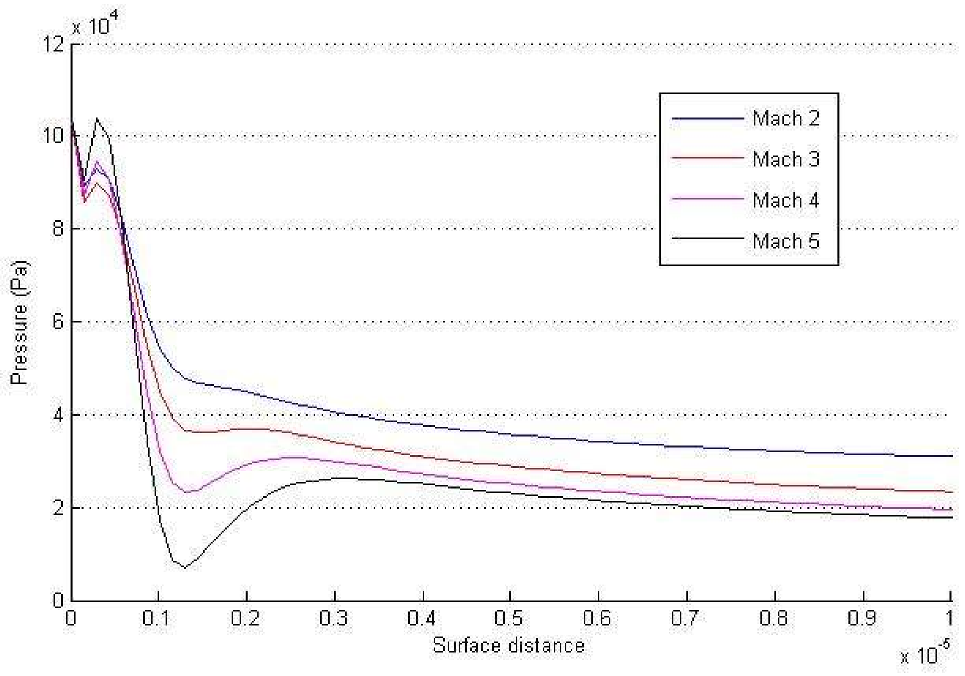


Figure 4.54. Pressure at surface for rearward slope, slope angle 26º, dependency with Mach number.

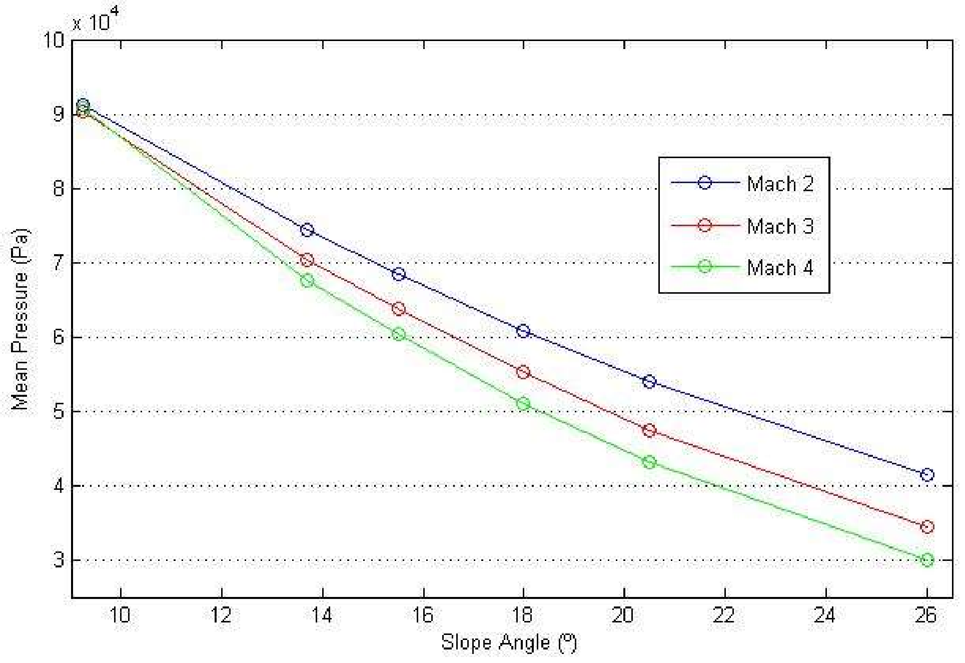


Figure 4.53. Mean pressure at surface of the slope. Dependency with Mach number and slope angle.

To sum up, the mean pressure values at the wall are shown in the Figure 4.53. The overall pressure is reduced when increasing the Mach number and with the slope angle. Also, it is shown that when reducing the angle value the dependency with the Mach starts fading, until eventually flips to become a higher pressure with higher Mach, as expected for the flat plate.

The diminishing pressure causes a backward force at the surface, if the external pressure is used as reference it is positive in the upward-backward direction. Its plot is very similar to the mean pressure plot, as the pressure and the force are directly related. However, when calculating the resistance coefficient in the horizontal direction taking into account the pressure forces, Figure 4.55, it is seen that due to the increase velocity for higher Mach numbers, even with greater force, the coefficient is lower for them. The dependency with the slope angle remains invariable.

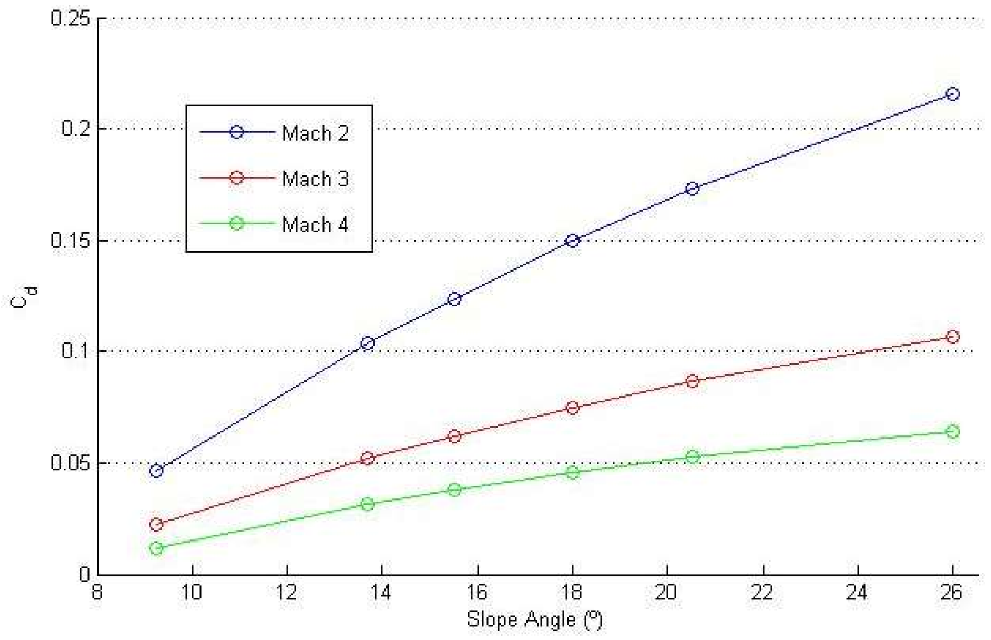


Figure 4.55. C_d due to pressure for the horizontal direction in rearward slope. Dependency with Mach number and slope angle.

4.4.2 Auxiliary functions for the rearward slope

Geometry Function

```
function
[x,y,dxi,deta,dxidx,dxidy,detadx,detady,J,dxdxi,dxdeta,dydxi,dydeta] =
Geometry_RearwardSlope(Nxi,Neta,LHOR,LVER)

xi_LHOR = LHOR;
eta_LVER = LVER;

dxi = xi_LHOR/(Nxi-1);
deta = eta_LVER/(Neta-1);

for j=1:Neta
    xi(:,j) = 0:dxi:xi_LHOR;
end
for i=1:Nxi
    eta(i,:) = 0:deta:eta_LVER;
end

% Slope X direction beginning
s = floor(0.0*Nxi);
xis = LHOR*(s-1)/(Nxi-1);
```

```
% Slope height
a = 0.23*LVER;

y0(1:s) = a;
y0(s+1:Nxi) = a - ( xi(s+1:Nxi) - xis )*a/( LHOR - xis );

for i=1:Nxi
    for j=1:Neta
        x(i,j) = xi(i,j);
    end
    y(i,:) = linspace(y0(i),LVER,Neta);
end

% Direct metrics DC
dxidx = Dx(xi,x);
dxidy = Dy(xi,y);
detadx = Dx(eta,x);
detady = Dy(eta,y);

% Inverse metrics DC
dxdxi = Dx(x,xi);
dxdeta = Dy(x,eta);
dydeta = Dy(y,eta);
dydxi = Dx(y,xi);

J = dxdxi.*dydeta - dydxi.*dxdeta;
```

Additional Functions (also for the leading edge)

```
function dudx = Dx(u,x)

[m,~] = size(x);

dudx(1,:) = ( u(2,:) - u(1,:) )./( x(2,:) - x(1,:) );
dudx(2:m-1,:) = ( u(3:m,:) - u(1:m-2,:) )./( x(3:m,:) - x(1:m-2,:) );
dudx(m,:) = ( u(m,:) - u(m-1,:) )./( x(m,:) - x(m-1,:) );


function dudx = Dy(u,x)

[~,n] = size(x);

dudx(:,1) = ( u(:,2) - u(:,1) )./( x(:,2) - x(:,1) );
dudx(:,2:n-1) = ( u(:,3:n) - u(:,1:n-2) )./( x(:,3:n) - x(:,1:n-2) );
dudx(:,n) = ( u(:,n) - u(:,n-1) )./( x(:,n) - x(:,n-1) );
```

Initial Conditions Function

```
function [rho,u,v,T,p] =
Initial_Conditions_RearwardSlope(rhoinf,uinf,Tinf,R,Nxi,Neta)

rho(1:Nxi,1:Neta) = rhoinf;
u(1:Nxi,1:Neta) = uinf;
u(2:Nxi,1) = 0;
v(1:Nxi,1:Neta) = 0;
T(1:Nxi,1:Neta) = Tinf;
```

```
p(1:Nxi,1:Neta) = R*rho.*T;
```

Boundary Conditions Function

```
function [rho,u,v,T,p] =
Boundary_Conditions_RearwardSlope(rho,u,v,T,p,Nxi,Neta)

% CC at the right side
rho(Nxi,2:Neta-1) = 2*rho(Nxi-1,2:Neta-1) - rho(Nxi-2,2:Neta-1);
u(Nxi,2:Neta-1) = 2*u(Nxi-1,2:Neta-1) - u(Nxi-2,2:Neta-1);
v(Nxi,2:Neta-1) = 2*v(Nxi-1,2:Neta-1) - v(Nxi-2,2:Neta-1);
T(Nxi,2:Neta-1) = 2*T(Nxi-1,2:Neta-1) - T(Nxi-2,2:Neta-1);
p(Nxi,2:Neta-1) = 2*p(Nxi-1,2:Neta-1) - p(Nxi-2,2:Neta-1);

% CC at the solid surface
rho(2:Nxi,1) = 2*rho(2:Nxi,2) - rho(2:Nxi,3);
p(2:Nxi,1) = 2*p(2:Nxi,2) - p(2:Nxi,3);
```

4.5 Leading Edge: Study of Shock Wave Angle

It is well known for the study of aerodynamic applications, and is comprehensibly explained by A. B. Ripoll and Miguel Pérez-Saborid in "*Fundamentos y aplicaciones de la mecánica de fluidos*" [Ref 3], that a leading edge that has a certain angle, frontally receiving a supersonic flow, creates an oblique shock wave to adequate the flow to its solid geometry. In this section this problem is going to be treated and the results obtained compared with the analytical solutions for oblique shock waves.

Before anything else, a description of the geometry is given. It consists of an edge that has a certain angle facing the stream of air. The horizontal dimension is set to $10^{-5}\ m$ and the edge is situated at half this distance. The vertical dimension for the computational domain is $1.63 \cdot 10^{-5}\ m$ in the positive direction and the same value in the negative direction.

The flow created is symmetrical, as can be seen in Figure 4.56 for a Mach 2 flow with an edge semi-angle of 40 degrees, as the problem geometry and initial and boundary conditions also are.

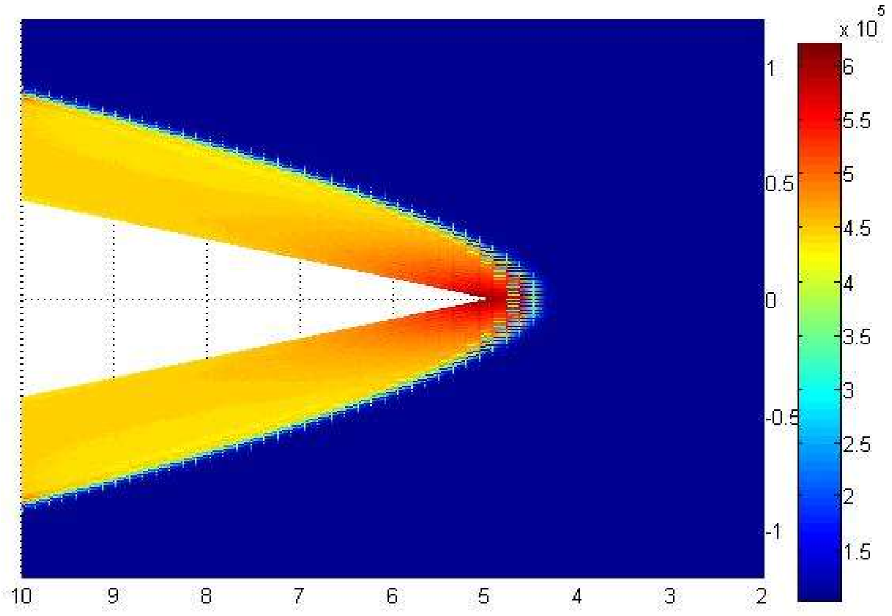


Figure 4.56. Pressure (Pa) for leading edge, Mach 2, edge semi-angle 40º.

The surface is set at constant temperature. That creates an effect seen in Figure 4.57, in which the air is not heated if is arrives exactly from the symmetrical line, while the rest of the flow is affected by the shock wave and heated substantially.

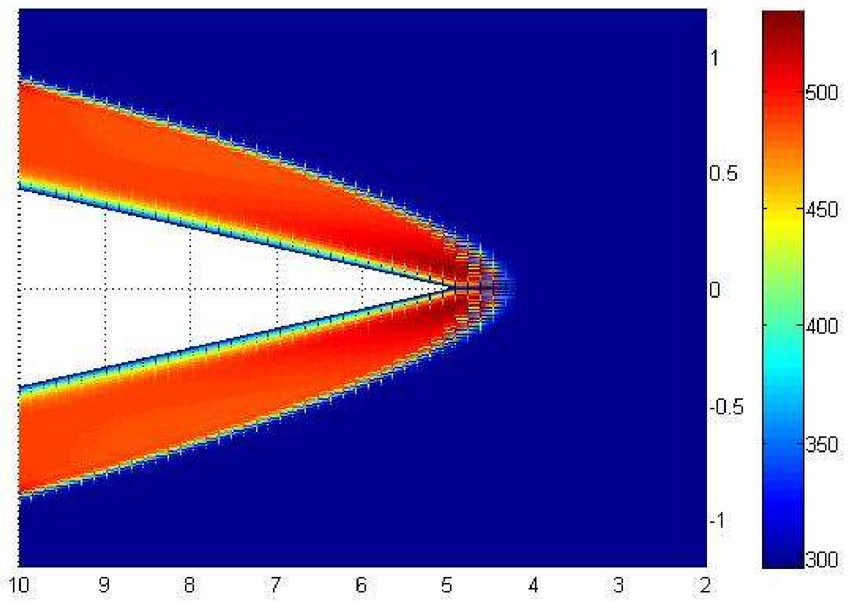


Figure 4.57. Temperature (K) for leading edge, Mach 2, edge semi-angle 40º

However, not always happens that the oblique shock wave begins before the leading edge. For a Mach three flow with a 25 degrees semi-angle, the shock wave begins exactly at the leading edge. When that happens, it is said that the shock wave is anchored to the edge.

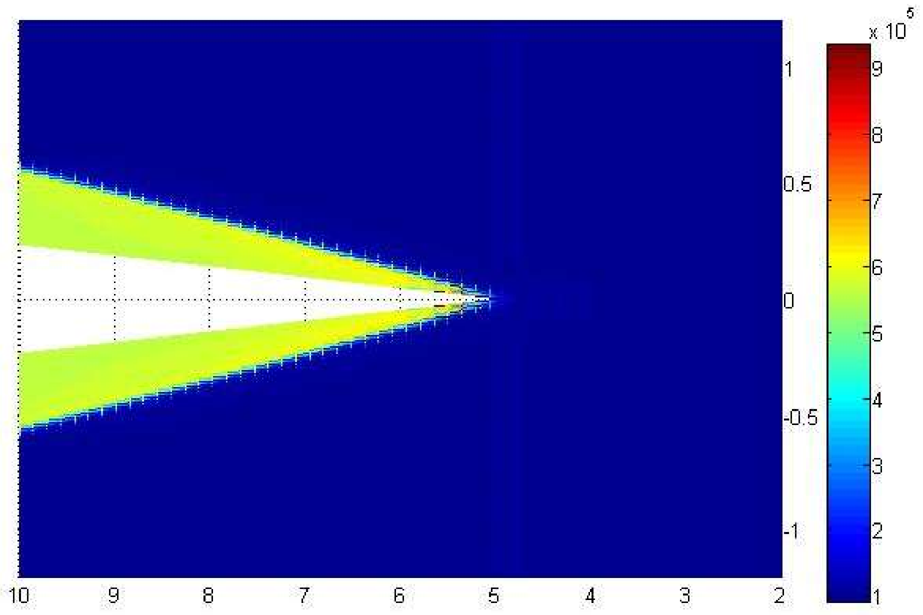


Figure 4.58. Pressure (Pa) for leading edge, Mach 3, edge semi-angle 25º.

It is interesting to know the pressure that the leading edge creates at the surface and its dependency with the Mach number and the edge semi-angle. For a Mach two flow the pressure profiles at the surface of the edge, plotted with respect to the distance in the X direction, are shown in Figure 4.59.

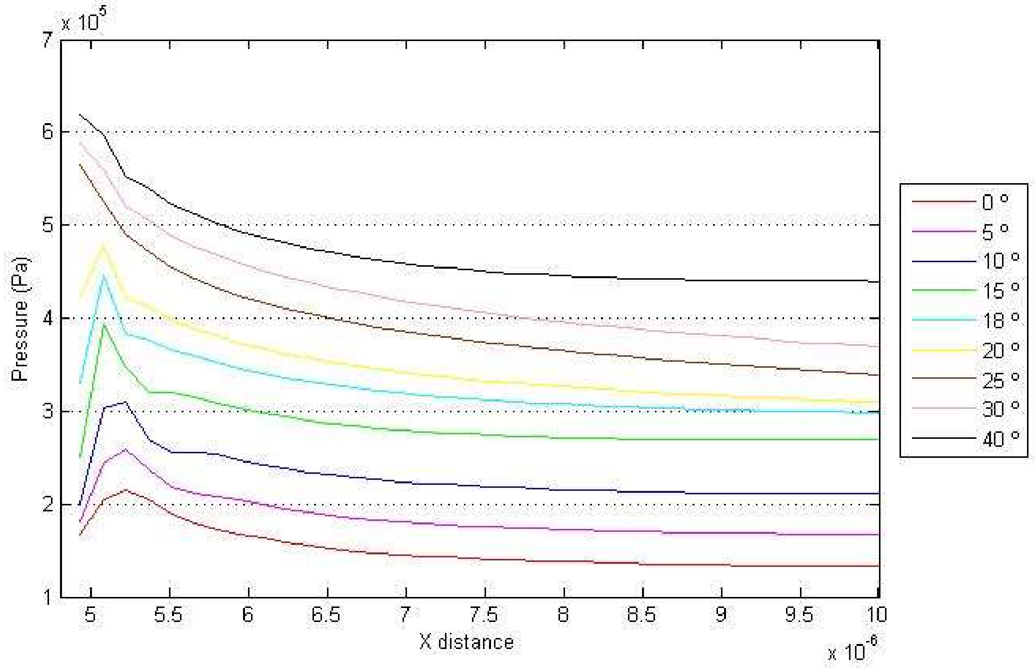


Figure 4.59. Pressure at the surface for a leading edge, Mach 2, dependency with edge semi-angle.

There is a change of behaviour between 20 and 25 degrees. With low angles the pressure rises at the beginning of the surface, signalling that the shock wave is anchored. After that the pressure at the surface starts high and only falls, a sign that the shock wave is no longer anchored.

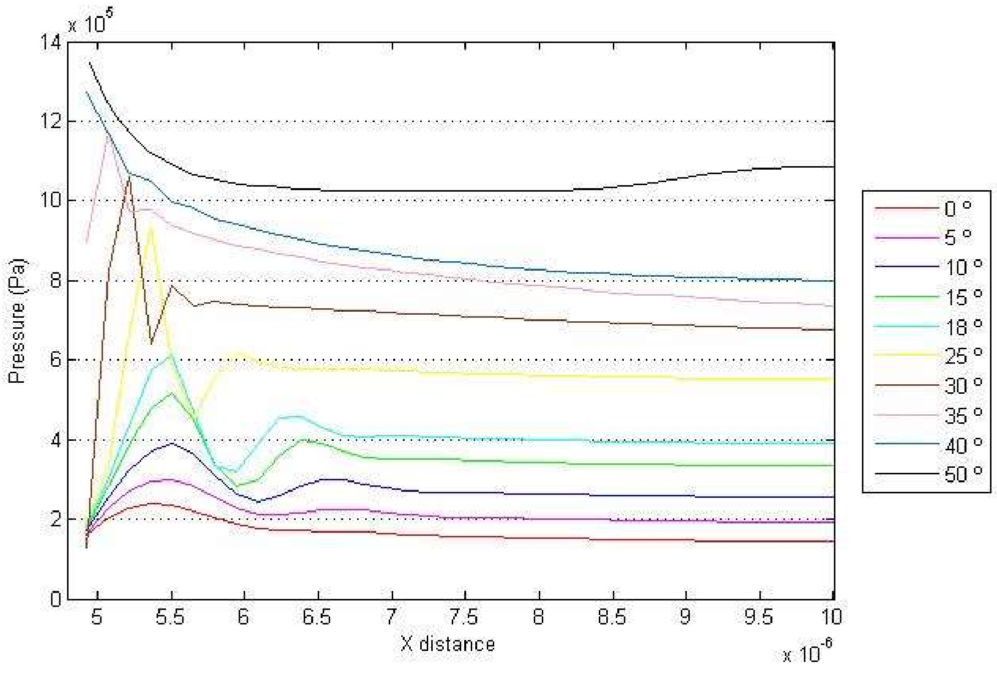


Figure 4.60. Pressure at the surface for a leading edge, Mach 3, dependency with edge semi-angle.

Looking at figures Figure 4.60 and Figure 4.61 that show the same pressure profiles for Mach three and four respectively, it is clear that this behaviour also depends on the Mach number. For Mach three and four, at a semi-angle of 30 degrees, the shock wave is still anchored. Also, it is appreciated that a peak of pressure followed by a peak of suction is accentuated by both the angle and the Mach number. At the beginning of the edge for Mach four, with the higher values of semi-angle, the flow seems to become turbulent.

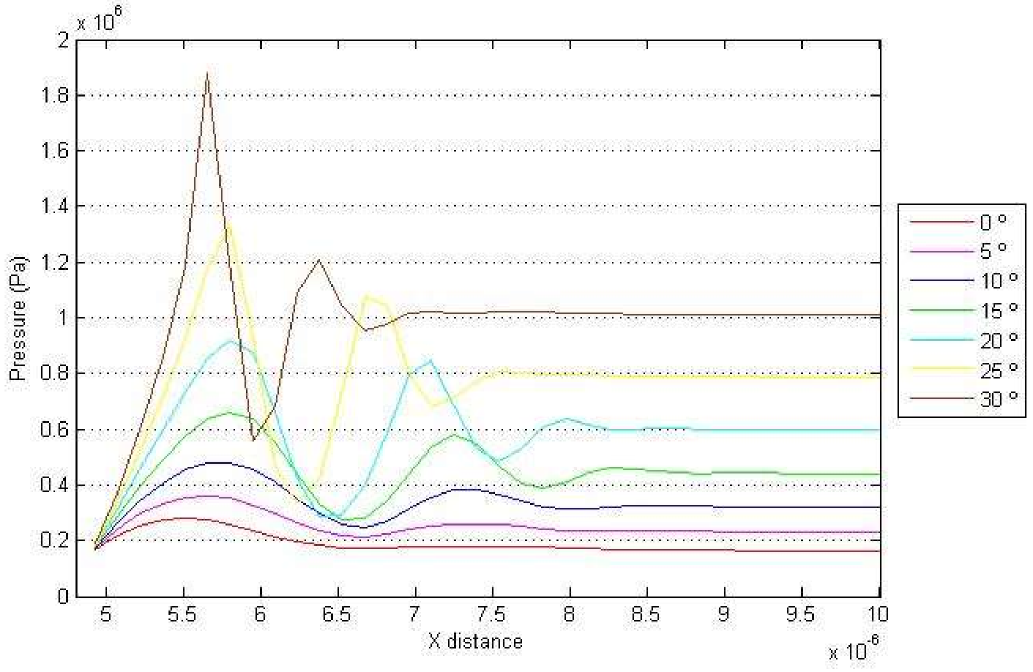


Figure 4.61. Pressure at the surface for a leading edge, Mach 4, dependency with edge semi-angle.

From [Ref 3] it is known that the analytical solution for the angle of an oblique shock wave depends on the angle of the flow after the wave and the Mach number with the following equation.

$$tan\, \alpha \;=\; 2\, cot\, \beta \left[\frac{M_1^2\, sin^2\, \beta \;-\; 1}{M_1^2\,(\gamma \;+\; cos\, 2\beta\,) \;+\; 2}\right]$$

Equation 4.4

Where γ is the specific heat ratio and M_1 is the Mach number of the incoming flow before the shock wave. α is the angle of the air stream after the shock wave, which in this case is the same as the edge semi-angle, and β the angle that the shock wave forms with respect to the horizontal direction.

Looking at the pressure fields to locate the shock wave, the angle β can be measured in each configuration that has been tried in this project. When plotting the lines corresponding to Equation 4.4, and plotting the pairs of values of edge semi-angle and shock wave angle created measured, the result is Figure 4.62.

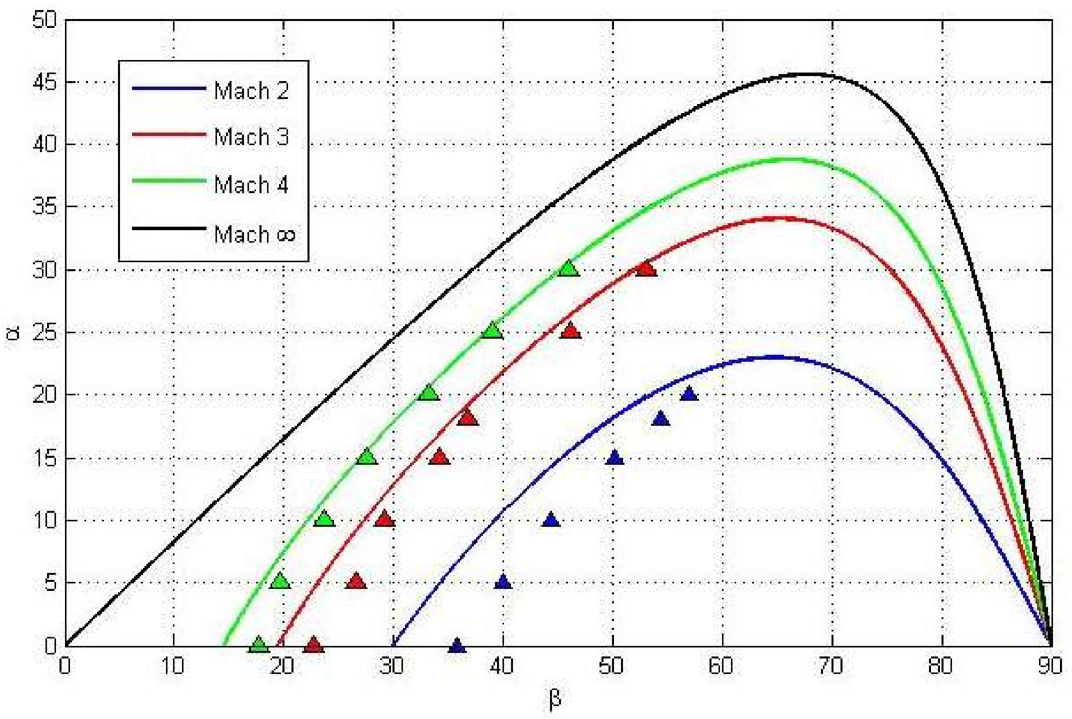


Figure 4.62. Oblique shock wave angle with respect to leading edge semi-angle and Mach number. Comparison between analytical and numerical results.

There, it can be seen that the results obtained match reasonably well with the theoretical curves. Specially for Mach three and four at high angles the results fit almost perfectly. However, when decreasing the edge semi-angle the accuracy decreases significantly, most notably when going under ten degrees, where the angle of the oblique shock wave is overestimated by the code. This also happens with Mach two, where overall the shock wave angle is overestimated, although the accuracy also grows with the leading edge semi-angle.

The Figure 4.63 shows that the value of the mean pressure at the surface, and thus the resistant force applied to the leading edge, is highly dependent on the edge semi-angle. Moreover, the dependency is accentuated when increasing the Mach number.

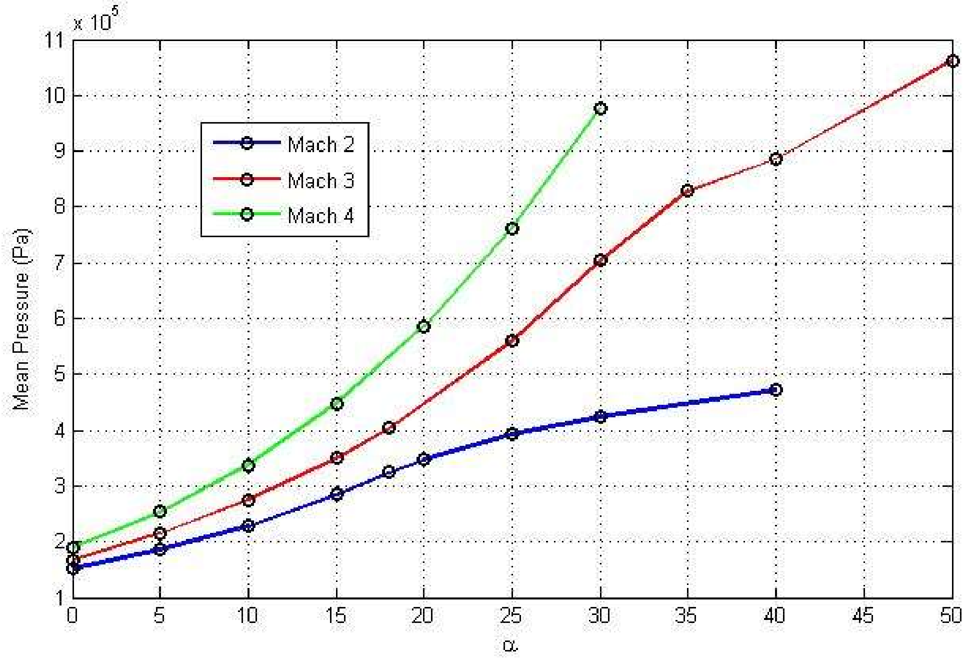


Figure 4.63. Mean pressure at the surface for leading edge, dependency with Mach number and leading edge semi-angle.

When comparing the Figure 4.63 to Figure 4.33, it can be seen that the leading edge creates much lower pressures, about one order of magnitude lower, than the forward step. However, it is expected that their behaviours become similar, if not identical, when the edge semi-angle approaches 90 degrees.

However, to calculate the pressure resistance that the leading edge suffers it is necessary to take into account the surface length and the surface angle. It is plotted at Figure 4.64, where it can be seen that the resistance approaches zero when the angle also does, an expected behaviour.

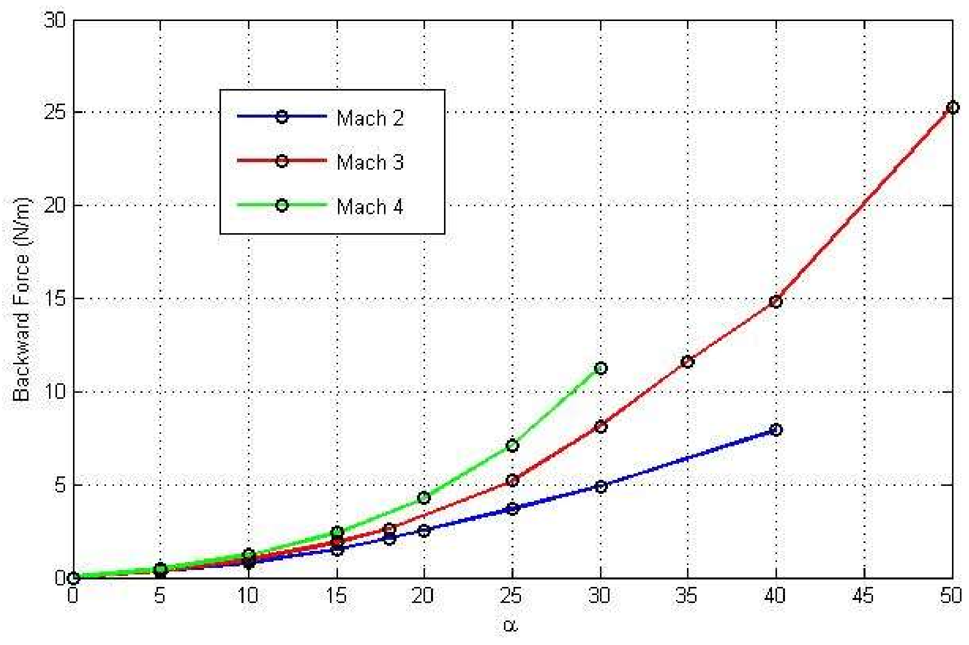


Figure 4.64. Backward pressure resistance force for leading edge, dependency with Mach number and leading edge semi-angle.

Also, it is clear that the higher the Mach number the higher the resistance that the leading edge suffers, as well as happens when increasing the angle. However, when calculating the resistance coefficient, the behaviour with α remains, but the one with the Mach number is reversed, because of the increased velocity that takes part in the adimensionalization.

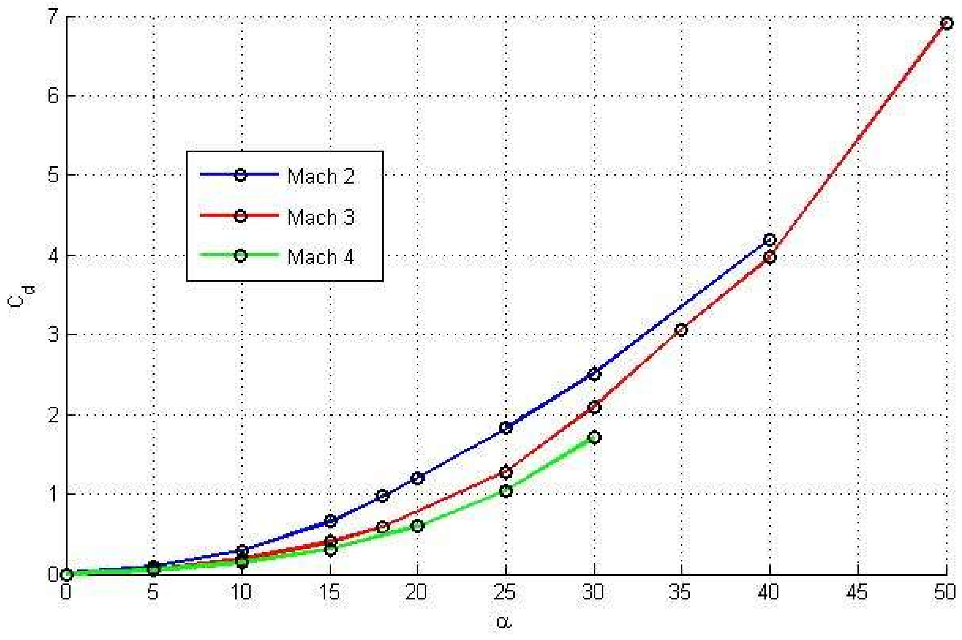


Figure 4.65. Backward pressure resistance force for leading edge, dependency with Mach number and leading edge semi-angle.

4.5.1 Auxiliary functions for the leading edge

Geometry Function

```
function
[x,y,dxi,deta,dxidx,dxidy,detadx,detady,J,dxdxi,dxdeta,dydxi,dydeta] =
Geometry_LeadingEdge(Nxi,Neta,LHOR,LVER)

xi_LHOR = LHOR;
eta_LVER = LVER;

dxi = xi_LHOR/(Nxi-1);
deta = eta_LVER/(Neta-1);

for j=1:Neta
    xi(:,j) = 0:dxi:xi_LHOR;
end
for i=1:Nxi
    eta(i,:) = 0:deta:eta_LVER;
end

% Edge X direction beginning
s = floor(0.5*Nxi);
xis = LHOR*(s-1)/(Nxi-1);

% Edge semi-angle (\alpha)
a = 30*pi/180;
```

```
y0(1:s) = 0;
y0(s+1:Nxi) = ( xi(s+1:Nxi) - xis )*tan(a);

for i=1:Nxi
    for j=1:Neta
        x(i,j) = xi(i,j);
    end
    y(i,:) = linspace(y0(i),LVER,Neta);
end

% Direct metrics DC
dxidx = Dx(xi,x);
dxidy = Dy(xi,y);
detadx = Dx(eta,x);
detady = Dy(eta,y);

% Inverse metrics DC
dxdxi = Dx(x,xi);
dxdeta = Dy(x,eta);
dydeta = Dy(y,eta);
dydxi = Dx(y,xi);

J = dxdxi.*dydeta - dydxi.*dxdeta;
```

Initial Conditions Function

```
function [rho,u,v,T,p] =
Initial_Conditions_LeadingEdge(rhoinf,uinf,Tinf,R,Nxi,Neta)

% Beginning of the edge in the X direction
s = floor(0.5*Nxi);

rho(1:Nxi,1:Neta) = rhoinf;
u(1:Nxi,1:Neta) = uinf;
u(s:Nxi,1) = 0;
v(1:Nxi,1:Neta) = 0;
T(1:Nxi,1:Neta) = Tinf;
p(1:Nxi,1:Neta) = R*rho.*T;
```

Boundary Conditions Function

```
function [rho,u,v,T,p] =
Boundary_Conditions_LeadingEdge(rho,u,v,T,p,Nxi,Neta)

% Beginning of the edge in the X direction
s = floor(0.5*Nxi);

% CC at the right side
rho(Nxi,2:Neta-1) = 2*rho(Nxi-1,2:Neta-1) - rho(Nxi-2,2:Neta-1);
u(Nxi,2:Neta-1) = 2*u(Nxi-1,2:Neta-1) - u(Nxi-2,2:Neta-1);
v(Nxi,2:Neta-1) = 2*v(Nxi-1,2:Neta-1) - v(Nxi-2,2:Neta-1);
T(Nxi,2:Neta-1) = 2*T(Nxi-1,2:Neta-1) - T(Nxi-2,2:Neta-1);
p(Nxi,2:Neta-1) = 2*p(Nxi-1,2:Neta-1) - p(Nxi-2,2:Neta-1);

% CC at the surface
```

```
rho(2:Nxi,1) = 2*rho(2:Nxi,2) - rho(2:Nxi,3);
p(2:Nxi,1) = 2*p(2:Nxi,2) - p(2:Nxi,3);
u(2:s-1,1) = 2*u(2:s-1,2) - u(2:s-1,3);
```

5 Conclusions and Further Developments

Reaching the end of the project, it is safe to say that the overall objectives of the project, which were developing a custom program that could numerically solve supersonic flows for both uniform and non-uniform grids and obtaining those results for a few interesting problems, have been met.

One of the conclusions that can be extracted from the hours of work put into this project is that the numerical stability of the program is a serious problem for the expansion of it. In general, it can be said that the Courant-Friedrichs-Lewy (CFL) criterion (Equation 3.1) is not enough to preserve a convergent behaviour of the numerical solution, specially when sharp edges like the ones that appear in the steps are involved. In those cases, a strong correction made with the safety factor had to be made in order to obtain any solutions. In this project it was considered the addition of numerical artificial viscosity to alleviate this problem, as John D. Anderson discusses in chapter 6.6 of [Ref 1], but finally it was discarded as it would have unnecessarily complicated the project, and in the end was not necessary to obtain the results.

Other conclusion reached is that the computation times can be efficiently optimised using the program structure. One may wonder why the programs showed in chapter 3 are written the way they are, with almost no auxiliary functions that perform small tasks like the differentiation, which could have avoided some repetitions, also minimising the probability of human error and making easier the creation of the program. Well the truth is that at the beginning they were made this way. However, when any function was called it had to make a copy of the variables it worked with, and then copy them back once it finished. This made the original programs painfully slow, and with successive testing and improvements the computation time was reduced in some cases to even a half of the original time.

Talking about the results I am very happy to see that they correspond with reasonable accuracy with the external results obtained by other authors or with the analytical solutions they have been compared to. It creates a profound satisfaction when something that has required so much time and effort finally works, and the solutions obtained are exactly what they were supposed to be, like what happened when comparing the results of the flat plate. Because it has not been an easy task. I would say that this project is the single work that I have invested more time into of all my life.

When developing the programs and the project overall, a lot of focus has been put into explaining how the things work piece by piece, and making everything, specially the programs, as easy to understand and manage as possible. The objective of this is to make it easier for a future student to take this project as basis and expand the work with other ideas or analysing other problems using the programs developed here. That was one of the key motivations of creating the chapter 3 to explain the programs, and to make them modular, with the information of the specific problems restricted to the auxiliary functions, which are interchangeable.

In the future I would like to see another student using the programs to generate and solve problems that due to time constrains have not been solved here, like the case of a rectangle or the elliptic and rhomboidal profiles. Also, it would be very interesting to investigate the artificial viscosity theme, or other similar method of stabilising the flow around sharp edges, so the numerical stability becomes less of a concern. This may be achieved by introducing turbulent viscosity in the method. The inclusion of chemical and radiating effects for the hypersonic flow in a future version of the program would be very interesting too, and as a final suggestion, the method could be modified to represent an axil-symmetrical flow. This way a non-bidimensional problem could be analysed, in contrast with all the ones done here, without really needing the expansion to three-dimensional space, that although could be easily done in the equations, would require an enormous amount of computational power.

6 References

[1] John D. Anderson Jr., “*Computational Fluid Dynamics, The Basis with Applications”*, McGraw-Hill, United States of America, 1995

[2] Manuel Carreño Ruiz, *“Aplicaciones del método de MacCormack a diversos problemas fluidomecánicos”*, Universidad de Sevilla, Sevilla, 2016

[3] A. B. Ripoll y Miguel Pérez-Saborid, *“Fundamentos y aplicaciones de la mecánica de fluidos”*, McGraw-Hill Interamericana, Madrid, 2005

[4] Pablo José Ruiz Contreras, *“El Método de Colocación para el problema de convección de Rayleigh-Bénard”*, Universidad de Sevilla, Sevilla, 2013

[5] Ángel Ruy-Díaz Rojas, *“Cálculo de Capas Límite en Régimen Hipersónico”*, Universidad de Sevilla, Sevilla, 2017

[6] John D. Anderson Jr., *“Hypersonic and High Temperature Gas Dynamics. Second Edition”*, American Institute of Aeronautics and Astronautics, United States of America, 2006

[7] Frederick Abernathy, *“Fundamentals of Boundary Layers”*, National Committee for Fluid Mechanics Films, Harvard University, 1968

[8] Robert W. MacCormack, *“Numerical Computation of Compressible and Viscous Flow”*, American Institute of Aeronautics and Astronautics, United States of America, 2014