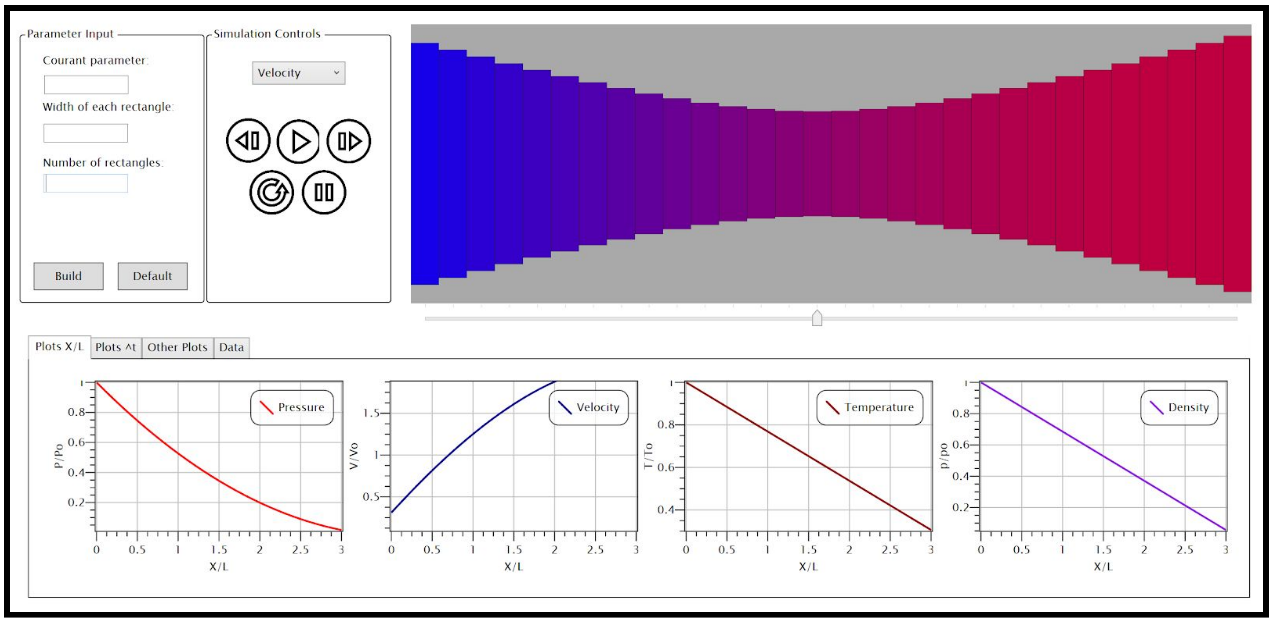
Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

SIMULATION



TITLE: Steady isentropic flow through a convergent-divergent nozzle
VERSION: 1.0
AUTHORS: Laura Parga, Samuel Picón, Marc Vila
DATE: December 20th, 2019

Title: Steady isentropic flow through a convergent-divergent nozzle
Version: 1.0
Authors: Laura Parga, Samuel Picón, Marc Vila
Date: December 20th, 2019

ABSTRACT

Nozzles have a multitude of utilities in an airplane, since they are part of a large part of systems as, for example, the ones used in the fuel system, in the engine and/or in the air conditioning system, among other ones. The design of these systems contribute to the aeronautical surveillance and, thus, is needed the best possible efficiency.

The use of simulations is widely common in the aeronautical field because an accurate simulation can reduce the costs in the nozzles manufacture, so that is why the realization of simulation programs are essential and necessary, which has been the main reason to choose this project.

By performing simulations it is possible to obtain numerical values so similar to the real ones, as well as the most relevant graphs. The main idea of this project is to program an application in which the user can change the simulation and shape of the nozzle, to be able of performing different designs, analyzing them and deciding which is the optimal one. Our objective is to simulate the flow inside a nozzle, which means to predict the behaviour (change on the properties of the flow: temperature, density, speed and pressure, in time and position). Developing a nozzle has great associated difficulties, so this simulation intends to help the choice of the nozzle geometry and specific characteristics.

A program has been developed in which through the application of finite elements and from some initial conditions it is possible to predict its behavior. In order to check the validity of the simulation results that the program performs, the graphs and values shown in *Anderson, "Computational Fluid Dynamics"* has been used.

After analysing and comparing the results, seeing the small difference with the ones on the book, we can conclude that the simulator has a correct operation and does his job.

Título: Flujo estacionario a través de una tobera convergente-divergente
Versión: 1.0
Autores: Laura Parga, Samuel Picón, Marc Vila
Fecha: 20 de Diciembre, 2019

RESUMEN

Las toberas tienen gran multitud de utilidades en las aeronaves, ya que son parte de gran cantidad de sistemas como, por ejemplo, las utilizadas en el sistema de combustible, en el motor i/o sistema de aire acondicionado, entre otras. El diseño de estos sistemas contribuye en gran parte a la seguridad aeronáutica y, por lo tanto, hace falta que sea lo más eficiente posible.

El uso de simulaciones es común en el ámbito aeronáutico, ya que cuanto más exactas sean estas simulaciones, mayor reducción de los costes será posible en la fabricación de toberas y/o mejora de las mismas, de manera que la realización de este tipo de programas es esencial y necesaria, razón que justifica principalmente la elección de este proyecto.

Mediante la realización de simulaciones es posible obtener valores numéricos aproximados, así como las gráficas pertinentes. La idea del proyecto es programar una aplicación con la que el usuario pueda variar los parámetros de simulación y forma de la tobera, por tal de poder hacer diferentes diseños que se acerquen a la realidad, analizarlos y decidir el más óptimo. Nuestro objetivo es simular el flujo en el interior de la tobera, es decir, predecir su comportamiento (variación de las sus características: temperatura, densidad, velocidad y presión, en tiempo y posición). Desarrollar una tobera tiene grandes dificultades asociadas, de manera que esta simulación pretende ayudar a escoger su geometría y características específicas.

Se ha desarrollado un programa en el cual mediante la aplicación de elementos finitos y partiendo de unas condiciones iniciales es posible predecir la variación en el tiempo de los parámetros mencionados con anterioridad. Por tal de comprobar la validez de los resultados de la simulación que realiza el programa, se han utilizado las gráficas y los valores mostrados en el *Anderson, "Computational Fluid Dynamics"*.

Después del análisis y la comparación de resultados, viendo que hay muy poca diferencia con los mostrados en el libro, podemos concluir que el simulador tiene un buen funcionamiento y cumple su objetivo.

Títol: Flux estacionari a través d'una tovera convergent-divergent
Versió: 1.0
Autors: Laura Parga, Samuel Picón, Marc Vila
Data: 20 de Decembre, 2019

RESUMEN

Les toveres tenen gran multitud d'utilitats a les aeronaus, ja que són part de gran quantitat de sistemes aviònics com, per exemple, les utilitzades en el sistema de combustible, en el motor i/o sistema d'aire condicionat, entre d'altres. El disseny d'aquests contribueix, per tant, a la seguretat aeronàutica i, per això, cal que sigui el més eficient possible.

La utilització de simulacions és comú en l'àmbit aeronàutic, ja que com més exactes siguin les simulacions, major reducció de costos serà permesa en la fabricació de toveres i/o millora de les mateixes, de manera que la realització d'aquest tipus de programes és essencial i necessària, raó per la qual s'ha escollit realitzar aquest projecte.

Mitjançant la realització de simulacions és possible obtenir valors numèrics molt semblants als reals, així com les gràfiques pertinents. La idea d'aquest projecte és programar una aplicació amb què l'usuari pugui variar els paràmetres de simulació i forma de la tovera, per tal de poder fer diferents dissenys que s'apropin a la realitat, analitzar-los i decidir el més òptim. El nostre objectiu és simular el flux a l'interior de la tovera, és a dir, predir el seu comportament (variació de les característiques del flux: temperatura, densitat, velocitat i pressió, en temps i posició). Desenvolupar una tovera té grans dificultats associades, de manera que aquesta simulació pretén ajudar a triar la seva geometria i característiques específiques.

S'ha desenvolupat un programa en el qual, mitjançant l'aplicació d'elements finits i partint d'unes condicions inicials, és possible predir la variació dels paràmetres esmentats anteriorment. Per tal de comprovar la validesa dels resultats de la simulació que realitza el programa, s'han utilitzat les gràfiques i els valors mostrats a l'*Anderson, "Computational Fluid Dynamics"*.

Després de l'anàlisi i comparació dels resultats, veient que hi ha molt poca diferència amb els mostrats al llibre, podem concluir que el simulador té un bon funcionament i compleix la seva funció.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

A nozzle could be defined as a pipe with a varying cross-section area, which can determine the way of direct or modify the flow. This device is frequently used to control the direction and/or the characteristics of a fluid flow such as the rate or the shape of flow, the speed, mass, and the pressure.

The main objectives of a nozzle are:

- To convert pressure and thermal energy into kinetic energy
- To direct the fluid jet at a specific angle (nozzle angle)

Along this section, characteristics and nozzle types will be described, in order to explain what have been chosen for the project and the reasons. Also nozzle applications will be explained.

1.1 Characteristics of the nozzle

A nozzle, is a device capable of transforming the internal energy of a fluid into a propulsion, which can be used to accelerate objects, or to perform other functions, such as lubricating elements, spraying fluids, or even spray fuel quantities, among many other functions.

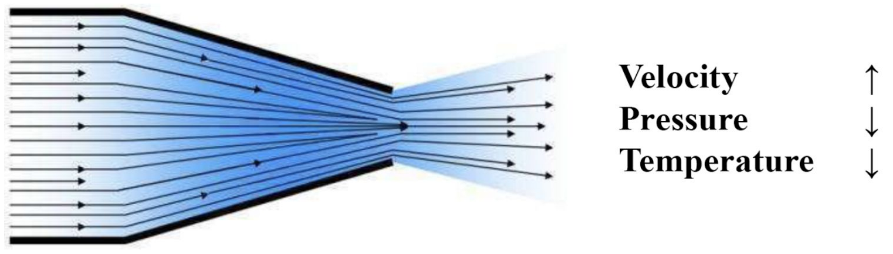
Every nozzle could be defined as defined as a tube in which the section varies with respect to the length, what will really define the type of nozzle is the use and purpose of it, as well as the type of variation of the area with respect to the length, this will give certain characteristics to the nozzle that will define it.

In our specific case a subsonic-supersonic isentropic nozzle which is used for propulse aircrafts will be studied as our project purpose.

1.2 Types of nozzle

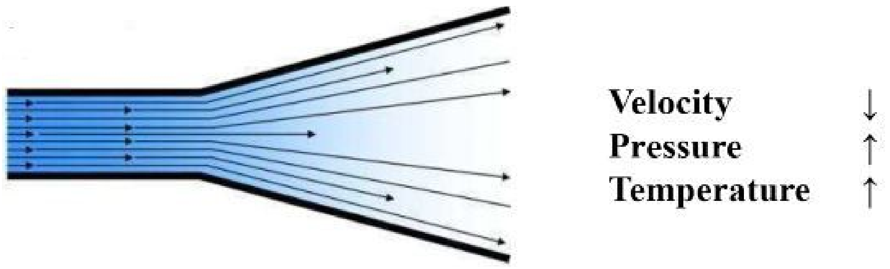
Different nozzle types will be explained below, most of them used in aviation but with different functionalities within it, since as previously has been explained, these are used in refrigeration, lubrication, combustion and propulsion systems.

- **Hydro jet or gas jet:** Type of nozzles that send a liquid or gas to a surrounding medium at a certain speed causing a flow. It is the example of gas cookers, which have a continuous flow of gas which is ignited at its outlet to generate a continuous flame due to a gas jet, nowadays are also used for water movement in fountains, creation of air currents among other uses.
- **High velocity:** The objective of a nozzle is to increase the kinetic energy of the flowing medium. It is achieved that the interior fluid accelerates flows along it. Regarding to the shape, they can be classified into three types: convergent, divergent and convergent-divergent nozzles, changing each one of them the properties of the fluid in a different way.
    - Convergent nozzle: This type of nozzle begins being large and gets smaller, by decreasing the area of the nozzle progressively. They are useful to accelerate subsonic flows.



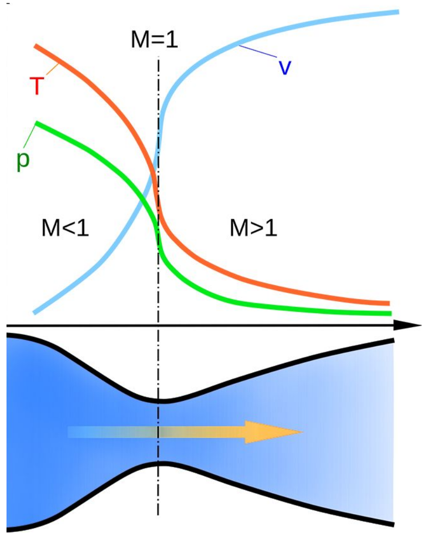
1.2.1. *Convergent nozzle scheme*

    - Divergent nozzle: It could be defined as the opposite case to the previous one, in which there is a widening of the nozzle with which the speed decreases at the end of it. They are useful to accelerate supersonic flows.



1.2.2. *Divergent nozzle scheme*

- <u>Convergent-divergent nozzle</u>: Also known as laval nozzle. This type of nozzle first converges down to the minimum area, which is called throat, and then it is expanded through the divergent section to the exit. Are useful to accelerate the flow from a subsonic to a supersonic one, so they are the type of nozzle that most increase the velocity of the stream.



1.2.3. *Convergent-divergent nozzle*

As figure 1.2.3. shows, the flow inside this type of nozzle is varying the Mach number and so the type of flow is changing. As we have said, the flow enters the nozzle being subsonic and it accelerates until it reaches $M = 1$ in the throat, but it goes on increasing the speed, so the exit flow is supersonic. Meanwhile, both pressure and temperature at the stream keep decreasing continuously (in a different way).

- **Propelling:** Use the internal energy of the gas to convert this into propulsion, Depending on the nozzle type, the gas can provide a propulsion to reach supersonic forces, such as rocket engines which maximize thrust and escape velocity by using convergent-divergent nozzles with very large area ratios.

- **Magnetic:** A magnetic nozzle, is a device that creates a magnetic field in order to expand and accelerate plasma in order to achieve a propulsion. The operation is similar to traditional (convergent-divergent) nozzles with the difference that the internal plasma energy is immediately converted into kinetic energy and the electric charges play an important role instead of the pressure in order to cause propulsion.

- **Spray:** There are different types of spray nozzles, one of these would be the atomizers, which are used for example for fuel injection in internal combustion engines, another type are swirl nozzles, which insert the fluid tangentially in order to increase the speed of the fluid that circulates due to a circular flow is generated inside the nozzle.

During this project we will simulate with a convergent-divergent nozzle because of the efficient engine operation over a wider airflow range it provides.

1.3 Applications

Nozzles have multitude of problems, some of them are related with high temperatures, so different materials must be used to protect the nozzle.

The application can be used to know the approximate temperature at different nozzle point, is really useful to know the exactly points with a higher temperature with the goal of beef up these parts and avoid possible damages.

Also starting off with some initial conditions at an initial point, such as, speed, pressure, density and temperature, evolution of the different points of the nozzle can be predicted.

New materials are constantly being developed, this kind of programs can help to know characteristics necessary for the development of materials that can withstand high temperatures inside the turbine and also therefore the nozzle.

Corrosion as well as areas where it could occur, can be predicted, it is true that this is more complicated, but many of the variables analyzed in this project and with the realization of the relevant simulation could give us an idea of the nozzle areas where there is a greater probability of its occurrence or improve areas that are also susceptible to this problem, as well as knowing the approximate useful life of each part of the nozzle and being able to get an idea of how often it will be necessary to perform maintenance or change the corresponding parts, avoiding possible complications.

An airplane has a large number of different nozzle types, each of these has a different purpose; either for the refrigeration, combustion system, and many other uses, it is therefore of great importance the study, in our case a subsonic-supersonic nozzle is studied, this is very useful to improve the nozzles and develop new systems.

2. EQUATIONS

We want to simulate the evolution, in time and in the horizontal axis of a nozzle, of four properties of the fluid crossing the nozzle (temperature, density, pressure and velocity). So, we need four equations and we need also to discretize them in order to do the computation of future states during the simulation.

In order to do that, we will use the MacCormack's method, in which we will use time and position explicit finite-difference technique for the equations discretizations. Since this numerical method is a method with an accuracy of first order, we need to use a predictor-corrector philosophy in order to improve the accuracy.

As all numerical methods, the fact that it uses points of the surrounding, we need to add points outside the nozzle in order to have references to simulate. In this case, where the fluid will be assumed one-dimensional, the nozzle will be discretized using some points in the axis where the flow goes through and so, we will add one point at the inflow and one point at the outflow, which will determine the properties of the fluid entering the nozzle and at the exit of the nozzle.

2.1 Physical equations

All fluids are governed by the conservation equations of the mass, the momentum and the energy. For the sake of simplicity, we will use one-dimensional fluid and an ideal gas. Adjusting these three conservations equations under the mentioned conditions, we have the four equations we will use:

Mass conservation: $A\frac{\partial\rho}{\partial t} + \frac{\partial}{\partial t}(\rho A V) = 0$

Momentum conservation: $-\frac{\partial p}{\partial x} = \rho\frac{\partial V}{\partial t} + \rho V\frac{\partial V}{\partial x}$

Energy conservation: $-p\frac{\partial V}{\partial x} - pV\frac{\partial[ln(A)]}{\partial x} = \rho\frac{\partial e}{\partial t} + \rho V\frac{\partial e}{\partial x}$

Ideal gas law: $p = \rho R T$

Considering the nozzle is circular in the z axis, the area (A in the previous equations) can be calculated as: $A = \pi r^2$, where the radius of the circle is a half of the height of the nozzle.

Assuming a calorically perfect gas, we can use the next relation between the energy and the temperature of the gas: $e = C_v T$, where C_v is the specific heat at constant volume, a constant that measures the amount of heat that a gas can release or absorb during a change on temperature keeping constant the volume. This constant can be expressed as $C_v = \frac{R}{\gamma-1}$ in the case of an ideal gas, where γ is the heat capacity ratio (relation between C_p and C_v), that depends on the gas.

Including the ideal gas law inside the conservation equations in such a way that the pressure is removed from these equations, we get a system with three unknowns (temperature, velocity and density) and three equations (mass, momentum and energy conservation). So, the pressure won't be in the discretized equations but can always be computed using the ideal gas law if we know these other properties.

2.2 Dimensionless variables

Although the dimensions can give us feelings about the magnitude of each one of the properties of the flow while simulating, we have expressed all equations in terms of non dimensional variables. The main benefit of simulating this way is that the results are the same, regardless the flow around the nozzle because they are computed with respect to this outer flow.

In order to differentiate the variables, we will call all dimensionless values with an apostrophe (B') and the reservoir ones with a subindex 0 (B_0). This lasts mentioned values are the ones of referents so, as we have said before, the ones of the outer flow of the nozzle.

The reference values will be the ones corresponding to the temperature, the pressure and the density, respectively: T_0, p_0 and ρ_0. So, the dimensionless values of these properties are easy to compute by dividing: $T' = \frac{T}{T_0}$, $p' = \frac{p}{p_0}$ and $\rho' = \frac{\rho}{\rho_0}$. We will also take as a reference the speed of sound in the reservoir: $a_0 = \sqrt{\gamma R T_0}$ and so the dimensionless velocity of the fluid is also easy to deduce by: $V' = \frac{V}{a_0}$. Also the dimensionless speed of sound will be useful during the simulation, so it can be obtained by: $a' = \frac{a}{a_0} = \frac{\sqrt{\gamma R T}}{\sqrt{\gamma R T_0}} = \sqrt{\frac{T}{T_0}} = \sqrt{T'}$.

A part from this dimensionless variables, which comes directly from the reservoir values, we need a measure of distance in order to get the dimensionless axis of the nozzle, which will also be useful in the dimensionless time (because of the dimensions). This measure of distance is the simplest one we have: the length of the nozzle (L). So, the dimensionless horizontal axis, which goes from the inflow to the outflow, of the nozzle will be $x' = \frac{x}{L}$ (the vertical one will not be necessary). Although it could be used also in the dimensionless area of the nozzle we will not use it. Instead of we will use the area of the nozzle in the throat (A^*), so that in the throat the dimensionless area is equal to the unity: $A' = \frac{A}{A^*}$.

As we have said before, this dimension of distance is useful to get dimensions of time by dividing by velocity this distance. So the dimensionless time is computed as $t' = \frac{t}{L/a_0}$.

From now on, all equations are dimensionless, although they won't have the apostrophe as we have done during this section.

2.3 MacCormack's technique

As we have said before, this technique uses a finite-difference method, represented in the following general equation: $\frac{\partial B}{\partial b} = \frac{B(b+\Delta b)-B(b)}{\Delta b} + O(\Delta b)$, where we would be discretizing the property of the fluid (B) with respect to b. In our case, B will be the density, the temperature and the velocity of the fluid and b will be the time and the x component (horizontal component of the nozzle).

In order to discretize the physical equations, we have to use the following change on variables: $x = x_0 + i\ \Delta x$ and $t = t_0 + j\ \Delta t$, so that $B_{i,j}$ is the property of the fluid B in the time t and in a position x, while B is the property of the fluid B in the same position as before (x), but a Δt later than before.

From the conservation equations, using the mentioned changes on variables in order to discretize, we can set up an explicit finite-difference solution in the horizontal component for the quasi-one-dimensional nozzle flow given by the Mac-Cormack technique from the time derivatives isolated from the conservation equations of mass, energy and momentum.

$$(\tfrac{\partial \rho}{\partial t})_{i,j} = \tfrac{-1}{\Delta x}[\rho_{i,j}(V_{i+1,j} - V_{i,j}) + \rho_{i,j}\ V_{i,j}\ ln(\tfrac{A_{i+1,j}}{A_{i,j}}) + V_{i,j}\ (\rho_{i+1,j} - \rho_{i,j})]$$

$$(\tfrac{\partial V}{\partial t})_{i,j} = \tfrac{-1}{\Delta x}[V_{i,j}(V_{i+1,j} - V_{i,j}) + \tfrac{T_{i+1,j}-T_{i,j}}{\gamma} + \tfrac{T_{i,j}}{\gamma\ \rho_{i,j}}(\rho_{i+1,j} - \rho_{i,j})]$$

$$(\tfrac{\partial T}{\partial t})_{i,j} = \tfrac{-1}{\Delta x}[V_{i,j}\ (T_{i+1,j} - T_{i,j}) + T_{i,j}\ (\gamma - 1)\ \{V_{i+1,j} - V_{i,j} + V_{i,j}\ ln(\tfrac{A_{i+1,j}}{A_{i,j}})\}]$$

Now, with these previous equations and the same explicit method in time, we can get the predicted values of the fluid properties, which will be identified with a bar.

$$\bar{\rho}_{i,j+1} = \rho_{i,j} + (\tfrac{\partial \rho}{\partial t})_{i,j}\ \Delta t$$

$$\bar{V}_{i,j+1} = V_{i,j} + (\tfrac{\partial V}{\partial t})_{i,j}\ \Delta t$$

$$\bar{T}_{i,j+1} = T_{i,j} + (\tfrac{\partial T}{\partial t})_{i,j}\ \Delta t$$

2.4 Predictor-Corrector method

As we have said, these obtained values are predicted because they do not have an accuracy of second order, so we have to improve the accuracy using a predictor-corrector method, which will compute the final value.

It uses the same general equation we mentioned from the MacCormack's technique with an explicit finite-difference, but using an average derivative, which corrects the predicted values of the used method. It is called like this because this derivative comes from doing the average of two derivatives, following the equation:

$$(\tfrac{\partial B}{\partial t})_{ave} = \tfrac{1}{2}[(\tfrac{\partial B}{\partial t})_{i,j} + (\tfrac{\partial \overline{B}}{\partial t})_{i,j+1}]$$

where B will be each one of the properties of the nozzle flow.

In this equation we are considering the time derivative of the physical equations in an instant t and the time derivative of the MacCormack's equations in the next time interval, $t+\Delta t$. The ones on time t, are the time derivative equations we get in the previous section and the ones on time $t+\Delta t$ are the same time derivatives but with the values we get doing MacCormack's technique. There is one difference in the position derivatives of this last ones equations, they discretize with the previous point of the nozzle: $\frac{\partial B}{\partial x} = \frac{B(x)-B(x-\Delta x)}{\Delta x}$.

So, finally we can get the equations:

$$\rho(t+\Delta t) = \rho(t) - \tfrac{\Delta x}{2\,\Delta t}\{[\rho_{i,j}\,(V_{i+1,j} - V_{i,j}) + \rho_{i,j}\,V_{i,j}\,ln(\tfrac{A_{i+1,j}}{A_{i,j}}) + V_{i,j}\,(\rho_{i+1,j} - \rho_{i,j})] +$$
$$+ [\rho_{i,j+1}\,(V_{i,j+1} - V_{i-1,j+1}) + \rho_{i,j+1}\,V_{i,j+1}\,ln(\tfrac{A_{i,j+1}}{A_{i-1,j+1}}) + V_{i,j+1}\,(\rho_{i,j+1} - \rho_{i-1,j+1})]\}$$

$$V(t+\Delta t) = V(t) - \tfrac{\Delta x}{2\,\Delta t}\{[V_{i,j}(V_{i+1,j} - V_{i,j}) + \frac{T_{i+1,j}-T_{i,j}}{\gamma} + \frac{T_{i,j}}{\gamma\,\rho_{i,j}}(\rho_{i+1,j} - \rho_{i,j})] +$$
$$+ [V_{i,j+1}(V_{i,j+1} - V_{i-1,j+1}) + \frac{T_{i,j+1}-T_{i-1,j+1}}{\gamma} + \frac{T_{i,j+1}}{\gamma\,\rho_{i,j+1}}(\rho_{i,j+1} - \rho_{i-1,j+1})]\}$$

$$T(t+\Delta t) = T(t) - \tfrac{\Delta x}{2\,\Delta t}\{[V_{i,j}\,(T_{i+1,j} - T_{i,j}) + T_{i,j}\,(\gamma-1)\,\{V_{i+1,j} - V_{i,j} +$$
$$+ V_{i,j}\,ln(\tfrac{A_{i+1,j}}{A_{i,j}})\}] + [V_{i,j+1}\,(T_{i,j+1} - T_{i-1,j+1}) + T_{i,j+1}\,(\gamma-1)\,\{V_{i,j+1} - V_{i-1,j+1} +$$
$$+ V_{i,j+1}\,ln(\tfrac{A_{i,j+1}}{A_{i-1,j+1}})\}]\}$$
$$p_{i,j} = T_{i,j} \cdot \rho_{i,j}$$

In these equations there are two parameters that depends on the discretization, Δx and Δt, and one depending on the fluid, γ. These ones of the discretization will be set by the user application while the parameter depending of the fluid will be, in principle, the one of the air: $\gamma = 1.4$.

2.5 Courant condition

The Courant condition or Courant number is used to calculate the time step needed, which determines the simple stability of the system, which is determined by a hyperbolic equation which gives us the value needed for Δt, the non dimensional increment of time for which the simulation will be stable and satisfies the Courant condition, which formula is:

$$\Delta t = C\frac{\Delta x}{a+V}$$

Which a (local speed of sound) and V (local velocity) are given in each Δx along the nozzle and C (Courant number) is a user input value which has to be $C \leq 1$ in order for the system to be stable.

If we discretize the equation for each Δx along the nozzle we finally have:

$$(\Delta t)_i^t = C\frac{\Delta x}{(a)_i^t + (V)_i^t}$$; one single Δt value for each increment of length.

We have two options in order to compute the value we need: the first option is to calculate each Δt locally and use the corresponding local value in each of the positions of the nozzle; and the second option is to compute all of the values of each increment and choosing the minimum value and then using the value of Δt obtained to compute the equations at each increment of Δx, which would be:

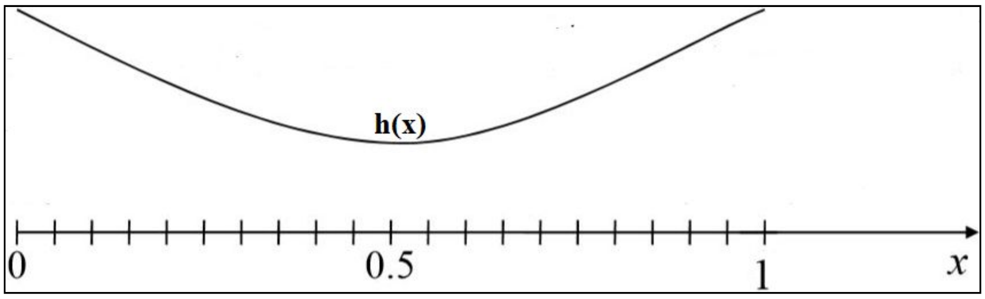
$$\Delta t = Min\{\ (\Delta t)_1^t, (\Delta t)_2^t, \ldots, (\Delta t)_i^t, \ldots\ldots (\Delta t)_N^t\ \}$$

The first option at first glance seems like the correct option although increases the computation load of the simulator since has to calculate a whole set of $(\Delta t)_i^t$ each iterations, which if we had a high number N would make the programm be very slow. Not only that but also if we think about it, each rectangle would have a different Δt increment, which means that the grid points would not be calculated at the same physical time as the neighbouring grid points, resulting in a non accurate and non realistic method to obtain an accurate solution. The other option on the other hand allows us to calculate each grid point at the same exact physical time thus allowing us to obtain a more realistic solution which will allow us to obtain more accurate results. That being said we will use the second option going forward to calculate Δt.

2.6 Nozzle shape

As it has already been explained, in this project the nozzle is a convergent-divergent one, so the shape of the axis of the flow direction first decreases and later increases on height. Regarding to the perpendicular plane, the nozzle is considered circular and as the shape is fixed (it does not change on time), the area in each position of this mentioned horizontal axis is given by a function $A = A(x) = \pi\left(\frac{h(x)}{2}\right)^2$, where $h(x)$ is the height of the nozzle in position x, as figure 2.6.1 shows.

We will make this function symmetric also with respect to the x axis in order to get the nozzle shape.



2.6.1. *Height of the nozzle function*

The equation that gives the area will be:

$$A(x) = 1 + 2.2\,(x - 1.5)^2$$

From this equation, the height of the nozzle in each position can be isolated by using the general equation of the area of a circle. Also in this equation can be seen that the throat of the nozzle, that corresponds to the minimum value of $h(x)$, is always situated at $x = 1.5$ because is where the area gets the unitary value and so the flow will be sonic $(M = 1)$ In the previous positions to the throat $(x < 1.5)$, where the nozzle is convergent, the flow is subsonic $(M < 1)$; while in the next positions $(x > 1.5)$, where the nozzle is divergent, the flow is supersonic $(M > 1)$.

2.7 Boundary conditions

Any numerical method uses the surrounding of a given point to apply an equation for this point. As we have created a nozzle with a list of points in an horizontal axis, we need to add two more points: at the inflow and at the outflow, which will be useful to for the first and last points of the nozzle, respectively.

On one hand, the inflow point, it will be assumed that the density and the temperature will be the ones of the reservoir and the velocity will be computed by a linear extrapolation of the first two points of the nozzle. So, density and temperature are unitary while the velocity is computed by: $V_{1,0} = 2\,V_{2,0} - V_{3,0}$. On the other hand, all properties of the outflow boundary point will be computed by a linear extrapolation of the last two points of the nozzle: $B_{N,0} = 2\,B_{N-1,0} - B_{N-2,0}$, where B could be any of the properties: ρ, T, V.

2.8 Initial conditions

In order to start the simulation we need to stipulate an initial state, which means some values of the properties $(\rho,\ T,\ V)$ before simulating. These values could be established arbitrary, but the best way to do it is to approach this initial state to a real one.

Again for the sake of simplicity, we assume linear decrease of these properties with position. As the flow expands, density and temperature decrease and velocity increases, but this last one also depends on temperature. Taking it into account, we have set up the next equations for the initial state:

$$\rho(x,0) = 1 - 0.316\,x$$
$$T(x,0) = 1 - 0.2314\,x$$
$$V(x,0) = (0.1 + 1.09\,x)\,\sqrt{T(x,0)}$$

We have not computed any equation for the pressure because, as it has been explained during the physical equations section, it is always from the ideal gas equation.

3. SIMULATOR

The simulation will consist of predict the behavior of a discretized in one dimension nozzle (the way we have explained in the equations section) when a flow circulates through it.

With the equations that have been already calculated and starting from an initial state, all the properties of a nozzle for a subsequent time can be calculated and consequently displayed graphically.

During this section, the program will be explained, focusing on the code, which will include how does it works and what is it able to do.

3.1 Functionalities

As the objective of the project is to create an application in which the user will be able to create different types of nozzle by changing the shape, the main functionalities our program shall have:

- Parameter box in which the user will be able to choose the starting parameters: number of rectangles of the nozzle, the horizontal axis step and the courant parameter.
- Graphic display showing the shape and values of the different parameters of the nozzle: temperature, density, pressure and speed. The user can choose which one of these parameters colors scales wants to see in the control box and can see the exact value of each one of the properties in one of the tabs.
- Control box allowing the user to play and pause the simulation (also with steps: forward and backward), reset the simulation and change the type of parameters showed in the graphic display.
- Tabs box, which include the visualization of different graphs showing the values of the nozzle parameters along time and the horizontal axis of the nozzle and a table showing the updated exact values of the mentioned parameters. Also it includes other plots related with the mass flow.
- Possibility of saving in a *.txt* file the status of the nozzle and loading a previously saved simulation.
- Comparison of the Anderson book "*Computational Fluid Dynamics*" values with the values obtained from our simulation (with the same parameters) to verify the simulator performance. It also shows a table with the average difference in percentage of each one of the discretization points.

3.2 Code organization

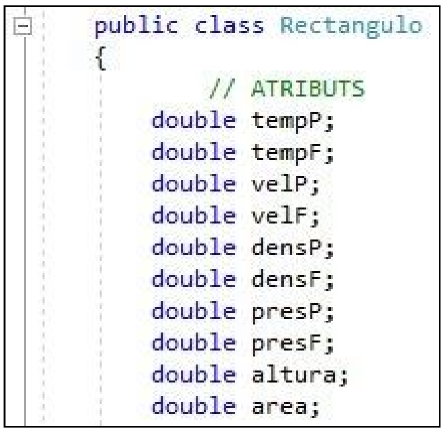
For the programming part we will be using WPF C# with Visual Studio 2019.

The project will be divided into two different libraries: one holding the classes which will calculate the equations; and the other one with the display library, consisting of the WPF windows which will display and manage the informations and methods of the class library.

3.2.1 Classes library

This library contains two classes: one describing a discretized point of the nozzle and another describing the whole nozzle, which means a set of points.

- ***Rectangulo* class:** Represents each *dx* inside the nozzle. Holds all the parameters calculated from the equations, which also holds. Each parameter is represented in an attribute and all of them have a present and a future value, for the calculation of each cycle.

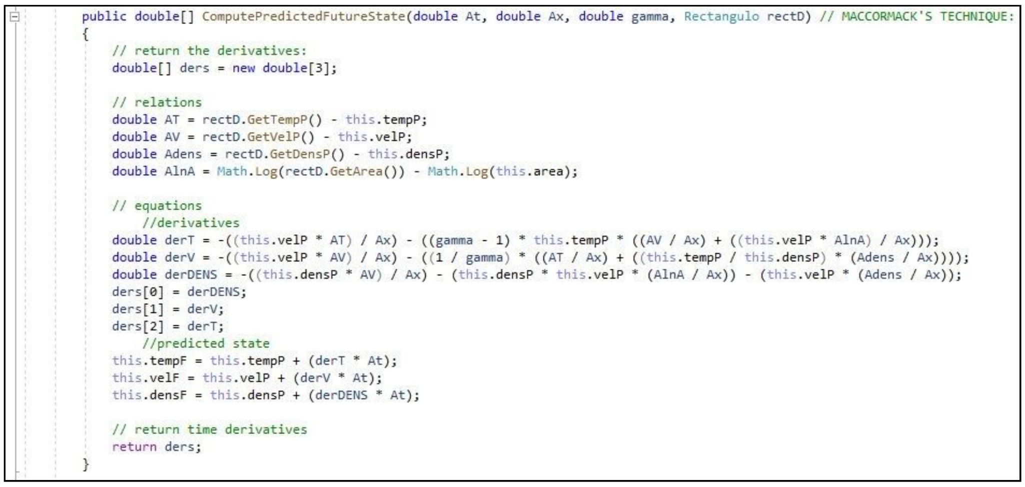


```
public class Rectangulo
{
        // ATRIBUTS
    double tempP;
    double tempF;
    double velP;
    double velF;
    double densP;
    double densF;
    double presP;
    double presF;
    double altura;
    double area;
```

3.2.1.1. *Rectangulo class attributes*

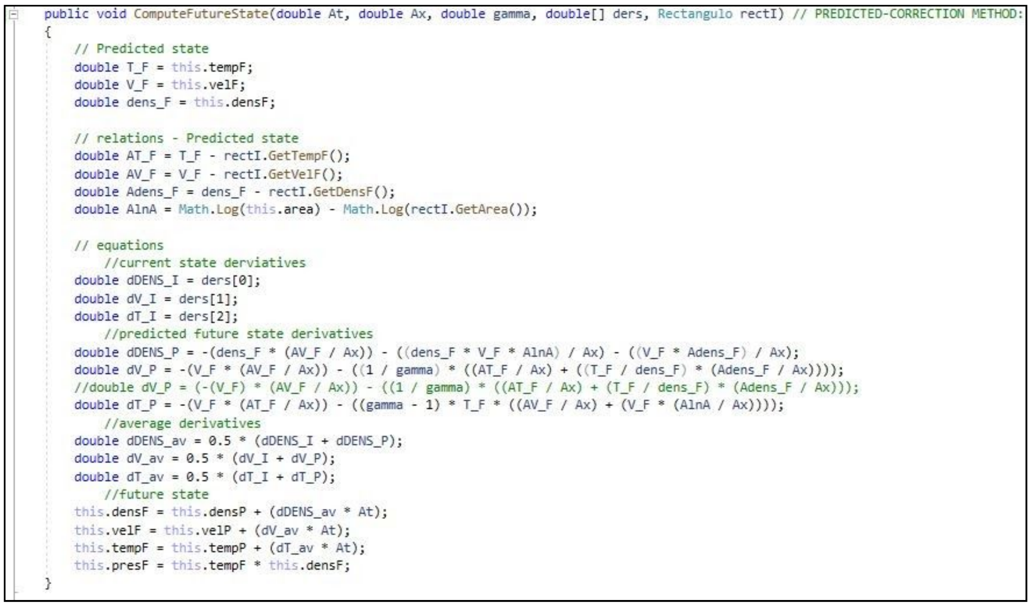
This class contains some different constructors and methods, that help us to optimize the execution of the code. Regarding to the constructors, one of them is empty, which means that creates a *Rectangulo* without any attribute; there is a copy constructor, which receives another *Rectangulo* and copy all the attributes; and finally two more constructors that receive as inputs the properties of the nozzle flow and assign it in the attributes. The difference between these two attributes is the fact that one of these has as input only the properties of the fluid (temperature, density, speed and pressure) and the other one also has the properties of the nozzle (height and area, which are related). Regarding to the methods, it has the typical ones and three more that compute each one of the parameters.

One one hand, these mentioned typical ones are the setters and getters, that are used to get and set each one of the attributes of the class. On the other hand, the method related with the equations are separated into three: the first one compute the predicted future value using MacCormack's technique, the second one corrects these previously computed predicted value using the corrector-predictor method, and the last one finally changes the present values with the obtained future values.



```
public double[] ComputePredictedFutureState(double At, double Ax, double gamma, Rectangulo rectD) // MACCORMACK'S TECHNIQUE:
{
    // return the derivatives:
    double[] ders = new double[3];

    // relations
    double AT = rectD.GetTempP() - this.tempP;
    double AV = rectD.GetVelP() - this.velP;
    double Adens = rectD.GetDensP() - this.densP;
    double AlnA = Math.Log(rectD.GetArea()) - Math.Log(this.area);

    // equations
        //derivatives
    double derT = -((this.velP * AT) / Ax) - ((gamma - 1) * this.tempP * ((AV / Ax) + ((this.velP * AlnA) / Ax)));
    double derV = -((this.velP * AV) / Ax) - ((1 / gamma) * ((AT / Ax) + ((this.tempP / this.densP) * (Adens / Ax))));
    double derDENS = -((this.densP * AV) / Ax) - (this.densP * this.velP * (AlnA / Ax)) - (this.velP * (Adens / Ax));
    ders[0] = derDENS;
    ders[1] = derV;
    ders[2] = derT;
        //predicted state
    this.tempF = this.tempP + (derT * At);
    this.velF = this.velP + (derV * At);
    this.densF = this.densP + (derDENS * At);

    // return time derivatives
    return ders;
}
```

3.2.1.2. *MacCormack's technique applied in the code*

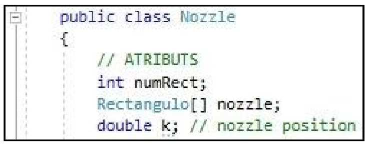


```
public void ComputeFutureState(double At, double Ax, double gamma, double[] ders, Rectangulo rectI) // PREDICTED-CORRECTION METHOD:
{
    // Predicted state
    double T_F = this.tempF;
    double V_F = this.velF;
    double dens_F = this.densF;

    // relations - Predicted state
    double AT_F = T_F - rectI.GetTempF();
    double AV_F = V_F - rectI.GetVelF();
    double Adens_F = dens_F - rectI.GetDensF();
    double AlnA = Math.Log(this.area) - Math.Log(rectI.GetArea());

    // equations
        //current state derviatives
    double dDENS_I = ders[0];
    double dV_I = ders[1];
    double dT_I = ders[2];
        //predicted future state derivatives
    double dDENS_P = -(dens_F * (AV_F / Ax)) - ((dens_F * V_F * AlnA) / Ax) - ((V_F * Adens_F) / Ax);
    double dV_P = -(V_F * (AV_F / Ax)) - ((1 / gamma) * ((AT_F / Ax) + ((T_F / dens_F) * (Adens_F / Ax))));
    //double dV_P = (-(V_F) * (AV_F / Ax)) - ((1 / gamma) * ((AT_F / Ax) + (T_F / dens_F) * (Adens_F / Ax)));
    double dT_P = -(V_F * (AT_F / Ax)) - ((gamma - 1) * T_F * ((AV_F / Ax) + (V_F * (AlnA / Ax))));
        //average derivatives
    double dDENS_av = 0.5 * (dDENS_I + dDENS_P);
    double dV_av = 0.5 * (dV_I + dV_P);
    double dT_av = 0.5 * (dT_I + dT_P);
        //future state
    this.densF = this.densP + (dDENS_av * At);
    this.velF = this.velP + (dV_av * At);
    this.tempF = this.tempP + (dT_av * At);
    this.presF = this.tempF * this.densF;
}
```

3.2.1.3. *Predictor-Corrector method applied in the code*

- ***Nozzle* class:** Represents the mesh in which the *Rectangulo* class objects are stored and how they are related to one another. Gives them an order and a height based on the equation of the nozzle shape.

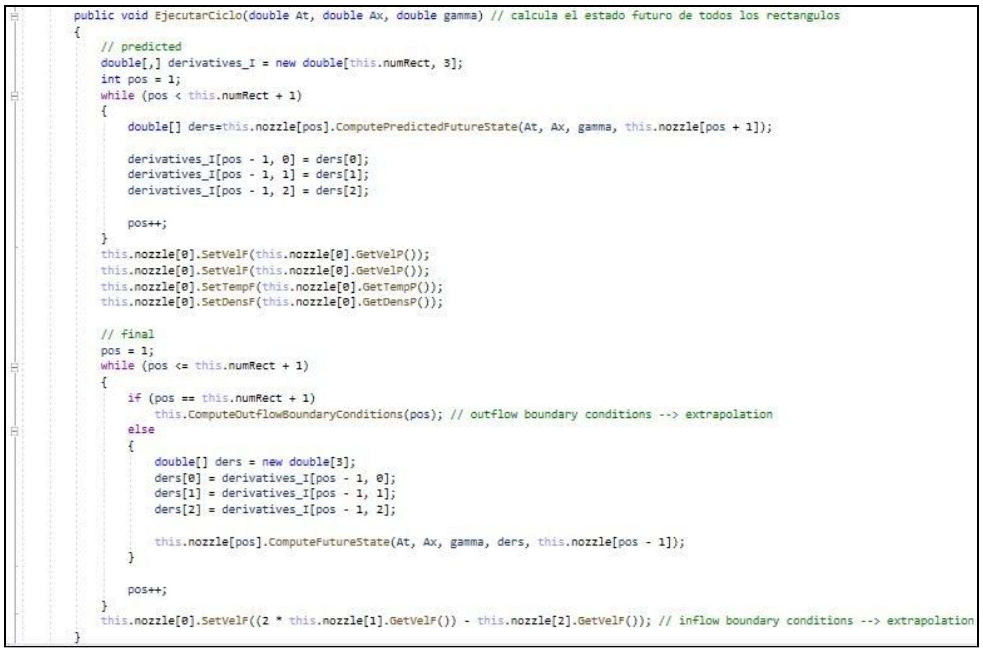
By creating a nozzle we automatically create a vector of *Rectangulos* with a certain number of *Rectangulo*, each one of them with certain height and fluid properties, as we have explained in the equations section. The attributes of this class are the vector and the number of *Rectangulos* and a parameter needed in the advanced study, which will be used to situate the throat.



```
public class Nozzle
{
    // ATRIBUTS
    int numRect;
    Rectangulo[] nozzle;
    double k; // nozzle position
```

3.2.1.4. *Nozzle class attributes*

This class *Nozzle* has also setters, getters (for the number of rectangles, the rectangles vector and for one rectangle in a given position) and other some constructors and other methods. On one hand, the constructors are one empty, one that copy another *Nozzle*, and two that creates the nozzle: one with the throat in the middle (symmetrical nozzle with respect to the horizontal axis) and another one with the throat in a given position (for the advanced study). One the other hand, the methods carry out some tasks that will be needed during the simulation, such as a function that execute one cycle, one that computes the outflow boundary conditions, a function that update the present state (changes from the one set in future), two functions that save or read the state to or from a *.txt* file, another one that gives a table with all the data of the nozzle in the current state and some other that returns list of the properties.



```
public void EjecutarCiclo(double At, double Ax, double gamma) // calcula el estado futuro de todos los rectangulos
{
    // predicted
    double[,] derivatives_I = new double[this.numRect, 3];
    int pos = 1;
    while (pos < this.numRect + 1)
    {
        double[] ders=this.nozzle[pos].ComputePredictedFutureState(At, Ax, gamma, this.nozzle[pos + 1]);

        derivatives_I[pos - 1, 0] = ders[0];
        derivatives_I[pos - 1, 1] = ders[1];
        derivatives_I[pos - 1, 2] = ders[2];

        pos++;
    }
    this.nozzle[0].SetVelF(this.nozzle[0].GetVelP());
    this.nozzle[0].SetVelF(this.nozzle[0].GetVelP());
    this.nozzle[0].SetTempF(this.nozzle[0].GetTempP());
    this.nozzle[0].SetDensF(this.nozzle[0].GetDensP());

    // final
    pos = 1;
    while (pos <= this.numRect + 1)
    {
        if (pos == this.numRect + 1)
            this.ComputeOutflowBoundaryConditions(pos); // outflow boundary conditions --> extrapolation
        else
        {
            double[] ders = new double[3];
            ders[0] = derivatives_I[pos - 1, 0];
            ders[1] = derivatives_I[pos - 1, 1];
            ders[2] = derivatives_I[pos - 1, 2];

            this.nozzle[pos].ComputeFutureState(At, Ax, gamma, ders, this.nozzle[pos - 1]);
        }

        pos++;
    }
    this.nozzle[0].SetVelF((2 * this.nozzle[1].GetVelF()) - this.nozzle[2].GetVelF()); // inflow boundary conditions --> extrapolation
}
```

3.2.1.5. *Cycle execution applied in the code*

3.2.2 Graphic library

Then we have the graphic library or, as we named it, the *Nozzle Display*. This is a library which contains the WPF windows that will use the class library to perform the simulation and show it on screen by using graphic libraries, plots, tables and similar tools.

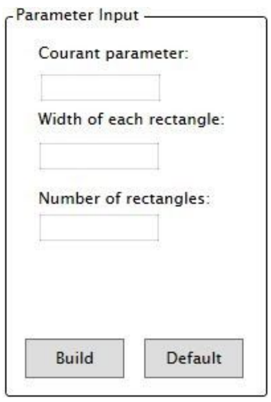
Each one of these windows will be explained in the next sections below.

3.3 Results presentation

The results of our simulator will be presented on the main window of the application, which will be divided into to four different zones, one for the input of the decided user values for the simulation, another for the control of the simulation (play, pause, reset...), one for the graphic display of the nozzle during the simulation (color scale in a graphical representation) and the last zone which is a *Tab Control* with four different tabs.

- **Parameter input**

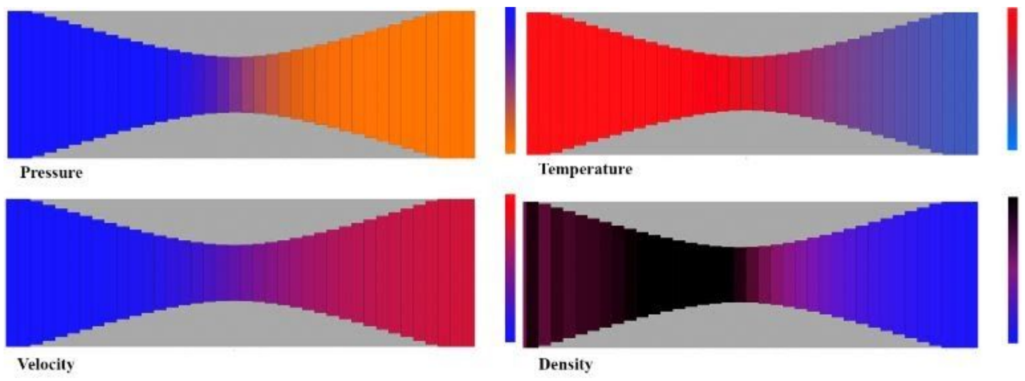
TextBoxes for different simulation parameters: the Courant parameter, the number of divisions of the nozzle and horizontal axis step. As figure 3.3.1. shows, there are two different buttons: the 'Default' writes the parameters that the Anderson book set: the Courant parameter equal to 0.5 and the nozzle formed by 30 rectangles of 0.1 width. The other one reads the values of the *TextBoxes* and builds the nozzle.



3.3.1. *Main parameters to start the simulation*

- **Graphic nozzle**

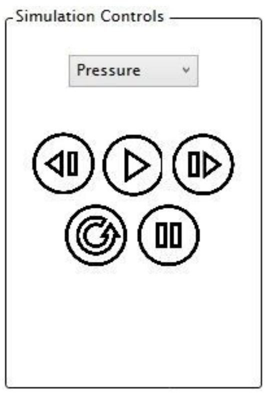
Display of the nozzle graphically with a different color scale for each of the four main parameters being updated which each cycle



3.3.2. *Display of each one of the nozzle parameters*

- **Simulation control**

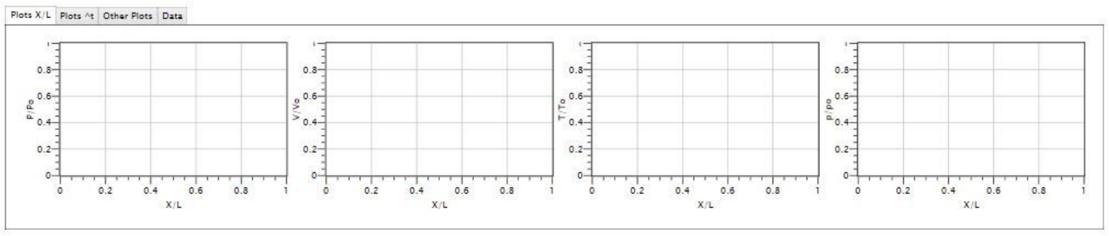
Buttons for the play and pause, step (backward or forward) and reset of the simulation. Also a *ComboBox* for the selection of which parameter the user wants to show in the display.



3.3.3. *Control buttons*

- **Tab Control**

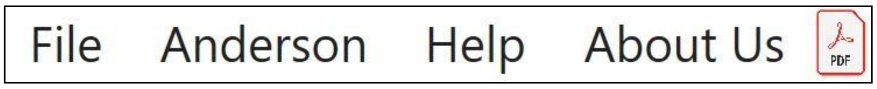
Different tabs which allow the user to navigate between them. Contains the different plots showing the values of the mentioned parameters along the length of the nozzle, the evolution of the parameters during time, values of the parameters on the nozzle during different steps and the updated values of the simulation of every parameter in the form of a Data Grid (similar to how the tables from Anderson are shown).



3.3.4. *Different tab options*

- **Upper menu**

There is also a menu in the upper left side of the main window with four items. The first one of them ('File') allows the user of saving the current status of the nozzle in a *.txt* file and be able to open it whenever he/she wants to do it in order to go on with the simulation. The 'Anderson' one shows the table of results that the Anderson book gives and the table of results obtained from our simulation, in order to check the validity of the code. The next ones are extra: one in order to help the user if he/she does not understand which will include a video explanation of how does the application work, another one with the members of the group that have developed the application and the last one in order to open a *.pdf* report, which helps the user to understand the code.



3.3.5. *Upper menu options*

3.4 Application forms

As we have mentioned, the graphic library of the application has four forms, each one of them accomplishing a different task while running the code.

- **Main Window:** The window in charge of the display of the results and data, control of the simulation and where the access to the other windows will be located, or in other words, the parent window. It is the window that we have been describing in the previous section.
- **Save/Load Window:** The window in which it will be possible to save the current state of the nozzle and load a previously saved state to go on with the simulation.
- **Anderson Window:** This window will allow us to check the validity of our results by comparing them with the ones given by Anderson book in the easiest way possible, using WPF DataGrid templates. It is composed by three different tables: one with the data of the book, another one with the values from out program simulation and the last one with the average difference (in percentage) between the values of each discretization point.
- **Help Window:** Simple window explaining each of the components of the simulation and how does the application work in general, in case of doubt while execution.
- **About us Window:** Window with the team workers of the development of this project.

4. Results

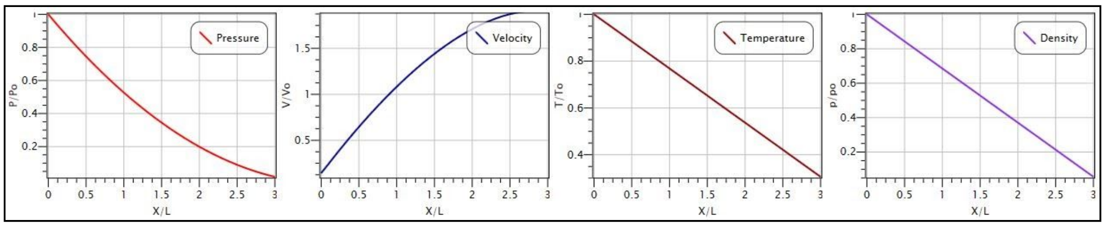
Now, after explaining how this project has been developed, it is time to put everything together and see if the results match with what we expect. As we have mentioned in some sections, we will check the validity of our code by comparing our results with the one performed in *Anderson*: "*Computational Fluid Dynamics*". The chapter 7 of this book describes the same problem we have been developing, equations and results included, but only with certain parameters: a nozzle made of 30 rectangles of width 0.1 and a Courant parameter equal to 0.5.

For this comparison and validation we will use the three different moments of the simulation, which are the three main ones in whom *Anderson* focuses the most: initial conditions of the nozzle, the first step of the simulation and the steady state (point where the simulation values no longer change and the nozzle becomes stationary). For each one of these points we will compare the table values of the Anderson with the one obtained with our simulation program, that way we can assure the simulator will work for any input value we give regardless.

As we will see from the results, the values of the parameters almost match from the values of the Andersons from the most part, except minimal differences between the values, mostly due to the fact that Anderson rounds the values obtained and then uses the values rounded to calculate the iterations.

4.1 Initial Conditions

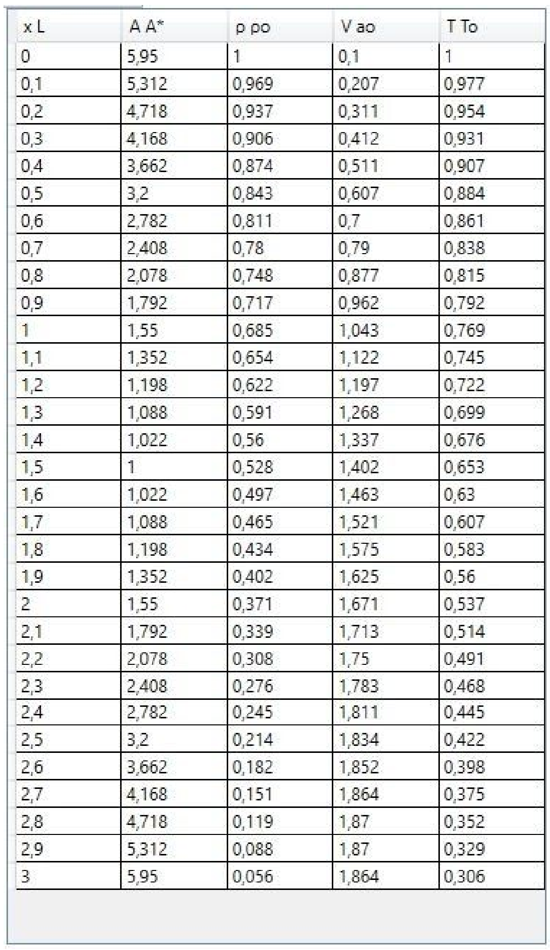
In this case, the application receives as an input the mentioned values that the book uses and generates a nozzle with the methods implemented in the class library of our simulator. The variation of each one of the parameters of the nozzle is represented in the plots of figure 4.1.1.



4.1.1. *Plots of each parameter against X/L*

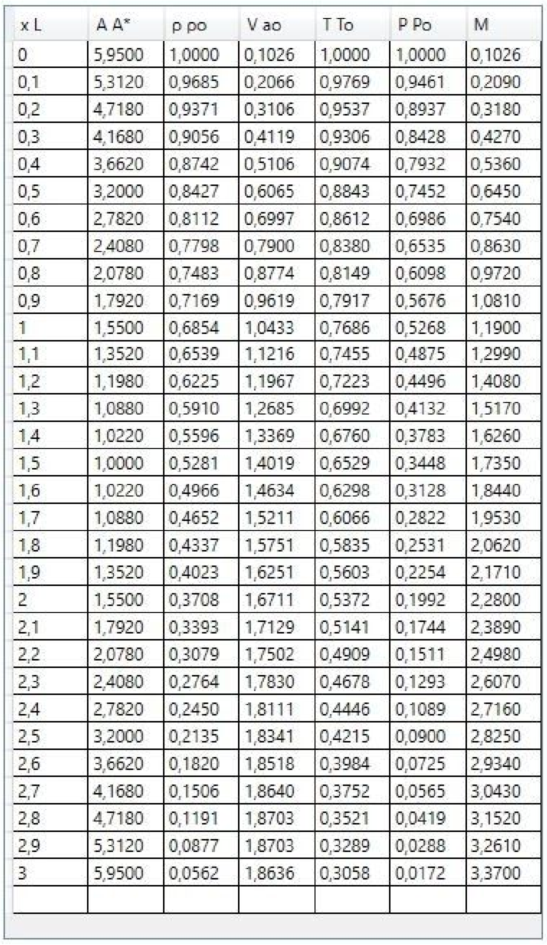
It can be seen that the temperature and the density decrease in a lineal way, as it was expected from the equations we set in the initial conditions. The pressure decreases and is not linear because it comes from the product of the temperature and density. The speed increases as we have said and it is not linear again because of the equation of initial conditions.

In the next two figures, the comparison between the Anderson and our simulation are shown:



x L	A A*	p po	V ao	T To
0	5,95	1	0,1	1
0,1	5,312	0,969	0,207	0,977
0,2	4,718	0,937	0,311	0,954
0,3	4,168	0,906	0,412	0,931
0,4	3,662	0,874	0,511	0,907
0,5	3,2	0,843	0,607	0,884
0,6	2,782	0,811	0,7	0,861
0,7	2,408	0,78	0,79	0,838
0,8	2,078	0,748	0,877	0,815
0,9	1,792	0,717	0,962	0,792
1	1,55	0,685	1,043	0,769
1,1	1,352	0,654	1,122	0,745
1,2	1,198	0,622	1,197	0,722
1,3	1,088	0,591	1,268	0,699
1,4	1,022	0,56	1,337	0,676
1,5	1	0,528	1,402	0,653
1,6	1,022	0,497	1,463	0,63
1,7	1,088	0,465	1,521	0,607
1,8	1,198	0,434	1,575	0,583
1,9	1,352	0,402	1,625	0,56
2	1,55	0,371	1,671	0,537
2,1	1,792	0,339	1,713	0,514
2,2	2,078	0,308	1,75	0,491
2,3	2,408	0,276	1,783	0,468
2,4	2,782	0,245	1,811	0,445
2,5	3,2	0,214	1,834	0,422
2,6	3,662	0,182	1,852	0,398
2,7	4,168	0,151	1,864	0,375
2,8	4,718	0,119	1,87	0,352
2,9	5,312	0,088	1,87	0,329
3	5,95	0,056	1,864	0,306

4.1.2. *Initial state from Anderson*



x L	A A*	p po	V ao	T To	P Po	M
0	5,9500	1,0000	0,1026	1,0000	1,0000	0,1026
0,1	5,3120	0,9685	0,2066	0,9769	0,9461	0,2090
0,2	4,7180	0,9371	0,3106	0,9537	0,8937	0,3180
0,3	4,1680	0,9056	0,4119	0,9306	0,8428	0,4270
0,4	3,6620	0,8742	0,5106	0,9074	0,7932	0,5360
0,5	3,2000	0,8427	0,6065	0,8843	0,7452	0,6450
0,6	2,7820	0,8112	0,6997	0,8612	0,6986	0,7540
0,7	2,4080	0,7798	0,7900	0,8380	0,6535	0,8630
0,8	2,0780	0,7483	0,8774	0,8149	0,6098	0,9720
0,9	1,7920	0,7169	0,9619	0,7917	0,5676	1,0810
1	1,5500	0,6854	1,0433	0,7686	0,5268	1,1900
1,1	1,3520	0,6539	1,1216	0,7455	0,4875	1,2990
1,2	1,1980	0,6225	1,1967	0,7223	0,4496	1,4080
1,3	1,0880	0,5910	1,2685	0,6992	0,4132	1,5170
1,4	1,0220	0,5596	1,3369	0,6760	0,3783	1,6260
1,5	1,0000	0,5281	1,4019	0,6529	0,3448	1,7350
1,6	1,0220	0,4966	1,4634	0,6298	0,3128	1,8440
1,7	1,0880	0,4652	1,5211	0,6066	0,2822	1,9530
1,8	1,1980	0,4337	1,5751	0,5835	0,2531	2,0620
1,9	1,3520	0,4023	1,6251	0,5603	0,2254	2,1710
2	1,5500	0,3708	1,6711	0,5372	0,1992	2,2800
2,1	1,7920	0,3393	1,7129	0,5141	0,1744	2,3890
2,2	2,0780	0,3079	1,7502	0,4909	0,1511	2,4980
2,3	2,4080	0,2764	1,7830	0,4678	0,1293	2,6070
2,4	2,7820	0,2450	1,8111	0,4446	0,1089	2,7160
2,5	3,2000	0,2135	1,8341	0,4215	0,0900	2,8250
2,6	3,6620	0,1820	1,8518	0,3984	0,0725	2,9340
2,7	4,1680	0,1506	1,8640	0,3752	0,0565	3,0430
2,8	4,7180	0,1191	1,8703	0,3521	0,0419	3,1520
2,9	5,3120	0,0877	1,8703	0,3289	0,0288	3,2610
3	5,9500	0,0562	1,8636	0,3058	0,0172	3,3700

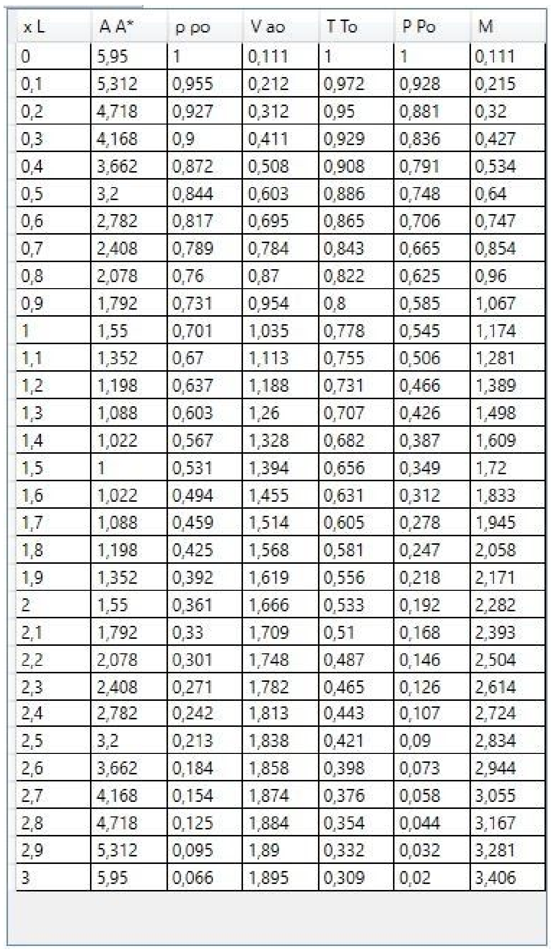
4.1.3. *Initial state from simulation*

There is also a table that shows the computes the percentage of variation between these two tables of results in order to see numerically the difference. The bigger one does not get the 1% of difference, that came only due to the fact of rounding the number in the book, so we can say these results are accurate.

4.2 First Step

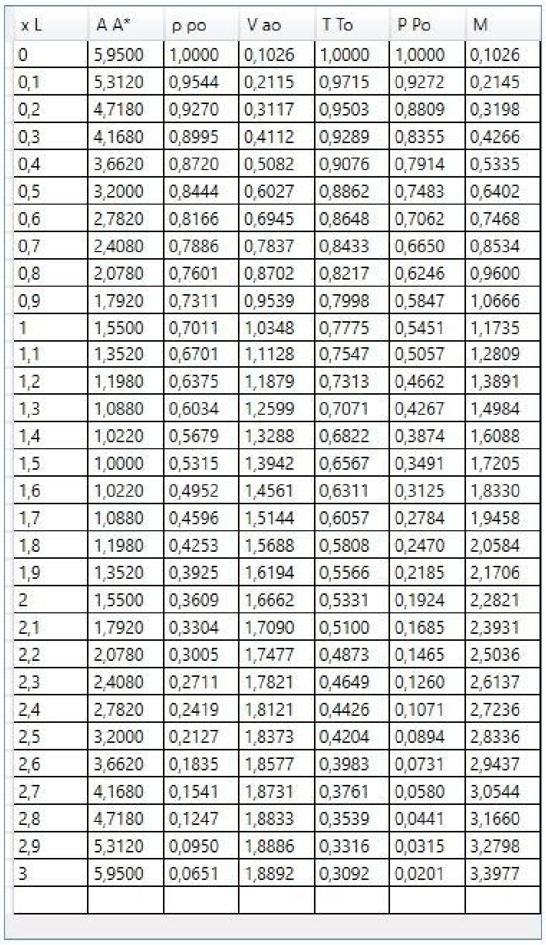
Now the application receives as an input the results of the previous section, which means the initial state of the nozzle, and then applies the equation of our simulator class library in order to get the first step results. The variation of each one of the parameters of the nozzle along the horizontal axis, in this case, don't give us any information.

As in the previous case, we attach the tables of the values from the Anderson and from our simulation, in order to an easy comparison.



x L	A A*	p po	V ao	T To	P Po	M
0	5,95	1	0,111	1	1	0,111
0,1	5,312	0,955	0,212	0,972	0,928	0,215
0,2	4,718	0,927	0,312	0,95	0,881	0,32
0,3	4,168	0,9	0,411	0,929	0,836	0,427
0,4	3,662	0,872	0,508	0,908	0,791	0,534
0,5	3,2	0,844	0,603	0,886	0,748	0,64
0,6	2,782	0,817	0,695	0,865	0,706	0,747
0,7	2,408	0,789	0,784	0,843	0,665	0,854
0,8	2,078	0,76	0,87	0,822	0,625	0,96
0,9	1,792	0,731	0,954	0,8	0,585	1,067
1	1,55	0,701	1,035	0,778	0,545	1,174
1,1	1,352	0,67	1,113	0,755	0,506	1,281
1,2	1,198	0,637	1,188	0,731	0,466	1,389
1,3	1,088	0,603	1,26	0,707	0,426	1,498
1,4	1,022	0,567	1,328	0,682	0,387	1,609
1,5	1	0,531	1,394	0,656	0,349	1,72
1,6	1,022	0,494	1,455	0,631	0,312	1,833
1,7	1,088	0,459	1,514	0,605	0,278	1,945
1,8	1,198	0,425	1,568	0,581	0,247	2,058
1,9	1,352	0,392	1,619	0,556	0,218	2,171
2	1,55	0,361	1,666	0,533	0,192	2,282
2,1	1,792	0,33	1,709	0,51	0,168	2,393
2,2	2,078	0,301	1,748	0,487	0,146	2,504
2,3	2,408	0,271	1,782	0,465	0,126	2,614
2,4	2,782	0,242	1,813	0,443	0,107	2,724
2,5	3,2	0,213	1,838	0,421	0,09	2,834
2,6	3,662	0,184	1,858	0,398	0,073	2,944
2,7	4,168	0,154	1,874	0,376	0,058	3,055
2,8	4,718	0,125	1,884	0,354	0,044	3,167
2,9	5,312	0,095	1,89	0,332	0,032	3,281
3	5,95	0,066	1,895	0,309	0,02	3,406

4.2.1. *First step state from Anderson*



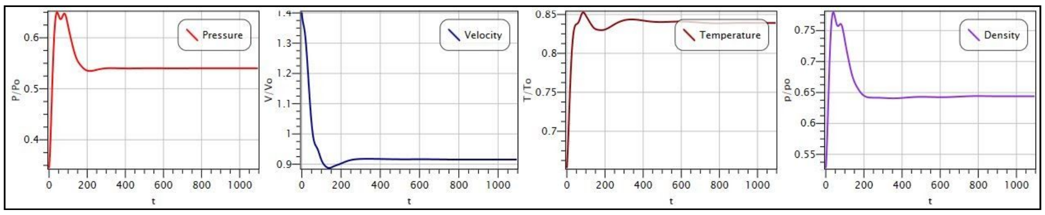
x L	A A*	p po	V ao	T To	P Po	M
0	5,9500	1,0000	0,1026	1,0000	1,0000	0,1026
0,1	5,3120	0,9544	0,2115	0,9715	0,9272	0,2145
0,2	4,7180	0,9270	0,3117	0,9503	0,8809	0,3198
0,3	4,1680	0,8995	0,4112	0,9289	0,8355	0,4266
0,4	3,6620	0,8720	0,5082	0,9076	0,7914	0,5335
0,5	3,2000	0,8444	0,6027	0,8862	0,7483	0,6402
0,6	2,7820	0,8166	0,6945	0,8648	0,7062	0,7468
0,7	2,4080	0,7886	0,7837	0,8433	0,6650	0,8534
0,8	2,0780	0,7601	0,8702	0,8217	0,6246	0,9600
0,9	1,7920	0,7311	0,9539	0,7998	0,5847	1,0666
1	1,5500	0,7011	1,0348	0,7775	0,5451	1,1735
1,1	1,3520	0,6701	1,1128	0,7547	0,5057	1,2809
1,2	1,1980	0,6375	1,1879	0,7313	0,4662	1,3891
1,3	1,0880	0,6034	1,2599	0,7071	0,4267	1,4984
1,4	1,0220	0,5679	1,3288	0,6822	0,3874	1,6088
1,5	1,0000	0,5315	1,3942	0,6567	0,3491	1,7205
1,6	1,0220	0,4952	1,4561	0,6311	0,3125	1,8330
1,7	1,0880	0,4596	1,5144	0,6057	0,2784	1,9458
1,8	1,1980	0,4253	1,5688	0,5808	0,2470	2,0584
1,9	1,3520	0,3925	1,6194	0,5566	0,2185	2,1706
2	1,5500	0,3609	1,6662	0,5331	0,1924	2,2821
2,1	1,7920	0,3304	1,7090	0,5100	0,1685	2,3931
2,2	2,0780	0,3005	1,7477	0,4873	0,1465	2,5036
2,3	2,4080	0,2711	1,7821	0,4649	0,1260	2,6137
2,4	2,7820	0,2419	1,8121	0,4426	0,1071	2,7236
2,5	3,2000	0,2127	1,8373	0,4204	0,0894	2,8336
2,6	3,6620	0,1835	1,8577	0,3983	0,0731	2,9437
2,7	4,1680	0,1541	1,8731	0,3761	0,0580	3,0544
2,8	4,7180	0,1247	1,8833	0,3539	0,0441	3,1660
2,9	5,3120	0,0950	1,8886	0,3316	0,0315	3,2798
3	5,9500	0,0651	1,8892	0,3092	0,0201	3,3977

4.2.2. *First step state from simulation*

In this case, in the table that gives the difference in percentage between the two tables the biggest one is below the 2%, so the results go on being accurate.

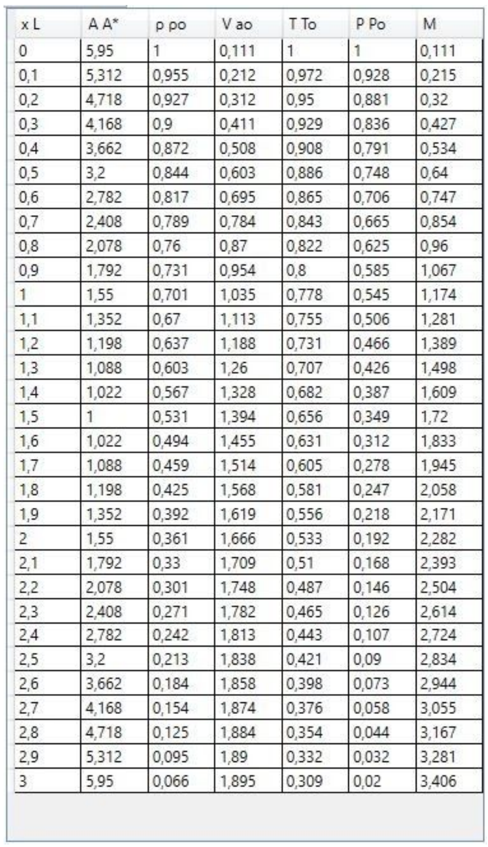
4.3 Steady state

Finally, in this last study state, the application keeps simulating from the last state until it reaches the steady state, in which the parameters doesn't vary with the time steps. So now the important plots are the ones that show the variation on time.



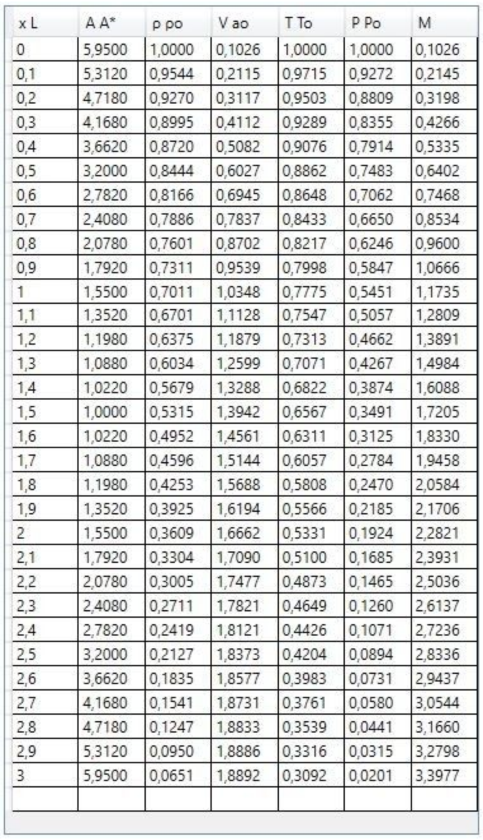
4.3.1. *Change on time of each one of the flow parameters*

As it can be seen, with 500 iterations the steady state is already reached. As in Anderson the table of the steady state is given with 1400 time steps, we have also done this number of steps to get the table:



x L	A A*	ρ ρo	V ao	T To	P Po	M
0	5,95	1	0,111	1	1	0,111
0,1	5,312	0,955	0,212	0,972	0,928	0,215
0,2	4,718	0,927	0,312	0,95	0,881	0,32
0,3	4,168	0,9	0,411	0,929	0,836	0,427
0,4	3,662	0,872	0,508	0,908	0,791	0,534
0,5	3,2	0,844	0,603	0,886	0,748	0,64
0,6	2,782	0,817	0,695	0,865	0,706	0,747
0,7	2,408	0,789	0,784	0,843	0,665	0,854
0,8	2,078	0,76	0,87	0,822	0,625	0,96
0,9	1,792	0,731	0,954	0,8	0,585	1,067
1	1,55	0,701	1,035	0,778	0,545	1,174
1,1	1,352	0,67	1,113	0,755	0,506	1,281
1,2	1,198	0,637	1,188	0,731	0,466	1,389
1,3	1,088	0,603	1,26	0,707	0,426	1,498
1,4	1,022	0,567	1,328	0,682	0,387	1,609
1,5	1	0,531	1,394	0,656	0,349	1,72
1,6	1,022	0,494	1,455	0,631	0,312	1,833
1,7	1,088	0,459	1,514	0,605	0,278	1,945
1,8	1,198	0,425	1,568	0,581	0,247	2,058
1,9	1,352	0,392	1,619	0,556	0,218	2,171
2	1,55	0,361	1,666	0,533	0,192	2,282
2,1	1,792	0,33	1,709	0,51	0,168	2,393
2,2	2,078	0,301	1,748	0,487	0,146	2,504
2,3	2,408	0,271	1,782	0,465	0,126	2,614
2,4	2,782	0,242	1,813	0,443	0,107	2,724
2,5	3,2	0,213	1,838	0,421	0,09	2,834
2,6	3,662	0,184	1,858	0,398	0,073	2,944
2,7	4,168	0,154	1,874	0,376	0,058	3,055
2,8	4,718	0,125	1,884	0,354	0,044	3,167
2,9	5,312	0,095	1,89	0,332	0,032	3,281
3	5,95	0,066	1,895	0,309	0,02	3,406

4.3.2. *Steady state from Anderson*



x L	A A*	ρ ρo	V ao	T To	P Po	M
0	5,9500	1,0000	0,1026	1,0000	1,0000	0,1026
0,1	5,3120	0,9544	0,2115	0,9715	0,9272	0,2145
0,2	4,7180	0,9270	0,3117	0,9503	0,8809	0,3198
0,3	4,1680	0,8995	0,4112	0,9289	0,8355	0,4266
0,4	3,6620	0,8720	0,5082	0,9076	0,7914	0,5335
0,5	3,2000	0,8444	0,6027	0,8862	0,7483	0,6402
0,6	2,7820	0,8166	0,6945	0,8648	0,7062	0,7468
0,7	2,4080	0,7886	0,7837	0,8433	0,6650	0,8534
0,8	2,0780	0,7601	0,8702	0,8217	0,6246	0,9600
0,9	1,7920	0,7311	0,9539	0,7998	0,5847	1,0666
1	1,5500	0,7011	1,0348	0,7775	0,5451	1,1735
1,1	1,3520	0,6701	1,1128	0,7547	0,5057	1,2809
1,2	1,1980	0,6375	1,1879	0,7313	0,4662	1,3891
1,3	1,0880	0,6034	1,2599	0,7071	0,4267	1,4984
1,4	1,0220	0,5679	1,3288	0,6822	0,3874	1,6088
1,5	1,0000	0,5315	1,3942	0,6567	0,3491	1,7205
1,6	1,0220	0,4952	1,4561	0,6311	0,3125	1,8330
1,7	1,0880	0,4596	1,5144	0,6057	0,2784	1,9458
1,8	1,1980	0,4253	1,5688	0,5808	0,2470	2,0584
1,9	1,3520	0,3925	1,6194	0,5566	0,2185	2,1706
2	1,5500	0,3609	1,6662	0,5331	0,1924	2,2821
2,1	1,7920	0,3304	1,7090	0,5100	0,1685	2,3931
2,2	2,0780	0,3005	1,7477	0,4873	0,1465	2,5036
2,3	2,4080	0,2711	1,7821	0,4649	0,1260	2,6137
2,4	2,7820	0,2419	1,8121	0,4426	0,1071	2,7236
2,5	3,2000	0,2127	1,8373	0,4204	0,0894	2,8336
2,6	3,6620	0,1835	1,8577	0,3983	0,0731	2,9437
2,7	4,1680	0,1541	1,8731	0,3761	0,0580	3,0544
2,8	4,7180	0,1247	1,8833	0,3539	0,0441	3,1660
2,9	5,3120	0,0950	1,8886	0,3316	0,0315	3,2798
3	5,9500	0,0651	1,8892	0,3092	0,0201	3,3977

4.3.3. *Steady state from simulation*

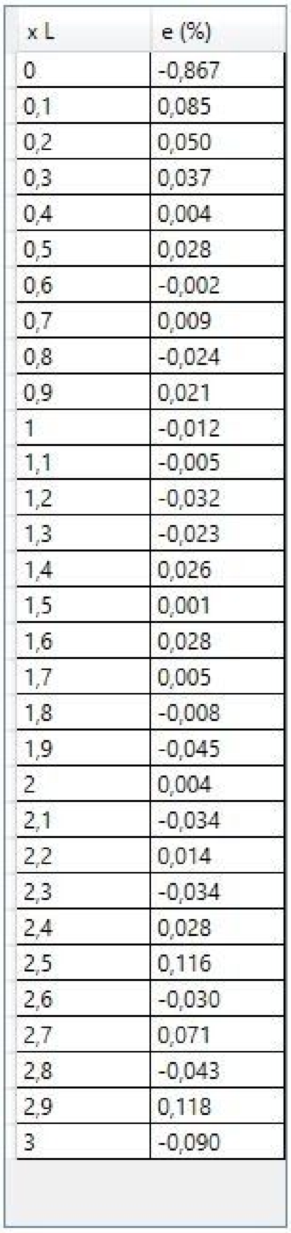
In this case, the difference between the values gets bigger with the horizontal axis, so in the outflow the results are more different than in the inflow.

As we have said before analysing the results, the book rounds all numbers and keeps simulating with the rounded values, so it is reasonable that the difference on values get bigger with the number of iterations.

4.4 Overall Errors

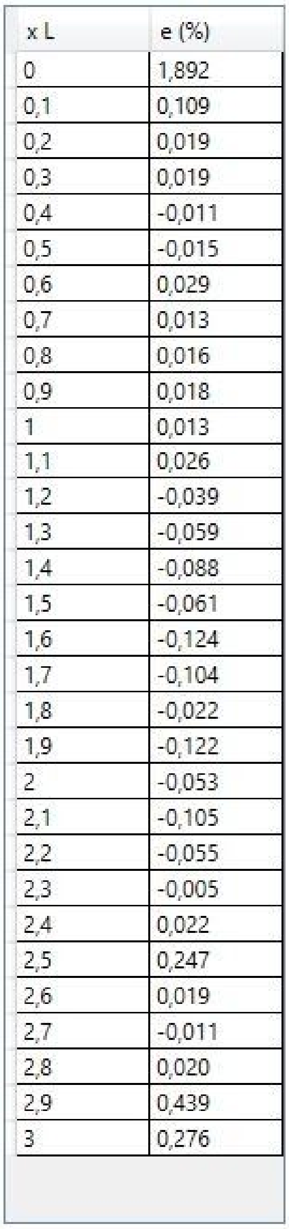
Finally we can talk about the differences (comparing with Anderson) made by the simulator overall, for that we've implemented a simple table on the *Anderson* window to allow the user a quick look of the comparison.

For this task we've used a simple *DataGrid* which display the average results variations of all the four parameters on each point of the grid (instead of showing the individual of each parameter), allowing a much more simpler way to compare:



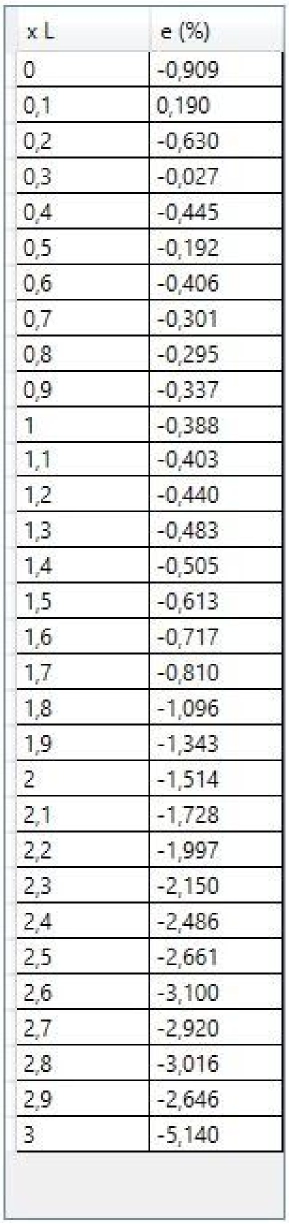
x L	e (%)
0	-0,867
0,1	0,085
0,2	0,050
0,3	0,037
0,4	0,004
0,5	0,028
0,6	-0,002
0,7	0,009
0,8	-0,024
0,9	0,021
1	-0,012
1,1	-0,005
1,2	-0,032
1,3	-0,023
1,4	0,026
1,5	0,001
1,6	0,028
1,7	0,005
1,8	-0,008
1,9	-0,045
2	0,004
2,1	-0,034
2,2	0,014
2,3	-0,034
2,4	0,028
2,5	0,116
2,6	-0,030
2,7	0,071
2,8	-0,043
2,9	0,118
3	-0,090

4.4.1. *Initial Conditions differences*



x L	e (%)
0	1,892
0,1	0,109
0,2	0,019
0,3	0,019
0,4	-0,011
0,5	-0,015
0,6	0,029
0,7	0,013
0,8	0,016
0,9	0,018
1	0,013
1,1	0,026
1,2	-0,039
1,3	-0,059
1,4	-0,088
1,5	-0,061
1,6	-0,124
1,7	-0,104
1,8	-0,022
1,9	-0,122
2	-0,053
2,1	-0,105
2,2	-0,055
2,3	-0,005
2,4	0,022
2,5	0,247
2,6	0,019
2,7	-0,011
2,8	0,020
2,9	0,439
3	0,276

4.4.2. *First Step differences*



x L	e (%)
0	-0,909
0,1	0,190
0,2	-0,630
0,3	-0,027
0,4	-0,445
0,5	-0,192
0,6	-0,406
0,7	-0,301
0,8	-0,295
0,9	-0,337
1	-0,388
1,1	-0,403
1,2	-0,440
1,3	-0,483
1,4	-0,505
1,5	-0,613
1,6	-0,717
1,7	-0,810
1,8	-1,096
1,9	-1,343
2	-1,514
2,1	-1,728
2,2	-1,997
2,3	-2,150
2,4	-2,486
2,5	-2,661
2,6	-3,100
2,7	-2,920
2,8	-3,016
2,9	-2,646
3	-5,140

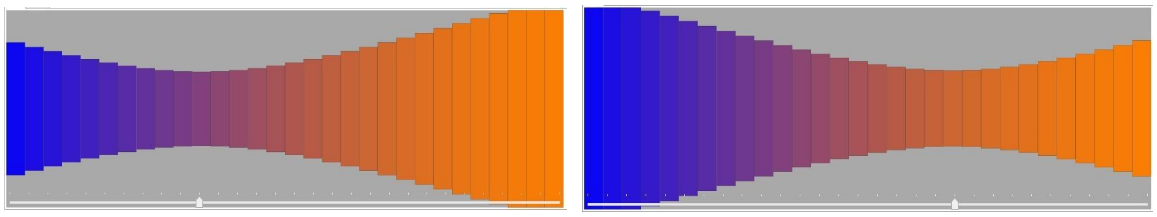
4.4.3. *Steady State differences*

As have been mentioned before, progress is getting on overall, in comparison to the *Anderson* values errors, currently are smaller, for the most part it does not get the 1% error, meaning that the simulator is accurate and calculates correctly.

5. Advanced Study

For the advanced study we've decided that it would be interesting not only to be able to change the number of steps and the number of rectangles of the nozzle but to change the position of the nozzle in a much more simpler way.

We've implemented a slider that allows the user to change the position of the throat before initiating the simulation in real time, the slider adaps to all possible locations of the throat depending on the number of rectangles, allowing for a more pleasant way of setting up the modified nozzle and then start the simulation, here are some examples:

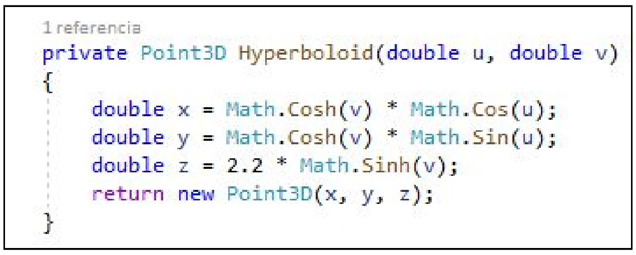


5.1. *Nozzle shape changed example*

After the position of the nozzle is set, the slider locks and the simulation can start with the newly set nozzle, although some shapes can make the simulation become a bit unstable.

Also for the sake of good visualization has been implemented a 3D view of the nozzle using *ViewPort3D*, allowing the user for a good visualization of the newly modified nozzle so the adequate shape can be decided.

We've created a new class: *ParametricSurface3D* which allows us to, given a certain function, compute 3D points based on a certain equation. In our case this function is a 3D hyperbola, which is an hyperboloid, and the function would be inputs in the class function that computes the points (the one below, being u and v the parameterised variables).



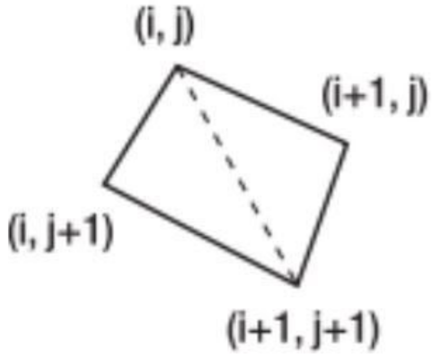
```
1 referencia
private Point3D Hyperboloid(double u, double v)
{
    double x = Math.Cosh(v) * Math.Cos(u);
    double y = Math.Cosh(v) * Math.Sin(u);
    double z = 2.2 * Math.Sinh(v);
    return new Point3D(x, y, z);
}
```

5.2. Function which generates three different values to generate the 3D simulation

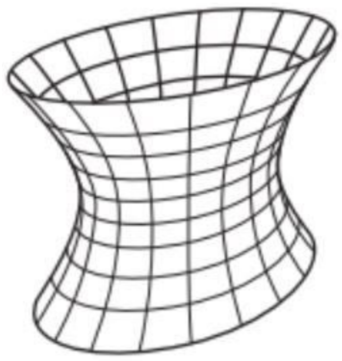
The *ParametricSurface3D* class will use the function and *Utility* to create a surface based on a certain parameters such as the number of rectangles, the variable range (ymax, xmax, vmax, umax , etc.. Related on how big the nozzle will be), the location, the surface color, etc.

For this task we have used what is called a mesh 3D object geometry using triangles and indexing those triangles to joining them together to form a 3D surface object.

For the representation of a 3D surface we cannot use triangle faces because it would not represent reality, therefore we need to find a way to work with rectangular meshes instead of triangles, that's why we've implemented a class called *Utility,* which will create the rectangle faces based on triangles to create the surfaces we want.

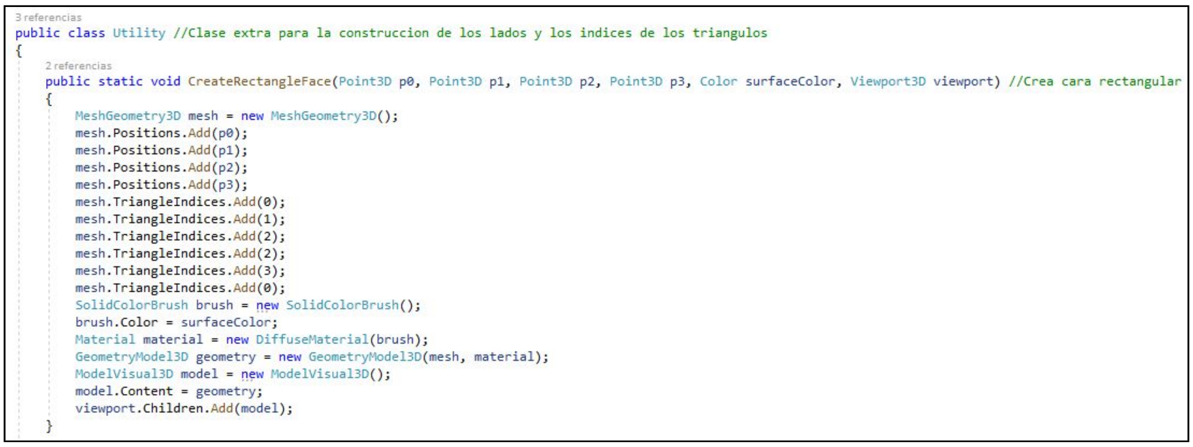


5.3. *Rectangular faces*



5.4. *Hyperboloid surface*

There is one method inside the class that do this task: *CreateRenctangleFace().*

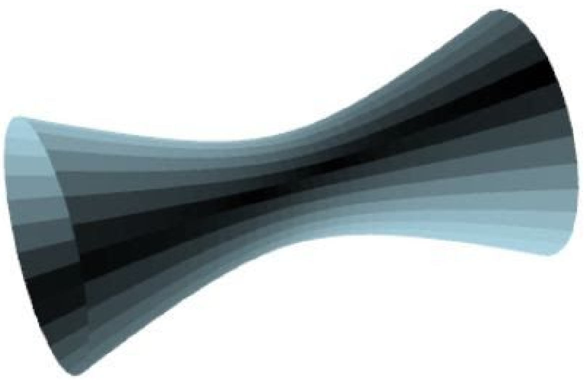


```
3 referencias
public class Utility //Clase extra para la construccion de los lados y los indices de los triangulos
{
    2 referencias
    public static void CreateRectangleFace(Point3D p0, Point3D p1, Point3D p2, Point3D p3, Color surfaceColor, Viewport3D viewport) //Crea cara rectangular
    {
        MeshGeometry3D mesh = new MeshGeometry3D();
        mesh.Positions.Add(p0);
        mesh.Positions.Add(p1);
        mesh.Positions.Add(p2);
        mesh.Positions.Add(p3);
        mesh.TriangleIndices.Add(0);
        mesh.TriangleIndices.Add(1);
        mesh.TriangleIndices.Add(2);
        mesh.TriangleIndices.Add(2);
        mesh.TriangleIndices.Add(3);
        mesh.TriangleIndices.Add(0);
        SolidColorBrush brush = new SolidColorBrush();
        brush.Color = surfaceColor;
        Material material = new DiffuseMaterial(brush);
        GeometryModel3D geometry = new GeometryModel3D(mesh, material);
        ModelVisual3D model = new ModelVisual3D();
        model.Content = geometry;
        viewport.Children.Add(model);
    }
```

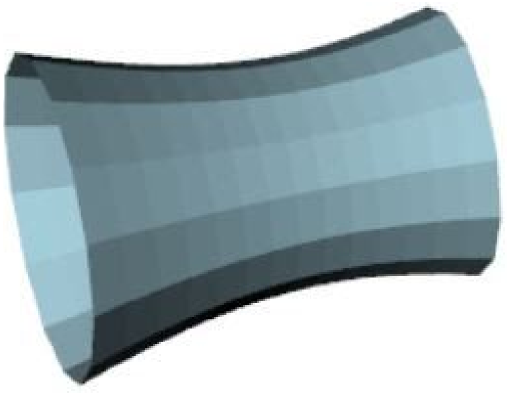
5.5. *CreateRectangleFace method*

The implementation of that will allow us using a *viewport3D* viewer to build and modify or paremetricised nozzle and, in combination with the slider, also will allow the user to see how the parameters imputed translate into the real life object its going to be simulated.

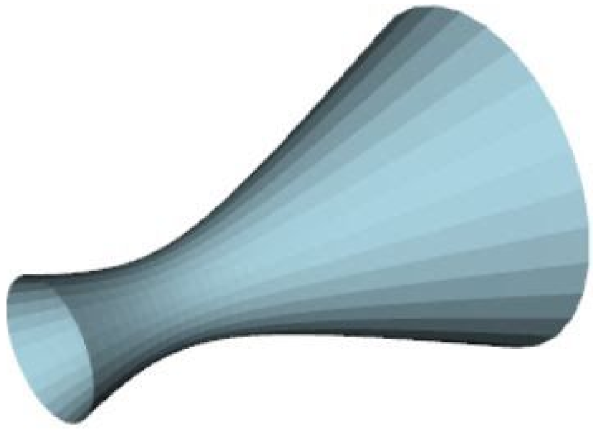
Some examples of different shapes in 3D are shown in next figures.



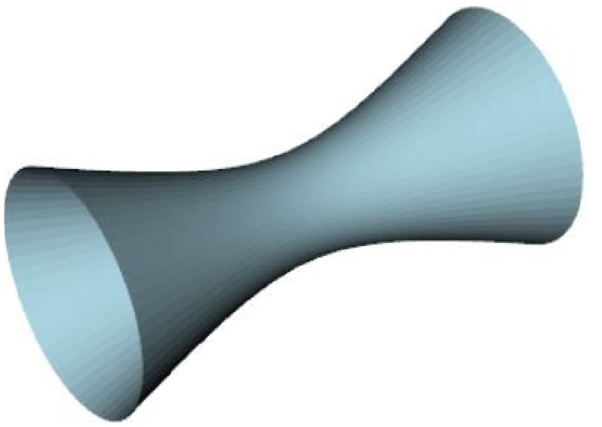
5.6. *Default nozzle*
(30 rectangles of 0.1 width with centered throat)



5.7. *Bad-discretized nozzle*
(15 rectangles of 0.1 width with centered throat)



5.8. *Non-symmetric nozzle*
(30 rectangles of 0.1 width with no-centered throat)



5.9. *Good-discretized nozzle*
(60 rectangles of 0.05 width with centered throat)

REFERENCES

[1] J. D. Anderson, Computational Fluid Mechanics, University of Maryland, McGraw-Hill, 1995

[2] Nozzle Wikipedia
https://en.wikipedia.org/wiki/Nozzle

[3] Engineering notes web
https://www.engineeringenotes.com/thermal-engineering/nozzle/nozzle-applications-general-flow-analysis-velocity-pressure-and-phenomenon-thermodynamics/50082

[4] Quora Free fluid jet
https://www.quora.com/What-does-“free-fluid-jet”-mean-in-fluid-mechanics

[5] Toberas, Rosa Sucasaca
https://es.scribd.com/document/269476892/TOBERAS

[6] Converging-Diverging Nozzles
http://www.dept.aoe.vt.edu/~devenpor/aoe3114/CD%20Nozzle%20Sim/index.html

[7] Practical WPF Charts and Graphics, Jack Xu