UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
Faculdade de Engenharia
Campus de Ilha Solteira

Pedro Henrique de Araújo Bitencourt

Algorithm for Two-dimensional Airfoil Automatic Structured Mesh Generation

Ilha Solteira - SP
Novembro de 2021

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
Faculdade de Engenharia
Campus de Ilha Solteira

Pedro Henrique de Araújo Bitencourt

Algorithm for Two-dimensional Airfoil Automatic Structured Mesh Generation

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia de Ilha Solteira – UNESP como parte dos requisitos para obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Prof. Dr. Aluisio Viais Pantaleão

Co-orientador: Eng. Henrique Matos Campos

Ilha Solteira - SP
Novembro de 2021

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

B624a

Bitencourt, Pedro Henrique de Araújo.
Algorithm for two-dimensional airfoil automatic structured mesh generation / Pedro Henrique de Araújo Bitencourt. -- Ilha Solteira: [s.n.], 2021
45 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Mecânica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2021

Orientador: Aluisio Viais Pantaleão
Co-orientador: Henrique Matos Campos
Inclui bibliografia

1. Cfd. 2. Openfoam. 3. Python. 4. Aerodinâmica. 5. Malha estruturada. 6. Blockmesh.

[illegible]

CURSO DE ENGENHARIA MECÂNICA

ATA DA DEFESA - TRABALHO DE GRADUAÇÃO

TITULO: **"ALGORITHM FOR TWO-DIMENSIONAL AIRFOIL AUTOMATIC STRUCTURED MESH GENERATION"**

ALUNO: Pedro Henrique de Araújo Bitencourt **RA: 161050727**

ORIENTADOR: Prof. Dr. Aluisio Viais Pantaleão
CO-ORIENTADOR: Eng Henrique Matos Campos

Aprovado (X) - Reprovado ( ) pela Comissão Examinadora

Comissão Examinadora:

[illegible]

Prof. Dr. Aluisio Viais Pantaleão
Presidente (Orientador)

Henrique Matos Campos

Eng. Henrique Matos Campos
(Co-Orientador)

[illegible]

Prof. Dr. Emanuel Rocha Woiski

Bianca T. Visoná Carnielo

Eng. Bianca Taís Visoná Carnielo

Pedro Henrique de A Bitencourt

Pedro Henrique de Araújo Bitencourt

Ilha Solteira, 10 de novembro de 2021.

Acknowledgments

I would like to thank my family and friends for all the support and love they have put in me during my journey.

I would like to express my gratitude to my advisor Professor Aluisio Viais Pantaleão who has always been a constant source of encouragement, corrections, teachings and help over the years, since the beginning of my scientific initiation.

For the extracurricular groups that I participated, Equipe Zebra Aerodesign and Empresa Júnior Precisão Engenharia, who gave me many lessons on how to be a professional, I thank for all the moments together.

I also thank everyone who has contributed in any way to this work, including professors and employees of Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculty of Engineering of Ilha Solteira and the Mechanical Engineering Department.

Resumo

A utilização de métodos numéricos para previsão de fenômenos aerodinâmicos exerce papel de destaque na indústria, e a geração de malhas é uma etapa extremamente importante para os resultados esperados. Este trabalho busca desenvolver a geração automática de malhas estruturadas por meio de integração entre a linguagem Python e o software livre OpenFOAM, o qual interpreta, por meio da ferramenta blockMesh, um dicionário e assim gera a malha estruturada. A comparação com dados experimentais se faz necessária a fim de verificar a eficiência computacional e acurácia das malhas estruturadas geradas.

Palavras-chave: CFD, aerodinâmica, malha estruturada, BlockMesh, OpenFOAM, Python.

Abstract

The use of numerical methods to forecast aerodynamic phenomena has an important role in the industry, and the generation of meshes is an extremely important step for the expected results. This work seeks to develop the automatic generation of structured meshes through integration between the Python language and the open source software OpenFOAM, which interprets, through the blockMesh tool, a dictionary, then generates the structured mesh. The comparison with experimental data was used in order to verify the computational efficiency and accuracy of the generated structured meshes.

Keywords: CFD, aerodynamics, structured mesh, BlockMesh, OpenFOAM, Python.

List of Figures

Contents

1 Introduction

This chapter presents a brief part of the knowledge used as inspiration to develop this study.

1.1 Structured mesh

Engineering has evolved in recent years, trying to reduce the number of experiments in the industry, considering the economic expenses. One of the solutions to this problem is the numerical simulations, since, with one computational structure, it is possible to obtain results equivalent to several experiments, thus, reducing costs. Therefore, several researches and evaluations are regarding numerical simulations, so they use computational meshes for their applications. The meshing generation method varies a lot according to the degree of geometric complexity since the more complex, the greater the difficulty in developing the computational mesh. The literature reports several generation methods, including structured, unstructured and mixed meshes. The choice of a method is due to the search for the best cost-benefit of generation time linked to greater efficiency in the convergence rate and accuracy obtained in the solution of the problem.

The methods to automatically generate meshes are being studied and developed by countless researchers. Within the aspect of unstructured meshes Pantaleão et al. (2003) developed an automatic generator for an unstructured mesh of triangular elements, in which it was produced by a computer code using the Delaunay algorithm applied to the Finite Element Method.

Benoit & Péron (2012) researched a method to automatically generate structured meshes around two-dimensional bodies for CFD simulations, with a method that consists of avoiding low-quality meshes for locally pointed bodies, such as the trailing edge of an airfoil and is able to automatically perform flow simulation around any geometry made of polylines.

Also Schmidt et al. (2012) developed a new library for the software OpenFOAM that can be used for the automated generation of structured meshes, the library provides tools for the organization of a large number of blocks and is used before to OpenFOAM's native block mesher blockMesh.

In the paper of Lu et al. (2018), the authors present a method of automatic generation of structured meshes in arbitrary aircraft, together with a method of automatic generation of a boundary layer mesh using multi-block structured from a superficial mesh, but there was no automatic generation of the *far-field*. Liu & Hu (2018) studied the fluid-structure interaction problems based on his previous research on the method of refining structured mesh adaptable to incompressible flows.

Zhang & Jia (2018) developed an algorithm for the automatic generation of structured meshes applicable to two-dimensional domains with complex geometries. In the work of Yu et al. (2019), the algorithm for automatic mesh generation was further studied, which was more robust on the defective CAD geometries, three cases were studied with regards to the validation of the mesh generation approach: a leaking hydrogen tank, a steam generator and a single car garage.

Faghih-Naini et al. (2020) proposed a method implemented within the code generation framework of the ExaStencils project using the SymPy Python library, in which the new formulation uses block-structured triangular meshes automatically generated for a given number of blocks.

Beaufort et al. (2020) used a robust pipeline to handle triangulations, based on the computation of a one-to-one parametrization for automatically selected patches of input triangles, which made each patch easier to re-meshing and was implemented in the open source mesh generator Gmsh.

And Lu et al. (2020) implemented, from his previous work, an interactive structured mesh generation software called NNW-GridStar, with accelerated techniques, which consist of automatic boundary-layer mesh generation, rapid block assembly, multi-block stretching, and O-type block stretching.

We realize the importance demonstrated by different authors in the mesh generation for various problems encountered. This work developed structured meshes automatically generated on 2D supporting surfaces, including *far-field*. The efficiency of solution convergence and the degree of accuracy produced by the structured meshes are evaluated, comparing the results obtained from calculations with non-structured meshes and experimental data from the literature.

It is noteworthy that the dictionary used to generate the structured meshes are automatically written by an integration with the Python programming language, in which the starting point is a generic 2D aerodynamic profile.

1.2 Goal

Develop a method to automatically generate structured meshes, including the *far-field* and y^+ calculation, with the outline of 2D lifting surfaces, included asymmetric airfoils, using free software, the Python language and OpenFOAM, leading to a high cost-benefit.

2 Methodology

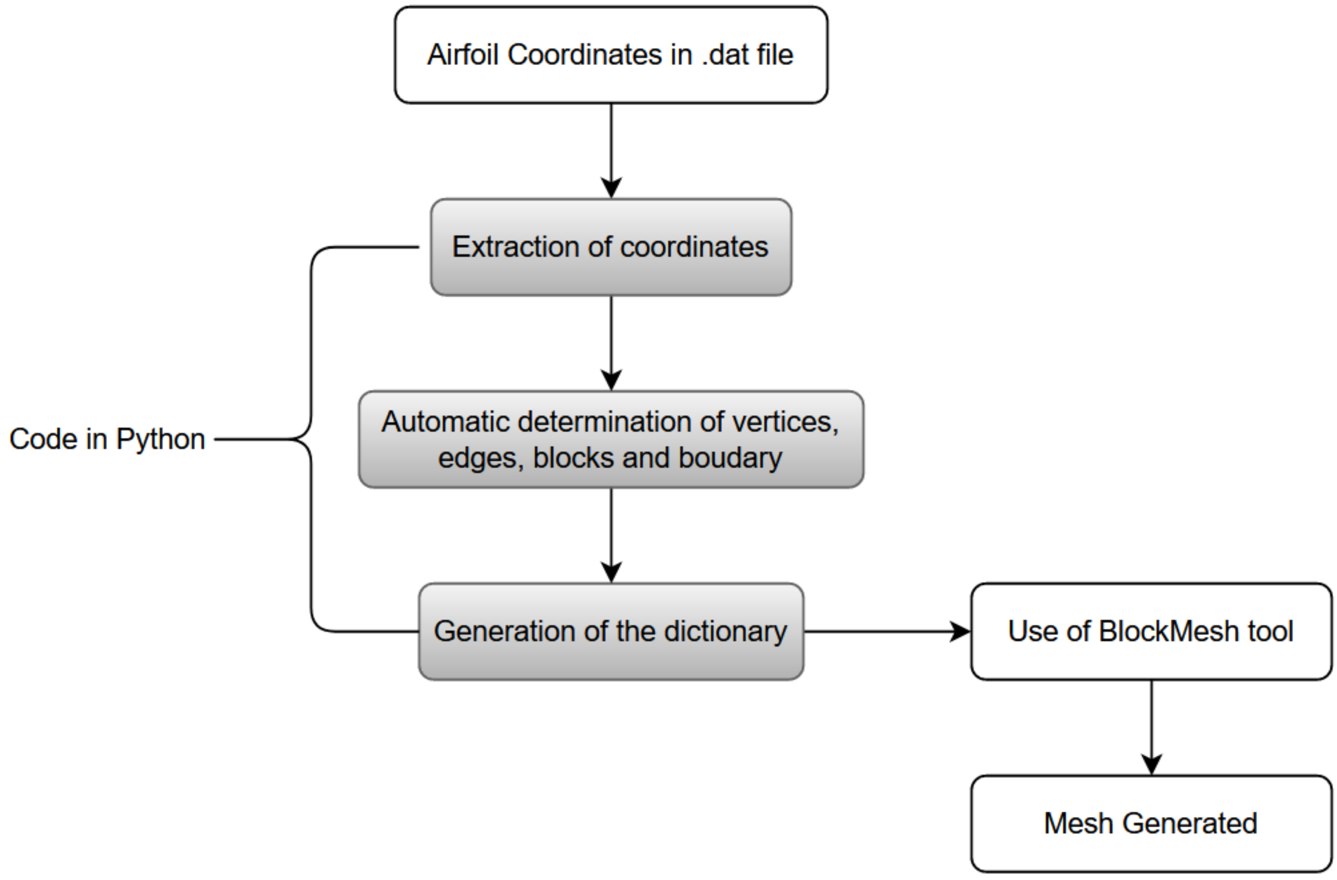
This chapter presents the description of the method and procedure used in the development of this study.

2.1 Integration between software

The methodology of the project consists in the integration between the selected simulation software and the programming language, in this case the OpenFOAM software and the Python language were chosen, since in addition to being open source, there is a tool for generating structured meshes within the software.

The method utilized is simplified in the diagram shown in Figure 1, in which the gray blocks correspond to the code reported here.

Figure 1: Diagram of the method used to generate the mesh.


Airfoil Coordinates in .dat file
Extraction of coordinates
Code in Python
Automatic determination of vertices, edges, blocks and boudary
Generation of the dictionary
Use of BlockMesh tool
Mesh Generated

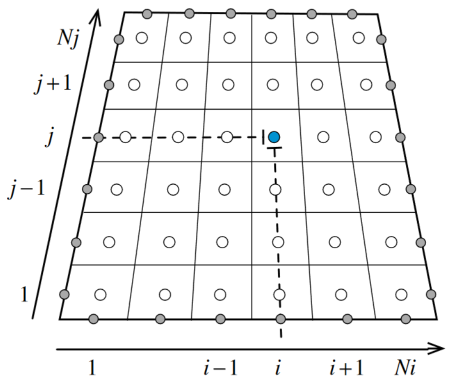
Source: Author

Given the points of desired airfoil, using the Python programming language, the code calculates far field, and generates the parameters necessary to construct a mesh, so the file is automatically generated. In this way, the blockMesh tool of OpenFOAM v7 from OpenFOAM Foundation (2020) was used, in which it interprets the dictionary generated by the Python language, and thus relates the data of points, faces, cells and mesh divisions in a new dictionary.

2.2 OpenFOAM - blockMesh

The blockMesh tool generates a structured mesh, that is explained in Moukalled et al. (2015). In this kind of mesh, every cell in the interior of the domain is connected to the same number of neighboring cells, which can be identified using the indices, like i,j,k, normally used in the x,y,z coordinates, and that it can be directly accessed by incrementing or decrementing the respective indices.

Figure 2: Example concerning the topology of a structured mesh.



Source: Moukalled et al. (2015)

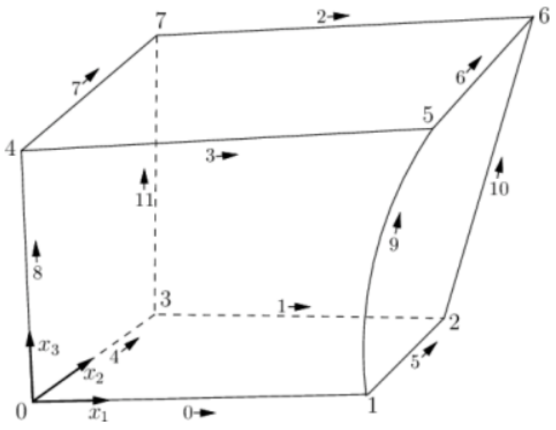
The generation of structured mesh by the blockMesh tool is through a dictionary named ”blockMeshDict”, located in the folder ”system” of the case under analysis in OpenFOAM. Basically, as explained in OpenFoam (2020), it decompose the domain geometry into several three-dimensional hexahedron blocks. The vertices allow the generation of lines, arcs and splines, in addition to specifying the number of cells in each direction of the block, whole set being characterized as a mesh.

2.2.1 blockMeshDict File

The file ”blockMeshDict”, specified in González et al. (2009), is a dictionary with keywords as: ”scale”, ”vertices”, ”edges”, ”block”, ”patches” e ”mergePatchPairs”.

The keyword ”scale” specify the scale factor applied in the coordinates of vertices specified in the mesh. The values in the dictionary are initially in units of the global system.

Figure 3: Example of block.



Source: OpenFoam (2020)

The vertices block are listed with the keyword ”vertices”, like the following example:

vertices((0 0 0) // vertices 0
(1 0 0.1) // vertices 1
(1.1 1 0.1) // vertices 2
(0 1 0.1) // vertices 3
(-0.1 -0.1 1) // vertices 4
(1.3 0 1.2) // vertices 5
(1.4 1.1 1.3) // vertices 6
(0 1 1.1) // vertices 7 **);**

The edges are connected between two vertices, using lines as the standard, however it can be specified like curves using a list with the keyword ”edges”. Different types of curves can be specified, like:”arc”, ”simpleSpline”, ”polyLine”, ”polySpline”, being the first representative of an arc, needing just one point of interpolation. The subsequent ones need a list of points, since they represent a ”spline” curve, a set of lines and a set of ”spline” respectively. An example:

edges(arc 1 5 (1.0 2.0 3.0)
spline 0 1((0.001 0.002 0.003)
(0.001 0.002 0.000)
(0.001 0.002 0.000)
(0.001 0.002 0.000)) **);**

Blocks are defines by a list using the keyword ”blocks”, which is composed by a list of vertices, describe above, in a vector form, also it contains the quantity of cells and a list with expansion proportion in each direction (x,y,z). The order os vertices must be made in a way that it follows two parallel circles, as demonstrated in Figure 3, the order of quantity of cells is following x, y, z respectively. The expansion proportion follows the same principle of cells quantity, and is this aspect that allows a refinement in the required areas without adding more cells.

blocks(hex (0 1 2 3 4 5 6 7) // number of the vertices
(10 10 10) // number of the cell in each direction
simpleGrading (1 2 3)
or
edgeGrading (1 1 1 1 2 2 2 2 3 3 3 3) // proportion of cell expansion **);**

Mesh division if presented with a list by the keyword "boundary", where the regions ("patches") are listed with their names specified by the author, being characterized by an intern dictionary that has "type" and "faces", the first one represents the conditions applied in the region or a particular geometry condition, the second determine faces where the conditions will be applied. Example:

boundary(inlet { // name of patch
type patch; // type of patch
faces((0 4 7 3)); // block that belongs in the patch } **);**

2.2.2 Using the blockMesh

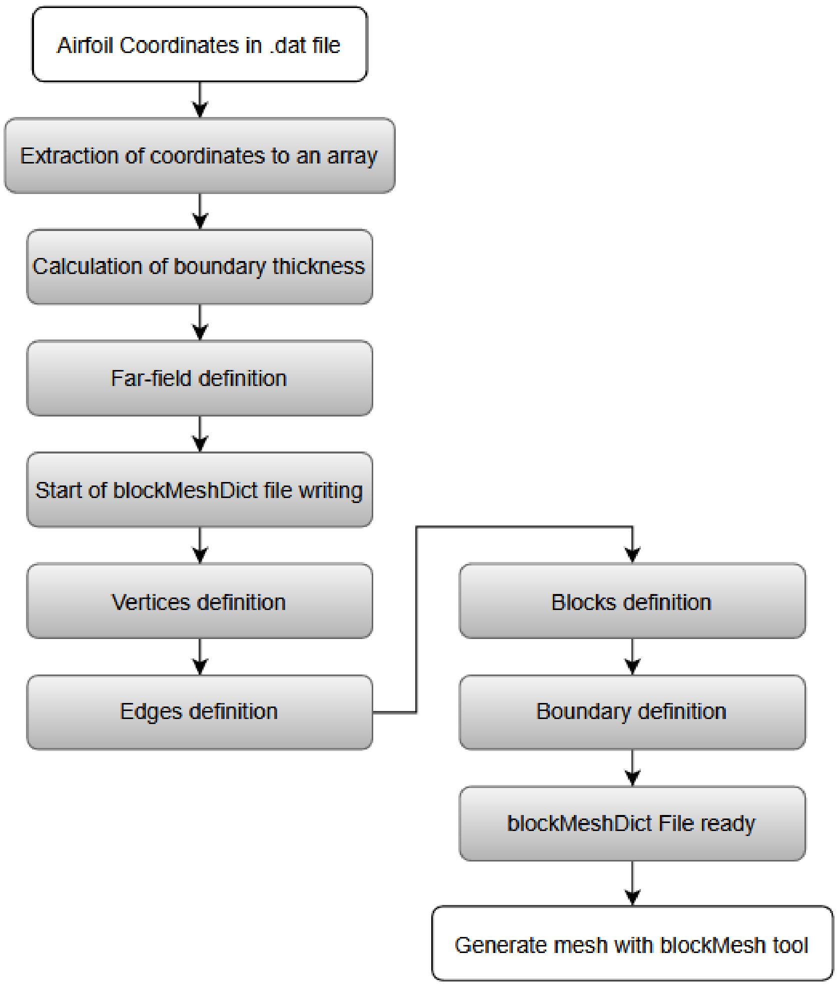
With the file "blockMeshDict" ready and the software OpenFOAM, we execute the command "blockMesh -Name-of-case-", generating the complementary files of the mesh. Other files of needs to be written and placed in their respective folders so the simulation can be done.

2.3 Python program

The Python language was selected for the packages used for scientific computing, such as numpy, matplotlib and panda, used in this work.

Using the structure of a blockMesh file, explained above, the program was developed to write the file after all calculation was done, as demonstrated in the Figure 4, the code are presented in the Appendix.

Figure 4: Diagram of the algorithm developed.


Airfoil Coordinates in .dat file
Extraction of coordinates to an array
Calculation of boundary thickness
Far-field definition
Start of blockMeshDict file writing
Vertices definition
Edges definition
Blocks definition
Boundary definition
blockMeshDict File ready
Generate mesh with blockMesh tool

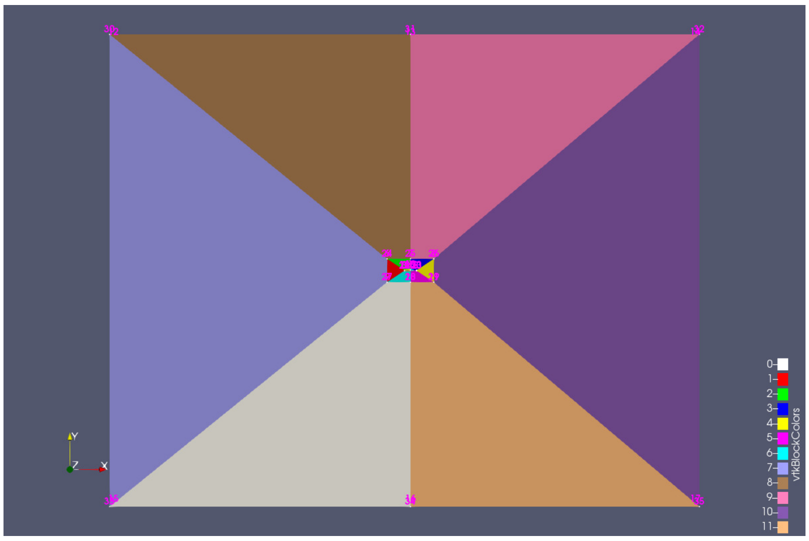
Source: Author

First, using panda, acquisition of the points from the airfoil file, the cord and y^+ desired, also the simulation parameters, such as temperature, Mach number and Reynolds number.

With all the initial aspects acquired, it is calculated the far-field, the vertices and edges locations in the mesh, using numpy functions.

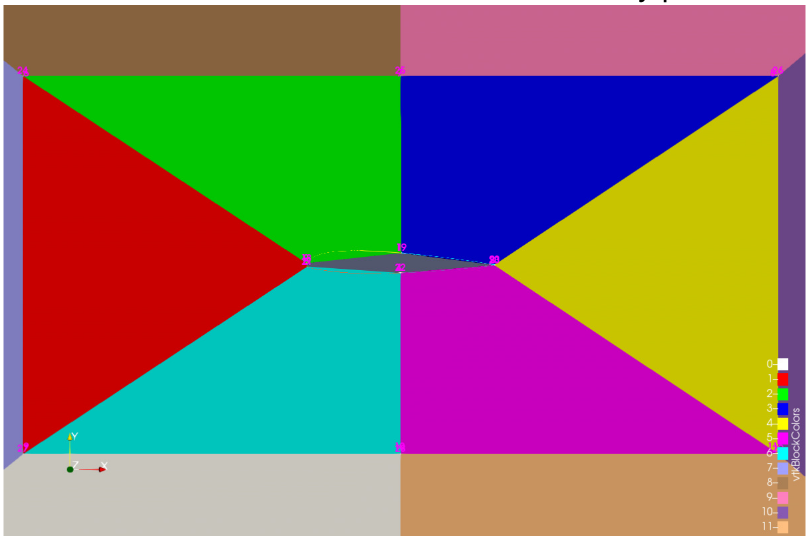
The final step is to locate the vertices according to the blockMesh file, separating correctly the blocks and boundaries, an example of how the blocks are distributed presented Figure 5.

Figure 5: Demonstration on how the blocks are automatically placed in the mesh generated.



Source: Author

Figure 6: Demonstration on how the blocks are automatically placed around the airfoil.



Source: Author

The distribution of blocks demonstrated on Figure 6, so near the airfoil the mesh can be more refined, while reducing the elements in the external area of the mesh.

After all preparation is done, the program write the file, and it is ready to be used in OpenFOAM.

2.4 GCI Analysis

The Grid Convergence Index (GCI) method described in Celik et al. 2008 is a recommended bibliografy that has been evaluated over several hundred CFD cases. It contains five steps, explained below:

Step 1. Define a representative cell, mesh, or grid size h. For two-dimensions calculations:

$$h = \left[\frac{1}{N} sum_{i=1}^{N}(\delta A_i)\right]^{\frac{1}{2}} \tag{1}$$

Step 2. Select three significantly different sets of grids. It is desirable that the grid refinement factor $r = h_{coarse}/h_{fine}$ be greater than 1.3. From the results, select a property from the simulation and three correspondent values to the different meshes, ϕ_1, ϕ_2 and ϕ_3.

$$\varepsilon_{21} = \phi_2 - \phi_1 \qquad and \qquad \varepsilon_{32} = \phi_3 - \phi_2 \tag{2}$$

Step 3. Let $h_1 < h_2 < h_3$ and $r_{21} = h_2/h_1$, $r_{32} = h_3/h_2$, and calculate the apparent order p of the method using the expression:

$$p = \frac{1}{ln(r_{21})}\left|ln\left|\frac{\varepsilon_{32}}{\varepsilon_{21}}\right| + q(p)\right| \tag{3}$$

$$q(p) = ln\left(\frac{r_{21}^{p} - s}{r_{32}^{p} - s}\right) \tag{4}$$

$$h = 1.sgn(\frac{\varepsilon_{32}}{\varepsilon_{21}}) \tag{5}$$

Step 4. Calculate the extrapolated values from:

$$\phi_{ext}^{21} = \frac{r_{21}^{p}\phi_1 - \phi_2}{r_{21}^{p} - 1} \qquad and \qquad \phi_{ext}^{32} = \frac{r_{32}^{p}\phi_2 - \phi_3}{r_{32}^{p} - 1} \tag{6}$$

Step 5. Calculate and report the following error estimates, along with the apparent order p:

Approximate relative error:

$$e_a^{21} = \left|\frac{\phi_1 - \phi_2}{\phi_1}\right| \tag{7}$$

$$e_a^{23} = \left|\frac{\phi_2 - \phi_3}{\phi_2}\right| \tag{8}$$

Extrapolated relative error:

$$e_{ext}^{21} = \left|\frac{\phi_{ext}^{21} - \phi_1}{\phi_{ext}^{21}}\right| \tag{9}$$

$$e_{ext}^{32} = \left|\frac{\phi_{ext}^{32} - \phi_2}{\phi_{ext}^{32}}\right| \tag{10}$$

Grid convergence index:

$$GCI_{21} = \frac{1.25 e_a^{21}}{r_{21}^p - 1} \tag{11}$$

$$GCI_{32} = \frac{1.25 e_a^{32}}{r_{32}^p - 1} \tag{12}$$

3 Experimental data

This chapter presents the description of experimental data used as parameters in the simulation to validate the mesh.

3.1 Analyzed test cases

To verify the quality of the meshes generated by the Python code, we used different flow conditions.

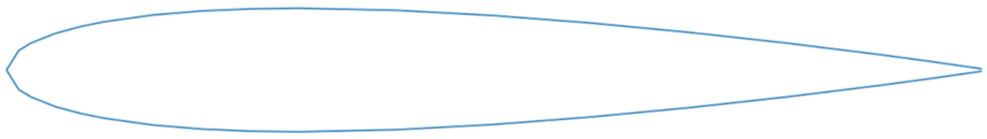
The first test case verified was a symmetric airfoil, NACA 0012, found in Rumsey (2019) and consists of an incompressible, turbulent, and steady flow over the airfoil. Table 1 presents more details about the flow conditions for the test case.

Figure 7: NACA 0012.

Source: Author

The second test case defined in the study was for the symmetric airfoil NACA 0021, Wolfe & Ochs (1997) presents more details about the test case. This test case also consists of an incompressible, turbulent, and steady flow.

Figure 8: NACA 0021.



Source: Author

The third test case defined in the study was for an asymmetric airfoil, NACA 23012, Pan & Loth (2003) presents more details about the test case, except for the temperature used in their case, so, for this work, were used 25℃ for the simulation. This test case also consists of an incompressible, turbulent, and steady flow.

Figure 9: NACA 23012.

Source: Author

Table 1 presents more information about the flow conditions for the test cases.

Table 1: Airfoil validation cases information.

Test case	**NACA 0012**	**NACA 0021**	**NACA 23012**
Reference	Rumsey (2019)	Wolfe & Ochs (1997)	Pan & Loth (2003)
Prandtl Number	0,72	0,72	-
Temperature [K]	299,85	299,85	-
Mach Number	0,15	0,20	0,12
Chord [m]	1	1	1
Reynolds Number	6$\cdot 10^6$	1,5$\cdot 10^6$	10,5$\cdot 10^6$
Angle of Attack [$^\circ$]	0	0	0

Source: Author

Since all the analyzed test cases use air as fluid, it was considered as an ideal gas. With this adoption, to evaluate the dynamic viscosity was used Sutherland's Law.

As all the flows analyzed are turbulent, to solve the closure problem, was adopted the Spalart and Allmaras turbulence model.

4 Results e Discussion

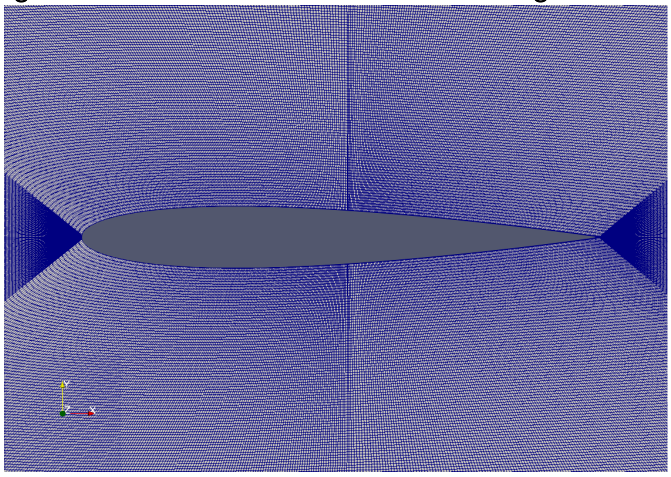
4.1 Analysis of NACA 0012 airfoil

The results of the mesh generation and simulation of a NACA 0012 are disposed below.

4.1.1 Mesh

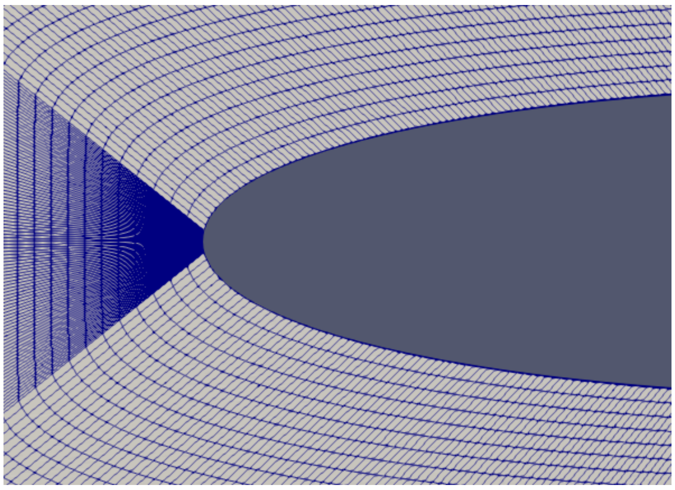
Using the points of a NACA 0012 and the method explained in the methodology, the far field has a size 20 times bigger than the airfoil chord. The mesh disposed in the Figures 10 and 11 was generated for an y^+ value of 60, with approximately 2 millions hexahedral elements, since OpenFOAM operates with a three-dimensional mesh, even for two-dimensional cases, where the third dimension only has 1 element. Other two mesh with different values of y^+ were generated for GCI analysis.

Figure 10: Demonstration of the mesh generated.



Source: Author

Figure 11: Demonstration of the mesh around the leading edge.

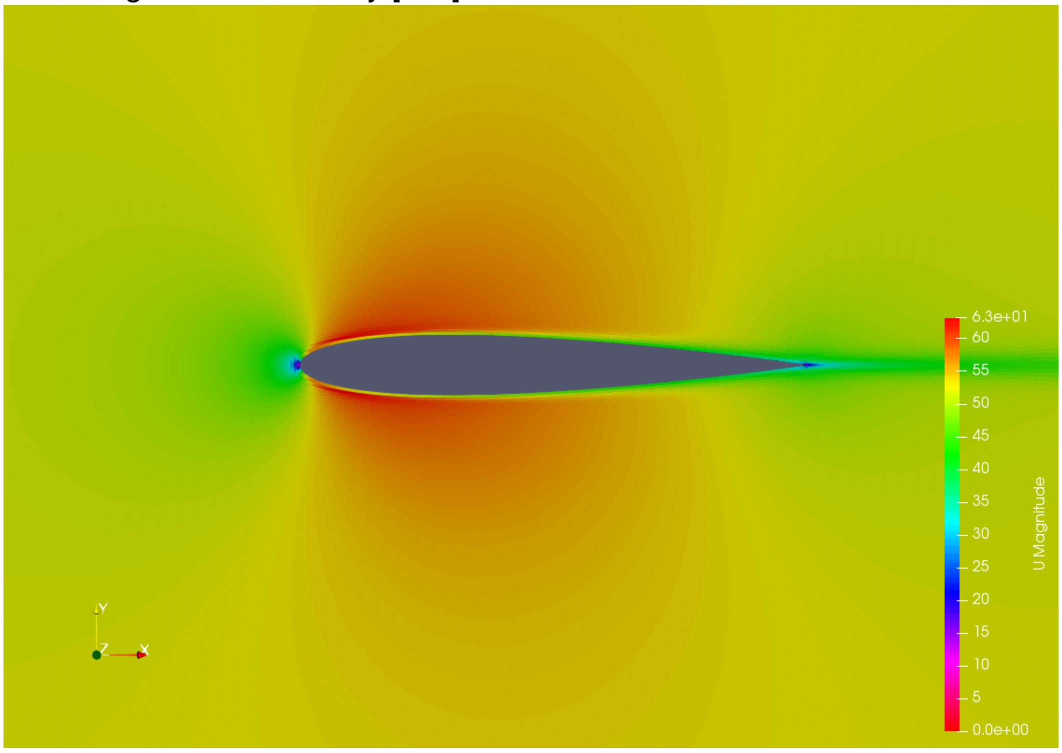


Source: Author

4.1.2 Numerical Results

Figure 12 presents the results evaluated for velocity fields obtained for NACA 0012:

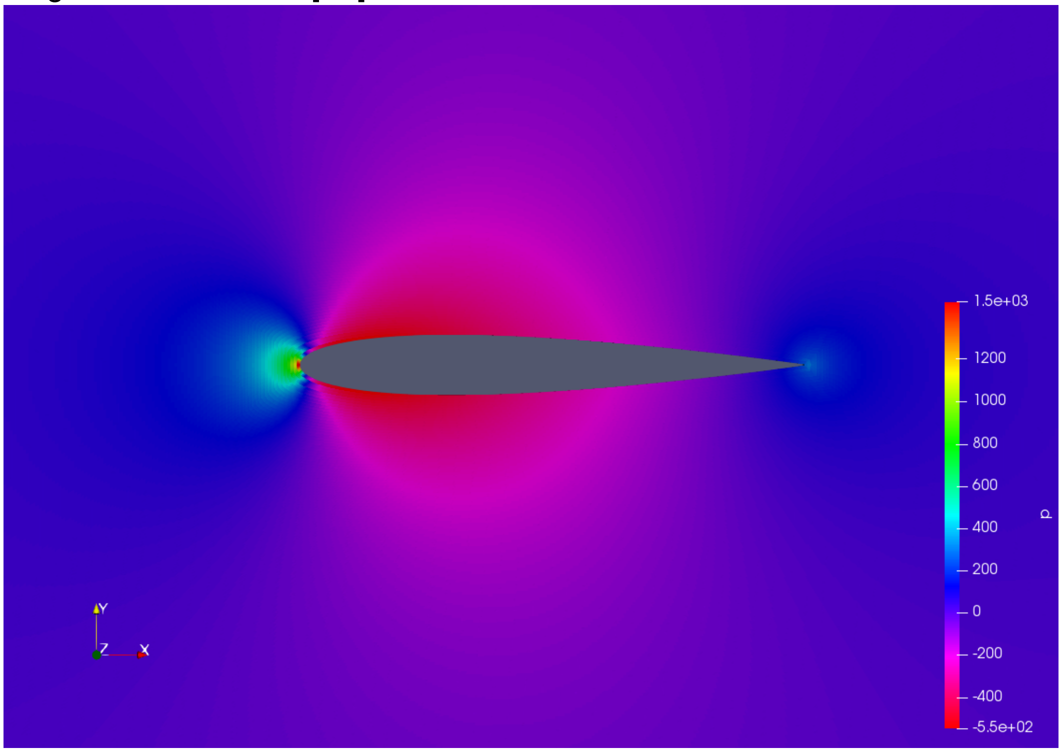
Figure 12: Magnitude of Velocity [m/s] data from the flow field around the NACA 0012.



Source: Author

Figure 13 presents results for pressure fields obtained:

Figure 13: Pressure [Pa] data from the flow field around the NACA 0012.



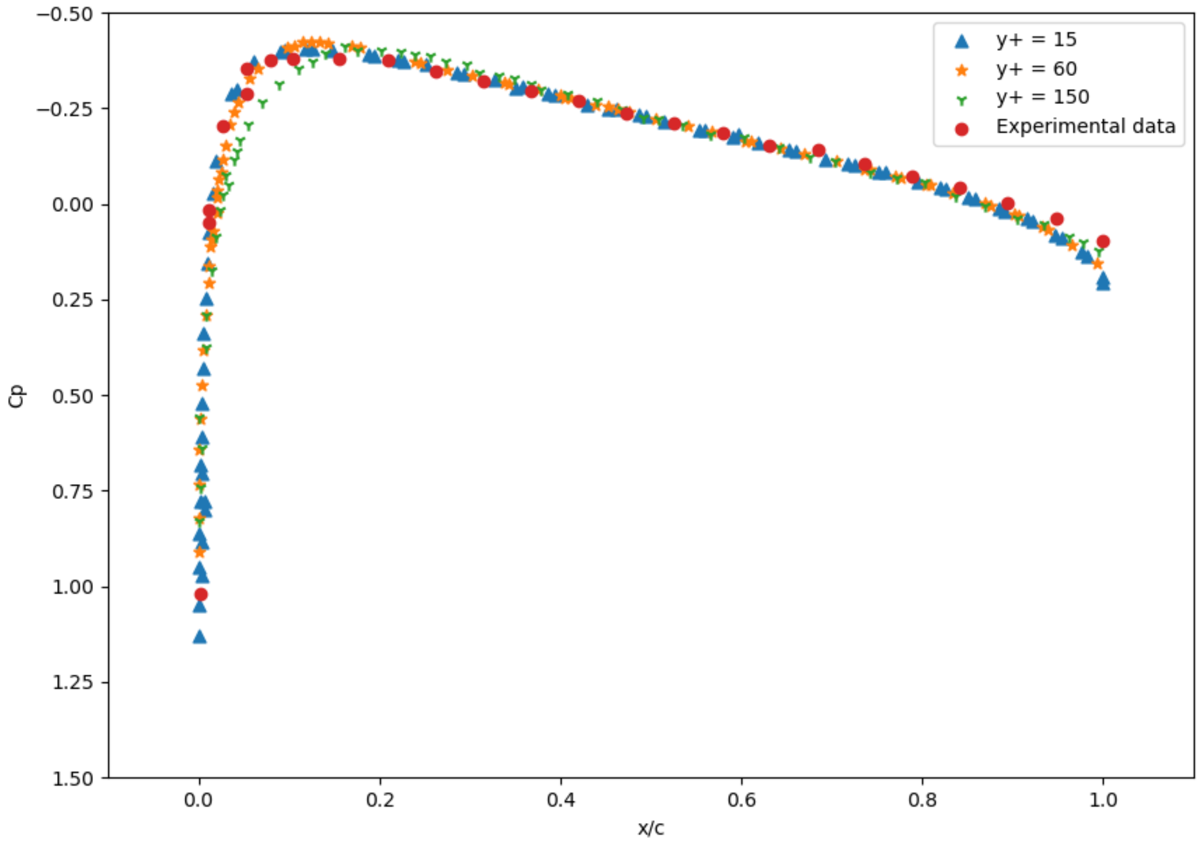
Source: Author

From Figures 12 and 13 can be noticed that since it is a symmetrical airfoil at 0°angle of attack, shows that the flow stream is symmetric and the velocity of the upper and lower surface of the airfoil are equal. The flow moves smoothly around the airfoil and is attached around the surface. And since there is no difference between the pressure on top and bottom surfaces, no lift force is generated as expected from the literature.

For the comparison with the literature, Jespersen et al. (2016) from NASA exercise a simulation with NACA 0012, exactly same case that was explained above.

Figure 14 presents the comparison between the experimental data provided by Jespersen et al. (2016) and the numerical data evaluated with the analysis for the Pressure Coefficient(C_p) over the airfoil.

Figure 14: Comparison of the C_p evaluated with the numerical analysis with the experimental data provided by Jespersen et al. (2016) for NACA 0012 airfoil.



Source: Author

As can be seen in Figure 14 the numerical data almost fit with the experimental data, just differing slightly in the trailing edge of the airfoil.

So, following the methodology proposed by Celik et al. (2008), the Grid Convergence Index (GCI) was verified for the NACA 0012 test case generated meshes with C_p as the chosen parameter. Table 2 presents the results evaluated in the GCI analysis.

Table 2: GCI analysis for NACA 0012 test case.

Number of Elements in mesh 1	7821000
Number of elements in mesh 2	1955000
Number of elements in mesh 3	782000
Refinement factor r_{21}	$1,5875$
Refinement factor r_{32}	$1,3572$
Approximate relative error e_{a21}	$4,26\%$
Approximate relative error e_{a32}	$16,5\%$
Extrapolated relative error e_{ex21}	$0,69\%$
Extrapolated relative error e_{ex32}	$5,81\%$
Convergence index GCI_{21}	$0,86\%$
Convergence index GCI_{32}	$7,72\%$

Source: Author

By comparing the results presented in Table 2, and considering the methodology proposed by Celik et al. (2008), can be verified that the medium grid must be used in the analysis, due to GCI_{21} being less than 1% and approximate relative error less than 5%.

Comparing the curves presented in Figure 14 was verified that the mesh with $y^+ = 15$ and $y^+ = 60$ presented better results, almost fitting with the experimental data during all the range o x/c, just diverging in trailing edge.

The worst results are found in the $y^+ = 150$ mesh where the data diverges in trailing edge and next to the maximum C_p peak.

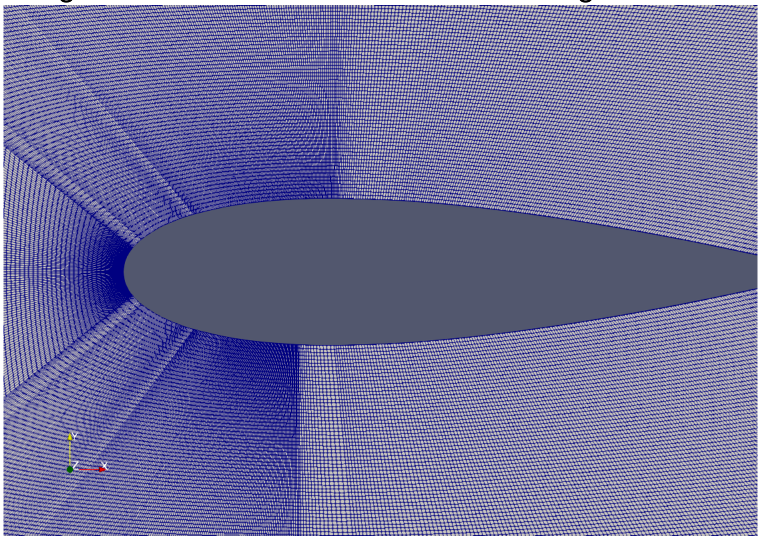
4.2 Analysis of NACA 0021 airfoil

Following the proposed methodology was studied the NACA 0021 airfoil, the results of the mesh generation, and evaluated in numerical simulation of the case are presented below.

4.2.1 Mesh

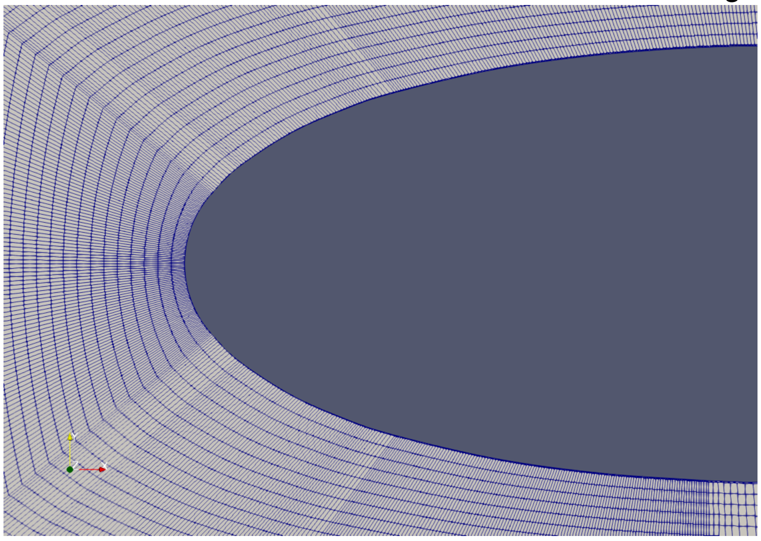
Using the points of a NACA 0021 and the method explained in the methodology, the far field has a size 20 times bigger than the airfoil chord. The mesh disposed in the Figures 15 and 16 was generated for an y^+ value of 60 , with approximately 2 millions hexahedral elements, since OpenFOAM operates with a three-dimensional mesh, even for two-dimensional cases, where the third dimension only has 1 element. Other two mesh with different values of y^+ were generated for GCI analysis.

Figure 15: Demonstration of the mesh generated.



Source: Author

Figure 16: Demonstration of the mesh around the leading edge.

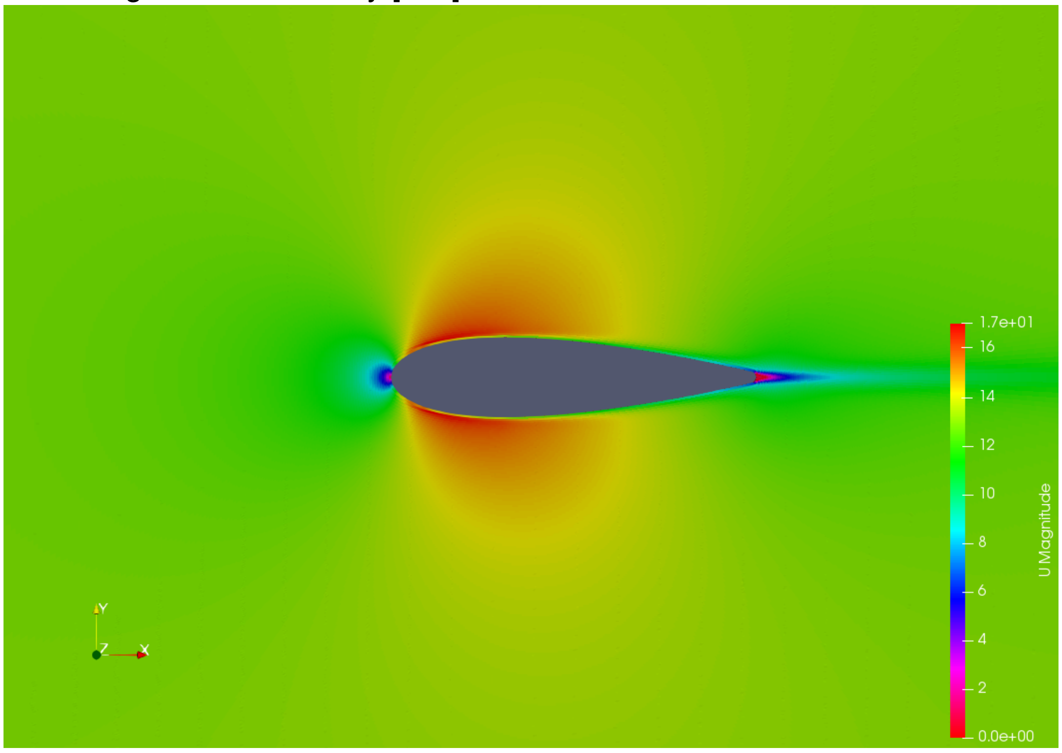


Source: Author

4.2.2 Numerical Results

Figure 17 presents the results evaluated for velocity fields obtained for NACA 0021 :

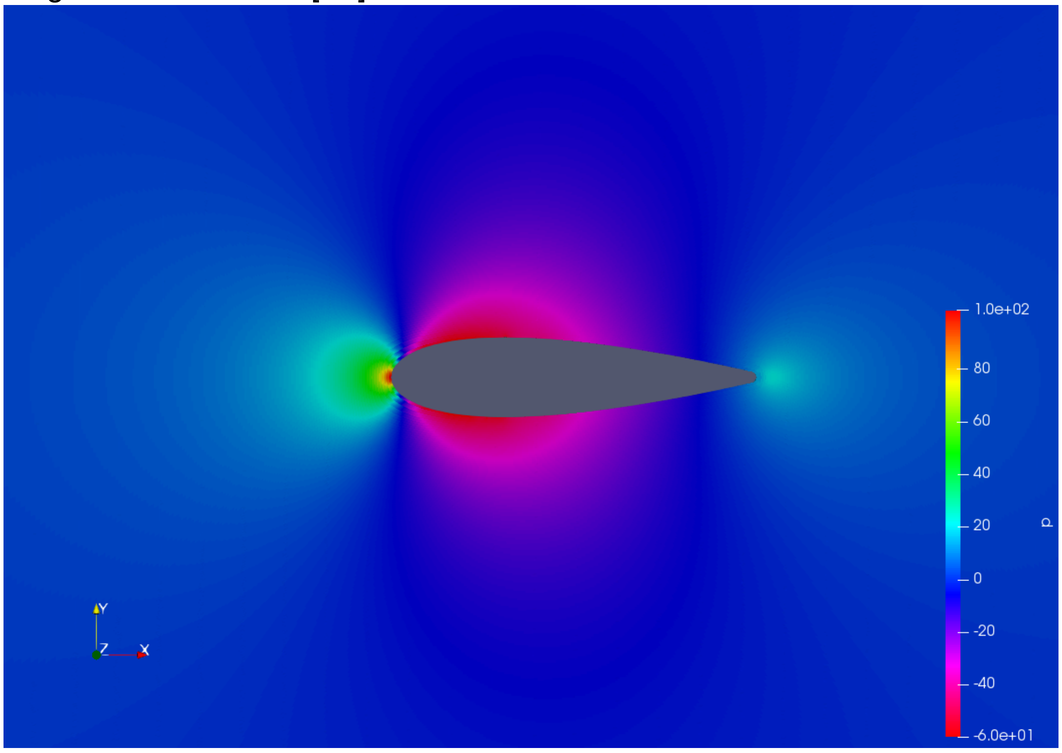
Figure 17: Magnitude of Velocity [m/s] data from the flow field around the NACA 0021.



Source: Author

Figure 18 presents results for pressure fields obtained:

Figure 18: Pressure [Pa] data from the flow field around the NACA 0021.

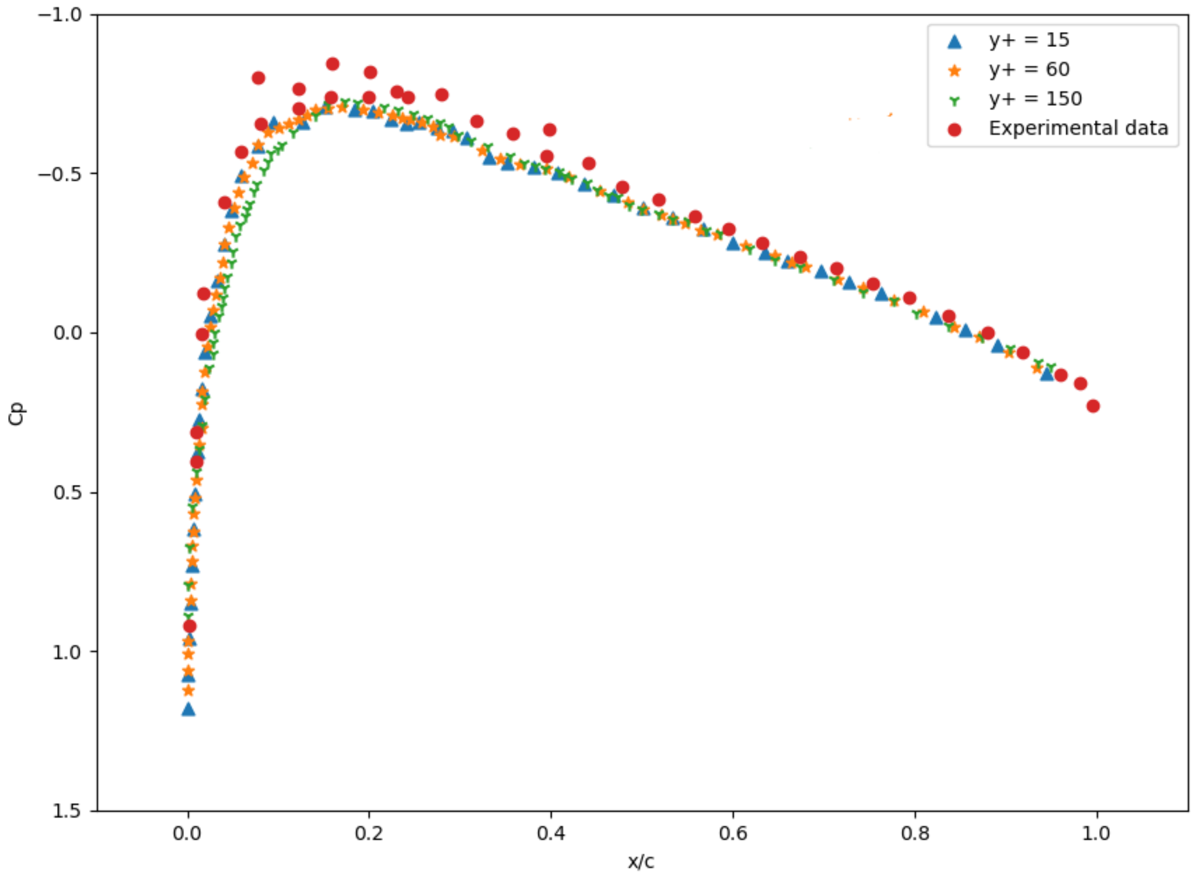


Source: Author

From Figures 17 and 18 can be noticed that since it is a symmetrical airfoil at 0°angle of attack, shows that the flow stream is symmetric and the velocity of the upper and lower surface of the airfoil are equal. The flow moves smoothly around the airfoil and is attached around the surface. And since there is no difference between the pressure on top and bottom surfaces, no lift force is generated as expected from the literature.

Figure 19 presents the C_p comparison between the experimental data provided by Wolfe & Ochs (1997) and the numerical data evaluated with the analysis over the airfoil NACA 0021.

Figure 19: Comparison of the C_p evaluated with the numerical analysis with the experimental data provided by Wolfe & Ochs (1997) for NACA 0021 airfoil.



Source: Author

As can be seen in Figure 19, the numerical data evaluated in the analysis keeps the same behavior of the experimental data presented by Wolfe & Ochs (1997). However, the numerical results diverge slightly from the values found by the experiments.

Another issue found in the NACA 0021 case was the representation of flow over the leading edge. In this region, the numerical data presented a significant difference from the experimental values of C_p.

Differing from the NACA 0012 test case, where was found a problem in the representation of the flow over the trailing edge, the numerical analysis was capable of representing the flow over the trailing edge adequately for the NACA 0021 case, fitting experimental and numerical data in this region.

Following the procedure of Celik et al. (2008), was verified the GCI for the NACA 0021 test case generated meshes with C_p as the chosen parameter, the results evaluated are presented in Table 3.

Table 3: GCI analysis for NACA 0021 test case.

Number of Elements in mesh 1	3886000
Number of elements in mesh 2	1943000
Number of elements in mesh 3	777000
Refinement factor r_{21}	$1,2599$
Refinement factor r_{32}	$1,3573$
Approximate relative error e_{a21}	$0,78\%$
Approximate relative error e_{a32}	$4,04\%$
Extrapolated relative error e_{ex21}	$0,362\%$
Extrapolated relative error e_{ex32}	$1,13\%$
Convergence index GCI_{21}	$0,45\%$
Convergence index GCI_{32}	$1,43\%$

Source: Author

Comparing the results presented in Table 3 was verified that the coarse grid can be used in NACA 0021 case analysis due to GCI_{21} is less than 1% and GCI_{32} being less than 2% and approximate relative error below 5%.

Comparing the curves in Figure 19 was verified that the meshes with $y^+ = 15$ and $y^+ = 60$ almost fit with the experimental data, only $y^+ = 150$ mesh presented little difference, diverging from the other meshes next to the C_p peak.

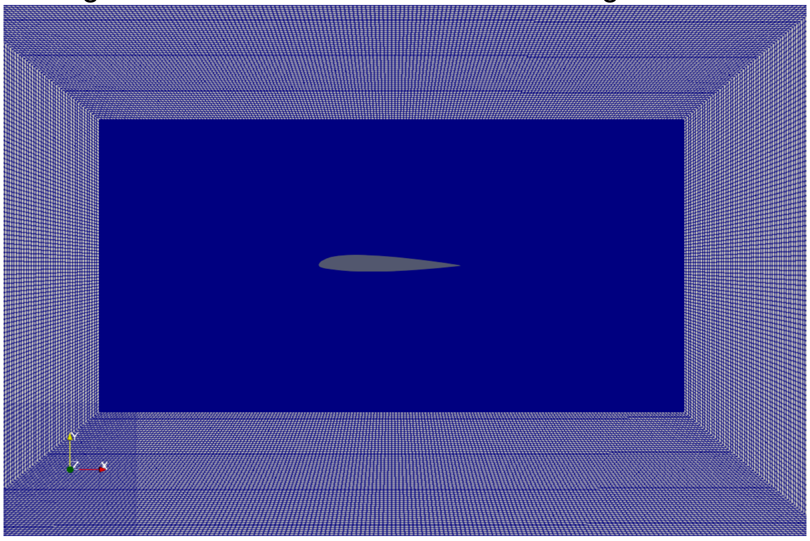
4.3 Analysis of NACA 23012 airfoil

Following the proposed methodology was studied the NACA 23012 airfoil, the results of the mesh generation, and evaluated in numerical simulation of the case are presented below.

4.3.1 Mesh

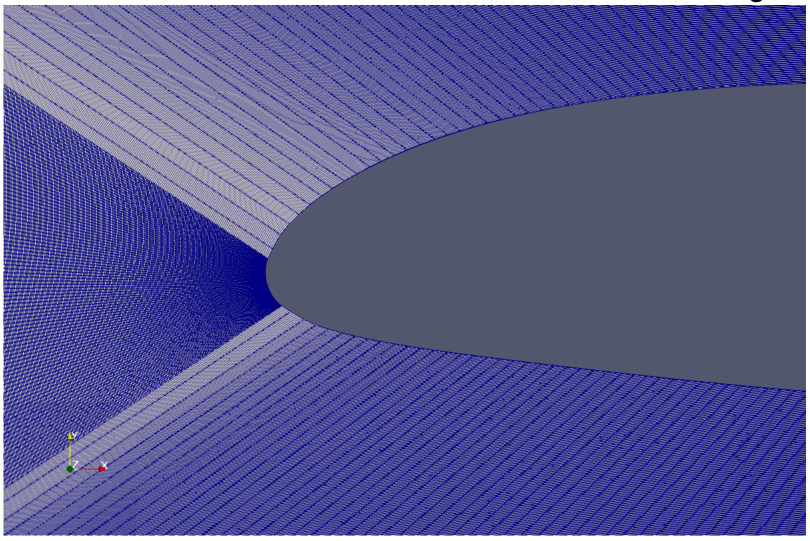
Using the points of a NACA 23012 and the method explained in the methodology, the far field has a size 20 times bigger than the airfoil chord. The mesh disposed in the Figures 20 and 21 was generated for an y^+ value of 80 , with approximately 2 millions hexahedral elements, since OpenFOAM operates with a three-dimensional mesh, even for two-dimensional cases, where the third dimension only has 1 element. Other two meshes with different values of y^+ were generated for GCI analysis.

Figure 20: Demonstration of the mesh generated.



Source: Author

Figure 21: Demonstration of the mesh around the leading edge.

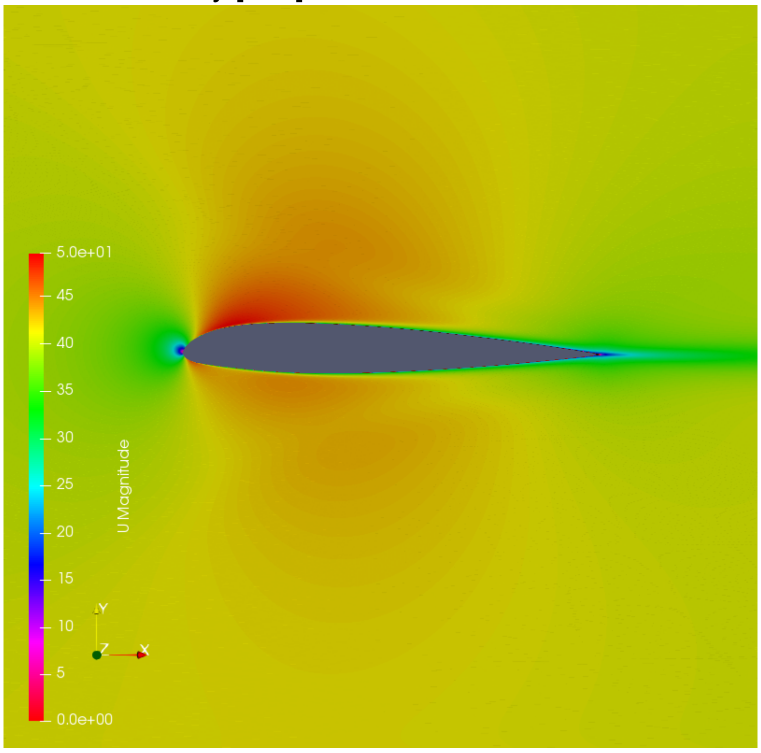


Source: Author

4.3.2 Numerical Results

Figure 22 presents the results evaluated for velocity fields obtained for NACA 23012 :

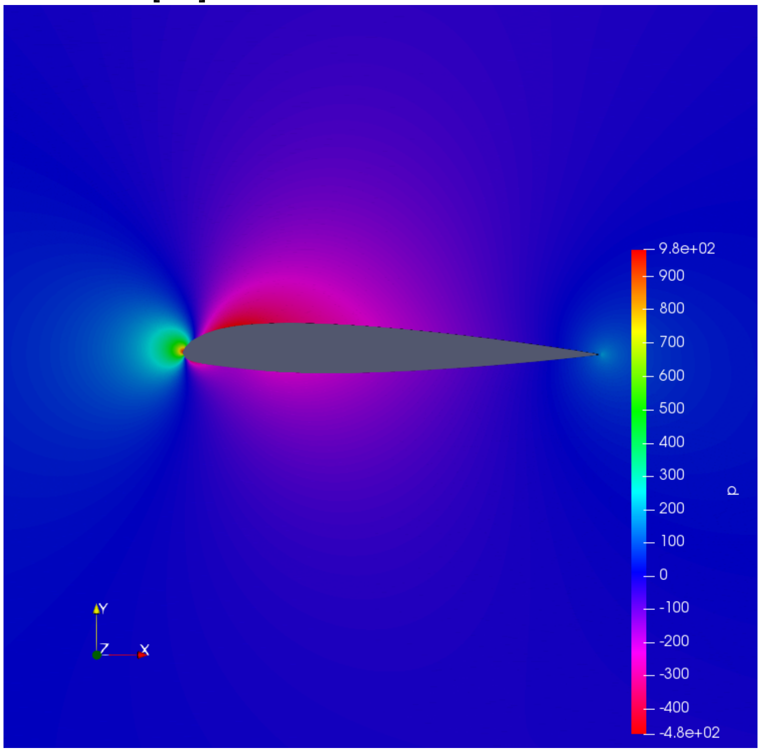
Figure 22: Magnitude of Velocity [m/s] data from the flow field around the NACA 23012.



Source: Author

Figure 23 presents results for pressure fields obtained:

Figure 23: Pressure [Pa] data from the flow field around the NACA 23012.

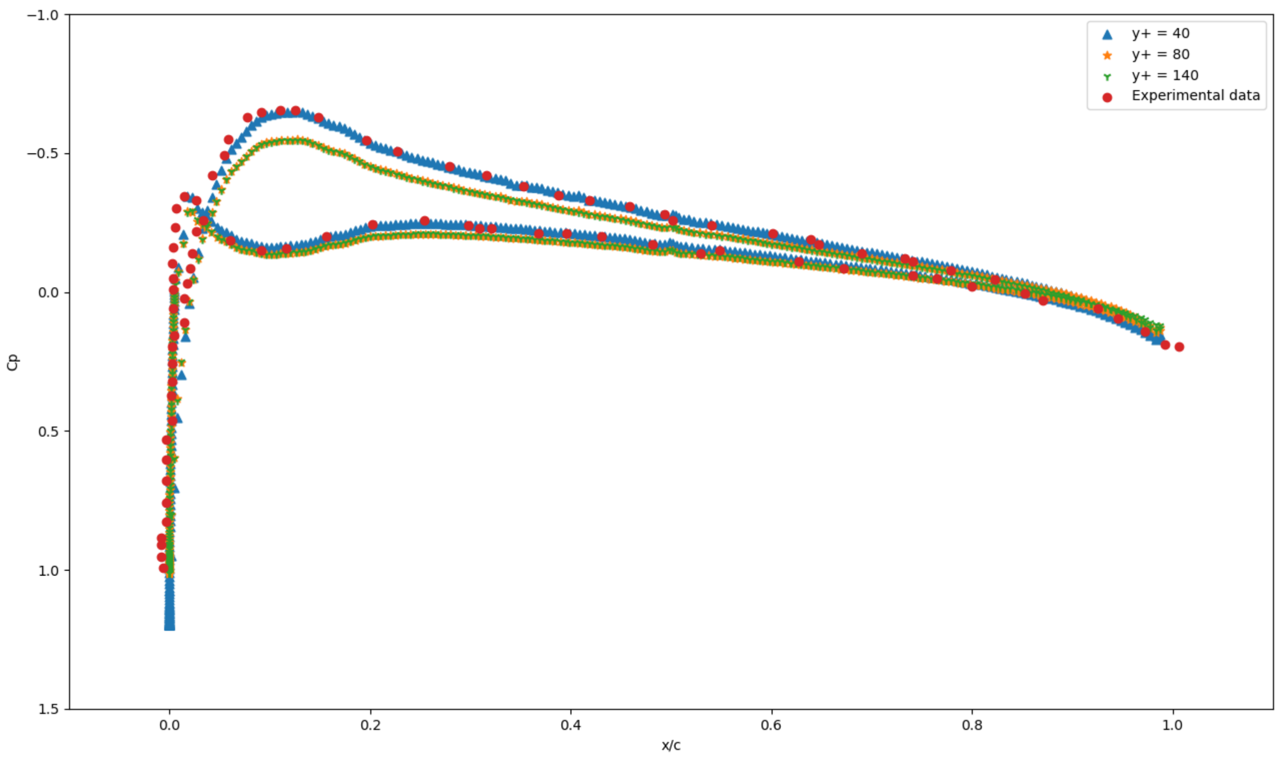


Source: Author

From Figures 22 and 23 present a asymmetrical airfoil at 0°angle of attack, shows that the flow stream is asymmetric and the velocity of the upper and lower surface of the airfoil are different. Therefore, it can be said that the pressure at the upper surface of the airfoil is less than the lower surface airfoil since the upper surface velocity is higher than the velocity at lower surface, pressure difference between the lower and upper body of the airfoil results in lift force as expected from the literature.

Figure 24 presents the C_p comparison between the experimental data provided by Pan & Loth (2003) and the numerical data evaluated with the analysis over airfoil NACA 23012.

Figure 24: Comparison of the C_p evaluated with the numerical analysis with the experimental data provided by Pan & Loth (2003) for NACA 23012 airfoil.



Source: Author

As can be seen in Figure 24, the numerical data evaluated in the analysis keeps the same behavior of the experimental data presented by Pan & Loth (2003). However, the numerical results diverge slightly from the values found by the experiments.

Another issue found in the NACA 23012 case was the representation of the flow over the leading edge. In this region, the numerical data presented a significant difference from the experimental values of C_p from the meshes with higher y^+.

Differing from the NACA0012 test case, where was found a problem in the representation of the flow over the trailing edge, the numerical analysis was capable of representing the flow over the trailing edge as the NACA0021 case, fitting experimental and numerical data in this region.

Following the procedure of Celik et al. (2008), was verified the GCI for the NACA 23012 test case generated meshes with C_p as the chosen parameter, the results evaluated are presented in Table 4.

Table 4: GCI analysis for NACA 23012 test case.

Number of Elements in mesh 1	4238400
Number of elements in mesh 2	2119200
Number of elements in mesh 3	1210200
Refinement factor r_{21}	$1,4142$
Refinement factor r_{32}	$1,3233$
Approximate relative error e_{a21}	$15,54\%$
Approximate relative error e_{a32}	$0,19\%$
Extrapolated relative error e_{ex21}	$0,35\%$
Extrapolated relative error e_{ex32}	$0,009\%$
Convergence index GCI_{21}	$0,44\%$
Convergence index GCI_{32}	$0,012\%$

Source: Author

Comparing the results presented in Table 4 was verified that the finest grid needs to be used in NACA 23012 case analysis due to approximate relative error being 15% even with GCI_{21} less than 1%.

Comparing the curves in Figure 19 was verified that the mesh with $y^+ = 40$ almost fit with the experimental data, while $y^+ = 80$ and $y^+ = 140$ mesh presented some difference, diverging from the other meshes next to the C_p peak.

5 Final Consideration

5.1 Conclusion

The main goal of this research was to develop an algorithm to automatically generate a structured mesh with airfoil points as input, using the simulation results and experimental values as parameters to validate the method.

The results obtained in symmetrical airfoils using the structured mesh generated by the method proposed have an error with the literature of approximately 0.9% for NACA 0012 and 2.1% for NACA 0021 results in the leading edge. The CGI analysis indicates that the quantity of elements in the mesh does not need to be high for the same results, being the number for symmetrical cases approximately 1 to 2 million elements, having results with great accuracy.

For the asymmetrical airfoil an error of approximately 0.81% for NACA 23012 results agrees with the literature in the leading edge. The CGI analysis show a quantity of 4 million elements in the mesh generate excellent results.

The results presented demonstrate the versatility and quality of the mesh, proving the efficacy of the method and good cost benefit since it uses only free software.

5.2 Future study

Suggestions for future works to amplify the understanding of the topics approached in this work:

- Amplify the number of cases;
- Generate a mesh for an airfoil with a higher camber;
- Develop an automatic mesh generation for wings;
- Generate other files for the simulation on OpenFOAM (controlDict, transportProperties, etc.);
- Develop better integration between Python and OpenFOAM to automate all process.

6 References

Beaufort, P.-A., Geuzaine, C. & Remacle, J.-F. (2020), ‘Automatic surface mesh generation for discrete models – a complete and automatic pipeline based on reparametrization’, *Journal of Computational Physics* **417**, 109575.
URL: *http://www.sciencedirect.com/science/article/pii/S0021999120303491*

Benoit, C. & Péron, S. (2012), ‘Automatic structured mesh generation around two-dimensional bodies defined by polylines or polyc1 curves’, *Computers & Fluids* **61**, 64 – 76. Onera Scientific Day.

Celik, I., Ghia, U., Roache, P., Freitas, C., Coleman, H. & Raad, P. (2008), ‘Procedure for estimation and reporting of uncertainty due to discretization in cfd applications’, *Journal of Fluids Engineering* **130**.

Faghih-Naini, S., Kuckuk, S., Aizinger, V., Zint, D., Grosso, R. & Köstler, H. (2020), ‘Quadrature-free discontinuous galerkin method with code generation features for shallow water equations on automatically generated block-structured meshes’, *Advances in Water Resources* **138**, 103552.
URL: *http://www.sciencedirect.com/science/article/pii/S0309170819310735*

González, A. O., Vallier, A. & Nilsson, H. (2009), ‘Mesh motion alternatives in openfoam’.

Jespersen, D. C., Pulliam, T. H. & Childs, M. L. (2016), ‘Overflow turbulence modeling resource validation results’, *NAS Technical Report: NAS-2016-01* .

Liu, C. & Hu, C. (2018), ‘Block-based adaptive mesh refinement for fluid–structure interactions in incompressible flows’, *Computer Physics Communications* **232**, 104 – 123.

Lu, F., Pang, Y., Jiang, X., Sun, J., Huang, Y., Wang, Z. & Ju, J. (2018), ‘Automatic generation of structured multiblock boundary layer mesh for aircrafts’, *Advances in Engineering Software* **115**, 297 – 313.

Lu, F., Qi, L., Jiang, X., Liu, G., Liu, Y., Chen, B., Pang, Y. & Hu, X. (2020), ‘Nnw-gridstar: Interactive structured mesh generation software for aircrafts’, *Advances in Engineering Software* **145**, 102803.
URL: *http://www.sciencedirect.com/science/article/pii/S0965997819306891*

Moukalled, F., Mangani, L. & Darwish, M. (2015), *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM and Matlab*, 1st edn, Springer Publishing Company, Incorporated.

OpenFoam (2020), ‘Openfoam user guide’.
URL: *https://www.openfoam.com/documentation/user-guide/*

OpenFOAM Foundation (2020), ‘Openfoam v7 homepage’.
URL: *https://openfoam.org/version/7/*

Pan, J. & Loth, E. (2003), ‘Effect of airfoil geometry on performance with simulated ice accretions volume 2: Numerical investigation’, http://www.tc.faa.gov/its/worldpac/techrpt/ar03-65.pdf.

Pantaleão, A. V., Andrade, C. R. & Zaparoli, E. L. (2003), ‘A review of the delaunay mesh generation for heat transfer finite element analysis’, *COBEM 2003* .

Rumsey, C. (2019), ‘2dn00: 2d naca 0012 airfoil validation case’, https://go.nasa.gov/37efp9v.

Schmidt, J., Peralta, C. & Stoevesandt, B. (2012), ‘Automated generation of structured meshes for wind energy applications’, *Open Source CFD International Conference, London* .

Wolfe, W. P. & Ochs, S. S. (1997), ‘Predicting aerodynamic characteristics of typical wind turbine airfoils using cfd’, http://www.tc.faa.gov/its/worldpac/techrpt/ar03-65.pdf.

Yu, F., Zhang, H., Class, A., Xiao, J., Travis, J. R. & Jordan, T. (2019), ‘Winding number based automatic mesh generation algorithm for hydrogen analysis code gasflow-mpi’, *International Journal of Hydrogen Energy* **44**(26), 14070 – 14084.
URL: *http://www.sciencedirect.com/science/article/pii/S036031991931290X*

Zhang, Y. & Jia, Y. (2018), ‘2d automatic body-fitted structured mesh generation using advancing extraction method’, *Journal of Computational Physics* **353**, 316 – 335.

Appendix

In this appendix the developed algorithm is presented.

Code

Pedro Henrique de Araújo Bitencourt

October 27, 2021

```
[ ]: %matplotlib
     import string
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     '''Este programa serve para criar um arquivo de BlockMesh do perfil inserido
     Autor: Pedro Henrique de Araújo Bitencourt'''
```

1 Airfoil Input

```
[ ]: airfoil = input('Airfoil name')
     perf = airfoil+'.dat'
     perfil = np.genfromtxt(perf,skip_header=1)
```

```
[ ]: plt.figure()
     plt.axis([-0.1, 1.1, -0.5, 0.5])
     plt.plot(perfil[:,0], perfil[:,1])
     perfil = np.concatenate((perfil,np.zeros((perfil.shape[0],1))),axis=1)
```

2 y+ calculation

```
[ ]: y_plus = eval(input('y+ value'))
     Mach = eval(input('Mach value'))
     rho = eval(input('rho value'))
     mi = eval(input('mi value'))
     L = eval(input('Chord value'))
     u = 0.12*Mach
     Re = rho*u*L/mi
     #Se Re < 10
     Cf = (2*np.log10(Re)-0.65)**-2.3
     tal = Cf*0.5*rho*u**2
     u_star = (tal/rho)**0.5
     y = y_plus*mi/(rho*u_star)
```

3 Far Field Calculation

[ ]:
```
lines =[]



Lu = -25*L          #Upstream length from tip of leading edge

Ld = 25*L

Hy = 20*L         #Height of computational domain perpendicular to flow

Y1 = Hy*0.2

Y2 = Hy*0.1

Y3 = Hy*0.05

Zw = 1

Y4 = -1.00*Y1

Y5 = -1.00*Y2

Y6 = -1.00*Y3

Hd = -1.00*Hy

X1 = -0.20

X2 = 0.50

X3 = 1.25

nx1 = 200

nx2 = 100

nx3 = 100

ny1 = np.int(Y2/y)

ny2 = np.int(ny1/3)

nz = 1

nd = 200
```

```
NLE = 100

NTE = 100

n1  = 100
```

4 File writing

```
[ ]: t = 'blockMeshDict'
     mesh = open(t,'w')
```

```
[ ]: cabe = '''/*--------------------------------*- C++␣
      ↪-*----------------------------------*\
     | =========                 |                                                 |
     | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
     |  \\    /   O peration     | Version:  1606+                                 |
     |   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
     |    \\/     M anipulation  |                                                 |
     \*---------------------------------------------------------------------------*/
     FoamFile
     {
       version     2.0;
       format      ascii;
       class       dictionary;
       object      blockMeshDict;
     }
     // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

     convertToMeters 1; \n'''

     mesh.write(cabe),mesh.flush()
```

```
[ ]: def apen(v):
         resp = '\n ('+str(np.format_float_positional(v[0]))+' '+str(np.
      ↪format_float_positional(v[1]))+' '+str(np.format_float_positional(v[2]))+')'
         return  resp
```

```
[ ]: ident = int((perfil.shape[0]-1))
     ident0 = int(ident/2-1)
     ident1 = int(0.45*(ident))
     ident2 = int(0.25*(ident))
     ident3 = int(ident/2+1)
     ident4 = int(0.56*ident)
     ident5 = int(0.76*ident)
```

4.1 Vertices

[ ]:
```
lines =[]
lines.append('\n vertices \n ( \n')
  # Upper surface
v0 = (perfil[ident0][0],   perfil[ident0][1],    0.000)         # 0
lines.append(apen(v0)) #0

v2 = (perfil[ident2][0],   perfil[ident2][1],    0.000)         # 1
lines.append(apen(v2))  #1
v3 = (perfil[1][0],   perfil[1][1],    0.000)         # 2
lines.append(apen(v3)) #2
# Lower Surface
v4 = (perfil[ident3][0],   perfil[ident3][1],    0.000)        # 3
lines.append(apen(v4)) #3

v6 = (perfil[ident5][0],   perfil[ident5][1],    0.000)        # 4
lines.append(apen(v6))#4
v7 = (perfil[-2][0],   perfil[-2][1],    0.000)       # 5
lines.append(apen(v7))#5

v8 = (-1.5*L,  L,    0.000)       # 6
lines.append(apen(v8))#6
v9 = (0.5*L,   L,    0.000)       # 7
lines.append(apen(v9))#7
v10 = (2.5*L,  L,    0.000)       # 8
lines.append(apen(v10))#8
v11 = (-1.5*L,   -L,    0.000)       #9
lines.append(apen(v11))#9
v12 = (0.5*L,  -L,    0.000)       # 10
lines.append(apen(v12))#10
v13 = (2.5*L,   -L,    0.000)       # 11
lines.append(apen(v13))#11


v14 =   (Lu,   Hy,   0.000)                  #  12
lines.append(apen(v14)) #12

v16 =   (X2,   Hy,   0.000)                  #  13
lines.append(apen(v16)) #13

v18 =   (Ld,  Hy,   0.000)                  #  14
lines.append(apen(v18)) #14


v25 =   (Lu,   Hd,   0.000)                  #  15
lines.append(apen(v25)) #15
```

```
v27 =   (X2,   Hd,   0.000)                    #  16
lines.append(apen(v27)) #16

v29 =   (Ld,   Hd,   0.000)                    #  17
lines.append(apen(v29)) #17

v30 =   (perfil[ident0][0],   perfil[ident0][1],     Zw )        # 18
lines.append(apen(v30)) #18

v32 =   (perfil[ident2][0],   perfil[ident2][1],     Zw )        # 19
lines.append(apen(v32)) #19
v33 =   (perfil[1][0],   perfil[1][1],     Zw )        # 20
lines.append(apen(v33)) #20

v34 =   (perfil[ident3][0],   perfil[ident3][1],    Zw )        # 21
lines.append(apen(v34)) #21

v36 =   (perfil[ident5][0],   perfil[ident5][1],    Zw)         # 22
lines.append(apen(v36)) #22
v37 =   (perfil[-2][0],   perfil[-2][1],    Zw )        # 23
lines.append(apen(v37)) #23

v38 = (-1.5*L,  L,    Zw )       # 24
lines.append(apen(v38))#24
v39 = (0.5*L,   L,    Zw )       # 25
lines.append(apen(v39))#25
v40 = (2.5*L,  L,    Zw )       # 26
lines.append(apen(v40))#26
v41 = (-1.5*L,   -L,    Zw )       # 27
lines.append(apen(v41))#27
v42 = (0.5*L,  -L,    Zw )       # 28
lines.append(apen(v42))#28
v43 = (2.5*L,   -L,    Zw)       # 29
lines.append(apen(v43))#29


v44 =   (Lu,   Hy,   Zw)                    # 30
lines.append(apen(v44)) #30

v46 =   (X2,   Hy,   Zw)                    # 31
lines.append(apen(v46)) #31

v48 =   (Ld,   Hy,   Zw)                    # 32
lines.append(apen(v48)) #32


v55 =   (Lu,   Hd,   Zw)                    # 33
```

```
lines.append(apen(v55)) #33

v57 =   (X2,   Hd,   Zw)                      # 34
lines.append(apen(v57)) #34

v59 =   (Ld,   Hd,   Zw)                      # 35
lines.append(apen(v59)) #35

lines.append(');')

mesh.writelines(lines),mesh.flush()
```

4.2 Edges

[ ]:
```
def apen_spline(spline):
    resp = ''
    for i in range(spline.shape[0]):
          resp += '\n ('+str(np.format_float_positional(spline[i,0]))+' '+str(np.
 ↪format_float_positional(spline[i,1]))+' '+str(np.
 ↪format_float_positional(spline[i,2]))+')'
    return  resp
```

[ ]:
```
lines = []
lines.append('\n edges \n ( \n')
    # Upper body (upper circular arcs).
arc_0_4 = np.array([0, 0, 0])
lines.append('\n arc 0 3 ('+str(arc_0_4[0])+' '+str(arc_0_4[1])+'␣
 ↪'+str(arc_0_4[2])+')')

arc_30_34 = np.array([0, 0,  Zw])
lines.append('\n arc 18 21  ('+str(arc_30_34[0])+' '+str(arc_30_34[1])+'␣
 ↪'+str(arc_30_34[2])+')')



# Upper body longitudinal splines.
spline_0_1 = perfil[ident0:ident2:-1,:]
spline_0_1[:,2] = 0

lines.append('\n spline 0 1 ('+apen_spline(spline_0_1)+')')
spline_30_31 = spline_0_1
spline_30_31[:,2] = Zw
lines.append('\n spline 18 19 ('+apen_spline(spline_30_31)+')')



spline_2_3 = perfil[ident2:0:-1,:]
lines.append('\n spline 1 2 ('+apen_spline(spline_2_3)+')')
spline_32_33 = spline_2_3
spline_32_33[:,2] = Zw
lines.append('\n spline 19 20 ('+apen_spline(spline_32_33)+')')

spline_4_5 = perfil[ident3:ident5,:]
lines.append('\n spline 3 4 ('+apen_spline(spline_4_5)+')')
spline_34_35 = spline_4_5
spline_34_35[:,2] = Zw
lines.append('\n spline 21 22 ('+apen_spline(spline_34_35)+')')


spline_6_7 = perfil[ident5:-1,:]
lines.append('\n spline 4 5 ('+apen_spline(spline_6_7)+')')
spline_36_37 =  spline_6_7
spline_36_37[:,2] = Zw
lines.append('\n spline 22 23 ('+apen_spline(spline_36_37)+')')

lines.append('\n ); \n')

mesh.writelines(lines),mesh.flush()
```

4.3 Blocks

[ ]:
```
def apen_block(blo):
    resp = '\n hex '
    cont = 0
    for j in blo:
        resp += '('
        for i in j:
            resp += str((i))+' '
        cont +=1
        if cont==2:
            resp += ') simpleGrading '
        else:
            resp += ') '
    return  resp
```

[ ]:
```
lines =[]
lines.append('\n blocks \n ( \n')
simpleGrading6 = [1, 1, 1]
hex_1 = np.array([[3, 0, 6, 9, 21, 18, 24, 27],[nx2, ny1, nz], simpleGrading])
lines.append(apen_block(hex_1))
hex_2 = np.array([[0, 1, 7, 6, 18, 19, 25, 24],[nx2, ny1, nz], simpleGrading])
lines.append(apen_block(hex_2))
hex_3 = np.array([[1, 2, 8, 7, 19, 20, 26, 25], [nx2,  ny1,  nz], ␣
 ↪simpleGrading])
lines.append(apen_block(hex_3))
hex_4 = np.array([[2, 5, 11, 8, 20, 23, 29, 26], [nx2,  ny1,  nz], ␣
 ↪simpleGrading])
lines.append(apen_block(hex_4))
hex_5 = np.array([[10, 11, 5, 4, 28, 29, 23, 22],[nx2, ny1, nz], simpleGrading])
lines.append(apen_block(hex_5))
hex_6 = np.array([[9, 10, 4, 3, 27, 28, 22, 21],[nx2, ny1, nz], simpleGrading])
lines.append(apen_block(hex_6))

hex_7 = np.array([[9, 6, 12, 15, 27, 24, 30, 33],[nx2, ny2, nz], simpleGrading])
lines.append(apen_block(hex_7))
hex_8 = np.array([[6, 7, 13, 12, 24, 25, 31, 30],[nx2, ny2, nz], simpleGrading])
lines.append(apen_block(hex_8))
hex_9 = np.array([[7, 8, 14, 13, 25, 26, 32, 31], [nx2,  ny2,  nz], ␣
 ↪simpleGrading])
lines.append(apen_block(hex_9))
hex_10 = np.array([[8, 11, 17, 14, 26, 29, 35, 32], [nx2,  ny2,  nz], ␣
 ↪simpleGrading])
lines.append(apen_block(hex_10))
hex_11 = np.array([[16, 17, 11, 10, 34, 35, 29, 28],[nx2, ny2, nz],␣
 ↪simpleGrading])
lines.append(apen_block(hex_11))
hex_12 = np.array([[10, 9, 15, 16, 28, 27, 33, 34],[nx2, ny2, nz],␣
 ↪simpleGrading])
lines.append(apen_block(hex_12))



lines.append('\n ); \n')

mesh.writelines(lines),mesh.flush()
```

4.4 Boundary

[ ]:
```
def apen_faces(face):
    resp = '\n faces ( \n'
    for i in range(face.shape[0]):
          resp += '('+str(int(face[i,0]))+'  '+str(int(face[i,1]))+' ␣
 ↪'+str(int(face[i,2]))+'  '+str(int(face[i,3]))+')'
    resp += '\n ); \n'
    return  resp
```

[ ]:
```
lines =[]
lines.append('\n boundary \n ( \n')

#inlet
lines.append('\n inlet{ \n')
```

```
type_inlet = 'type patch;'
lines.append(type_inlet)
faces_inlet = np.array([[15, 12, 30, 33]])

lines.append(apen_faces(faces_inlet))
lines.append(' }')

#outlet
lines.append('\n outlet{ \n')
type_outlet = 'type patch;'
lines.append(type_outlet)
faces_outlet = np.array([[14, 17, 35, 32]])
lines.append(apen_faces(faces_outlet))
lines.append(' }')

#wall
lines.append('\n walls{ \n')
type_aerofoil = 'type wall;'
lines.append(type_aerofoil)
faces_aerofoil = np.array([
    [0, 1, 19, 18], [1, 2, 20, 19], [2, 5, 23, 20],
    [5, 4, 22, 23], [4, 3, 21, 22], [3, 0, 18, 21]])
lines.append(apen_faces(faces_aerofoil))
lines.append(' }')


#topAndBottom
lines.append('\n topAndBottom{ \n')
type_upper = 'type wall;'
lines.append(type_upper)
faces_upper = np.array([
    [12, 13, 31, 30], [13, 14, 32, 31], [15, 16, 34, 33],
    [16, 17, 35, 34]])
lines.append(apen_faces(faces_upper))
lines.append(' }')

#frontandback
lines.append('\n frontAndBack{ \n')
type_lower = 'type empty;'
lines.append(type_lower)
faces_lower = np.array([
    [0, 1, 7, 6], [3, 0, 6, 9], [1, 2, 8, 7], [5, 2, 8, 11],
    [5, 4, 10, 11], [4, 3, 9, 10], [6, 7, 13, 12], [9, 6, 12, 15],
    [7, 8, 14, 13], [8, 11, 17, 14], [11, 10, 16, 17], [10, 9, 15, 16],

    [18, 19, 25, 24], [21, 18, 24, 27], [19, 20, 26, 25], [23, 20, 26, 29],
    [23, 22, 28, 29], [22, 21, 27, 28], [24, 25, 31, 30], [27, 24, 30, 33],
    [25, 26, 32, 31], [26, 29, 35, 32], [29, 28, 34, 35], [28, 27, 33, 34]])
lines.append(apen_faces(faces_lower))
lines.append('\n }); \n')

mesh.writelines(lines),mesh.flush()

end = '''mergePatchPairs ();
// ************************************************************************* //
 ↪'''
mesh.write(end),mesh.flush()
```

5 Data analysis

```
[ ]: def data_analysis_graph(File):
         Cp = File+'.csv'
         file = open(Cp)
         data = pd.read_csv(Cp)
         file.close()
         Cp_Data = np.array(data.loc[data['z']>0,'Cp'])
         x_Data = np.array(data.loc[data['z']>0,'x'])
         return Cp_Data,x_Data
```

```
[ ]: Cp_Simu1, x_Simu1 = data_analysis_graph('Cp_Naca23012_y+40')
     Cp_Simu2, x_Simu2 = data_analysis_graph('Cp_Naca23012_y+80')
     Cp_Simu3, x_Simu3 = data_analysis_graph('Cp_Naca23012_y+140')
     Cp_Teoric, x_Teoric = data_analysis_graph('Cp_Naca23012_Teorico')


     plt.figure()
     plt.axis([-0.1, 1.1, 1.5, -1.5])
     plt.scatter(x_Simu1, Cp_Simu1, marker='^', s=40)
     plt.scatter(x_Simu2, Cp_Simu2, marker='*')
     plt.scatter(x_Simu3, Cp_Simu3, marker='1')
     plt.scatter(x_Teoric, Cp_Teoric)
     plt.xlabel('x/c')
     plt.ylabel('Cp')
     #plt.title('Naca23012')
     plt.legend(['y+ = 40', 'y+ = 80', 'y+ = 140', 'Experimental data'])
```