MARMARA UNIVERSITY

FACULTY OF ENGINEERING

CREATING A PYTHON BASED SOFTWARE FOR OPTIMIZING DELTA WING PARAMETERS USING OPEN SOURCE PROGRAMS

Umut Alpay, Selçuk Özcan, Erdem Aktürk, Umut Deniz Özyurt

GRADUATION PROJECT REPORT

Department of Mechanical Engineering

Supervisor

Prof. Dr. Emre ALPMAN

ISTANBUL, 2025

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF MECHANICAL ENGINEERING

2024-2025 FALL

ME 4097-ENGINEERING PROJECT I

FINAL THESIS FALL REPORT

Name	Surname	Student Number
Umut	ALPAY	150420020
Erdem	AKTÜRK	150420013
Selçuk	ÖZCAN	150420016
Umut Deniz	ÖZYURT	150421001

Project Topic: CREATING A PYTHON BASED SOFTWARE FOR OPTIMIZING DELTA WING PARAMETERS USING OPEN SOURCE PROGRAMS

Academic Advisor: Prof.Dr. Emre ALPMAN

ACKNOWLEDGEMENT

We would like to thank our supervisor Prof. Dr. Emre ALPMAN for the valuable guidance and advice on preparing this thesis.

June, 2025 Umut Alpay, Selçuk Özcan, Erdem Aktürk, Umut Deniz Özyurt

Contents

ABSTRACT

This thesis focused on performing optimization of various delta wing profiles in both supersonic flow condition and off-design performance using python-based coding. During the fall term, research about aerospace literature, delta wing profiles and related softwares which are used in aerospace industry is completed. Open-source software was preferred. FreeCAD software has been used to draw delta wing profiles and aerofoils. Additionally, macro modules were enhancing the design process. It is aimed to provide creating various delta wing profiles in terms of quality and quantity in a short time using macro modules. After the design process is completed, meshing were done by GMSH. After the meshing is completed, the aerodynamic analysis of the wing profiles is performed with SU2 open-source program. Weighted average of results for each sample is calculated and the best sample were selected as the most optimized one.

Bu tez, Python tabanlı kodlama kullanarak hem süpersonik akış koşulunda hem de tasarım dışı performansta çeşitli delta kanat profillerinin optimizasyonunu gerçekleştirmeye odaklanmıştır. Güz döneminde havacılık literatürü, delta kanat profilleri ve havacılık endüstrisinde kullanılan ilgili yazılımlar hakkında araştırma tamamlanmıştır. Açık kaynaklı yazılım tercih edilmiştir. Delta kanat profilleri ve aerofoilleri çizmek için FreeCAD yazılımı kullanılmıştır. Ayrıca, makro modüller tasarım sürecini geliştirmiştir. Makro modüller kullanılarak kısa sürede nitelik ve nicelik açısından çeşitli delta kanat profillerinin oluşturulması amaçlanmıştır. Tasarım süreci tamamlandıktan sonra, GMSH tarafından hücreleme yapılmıştır. Hücreleme tamamlandıktan sonra, kanat profillerinin aerodinamik analizi SU2 açık kaynaklı programı ile gerçekleştirilir. Her örnek için sonuçların ağırlıklı ortalaması hesaplanır ve en iyi örnek en optimize edilmiş örnek olarak seçilir.

SYMBOLS

V = Free stream velocity

Ma = Mach number

C_D = Drag coefficient

C_D = Lift coefficient

α = Speed of sound

c = Chord length

b = Wingspan

θ_{sweep} = Sweeping angle

$\theta_{trailing}$ = Trailing angle

t = thickness of wing at the root [m]

Δ = distance between leading edge and trailing edge at the tip of the wing [m]

ABBREVIATIONS

2D	Two dimensional
3D	Three dimensional
API	Application Programming Interface
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy
GUI	General User Interface
HVAC	Heating Ventilating and Air Conditioning
LHS	Latin Hyper Cube
SU2	Stanford University Unstructured
QMC	Quasi-Monte Carlo

LIST OF FIGURES

1. INTRODUCTION

The optimization of delta wing design parameters has been a significant area of study in aerospace engineering, particularly for aircraft operating in supersonic and hypersonic regimes. Delta wings, characterized by their triangular shape and high sweep angles, provide critical aerodynamic advantages, such as reduced drag and improved high-speed stability. However, optimizing their design requires careful consideration of various geometric parameters, including aerofoil shape, sweep angle, and thickness distribution.

This study focuses on utilizing Python-based computational tools to optimize delta wing designs by leveraging parametric modelling and computational fluid dynamics (CFD) simulations. The project aims to develop an automated workflow integrating FreeCAD for geometry generation, SU2 for aerodynamic analysis, and Latin Hypercube Sampling (LHS) for efficient design space exploration. Through this approach, a range of delta wing configurations will be analysed to identify optimal designs that enhance performance design and off-design conditions.

The research methodology incorporates an extensive review of aerodynamic principles, advanced computational modelling techniques, and experimental design strategies to systematically evaluate different wing configurations. By employing open-source software, this project seeks to provide a cost-effective and scalable solution for optimizing delta wing designs, making it accessible for further academic and industrial applications.

This report outlines the theoretical background, methodologies employed, and initial steps completed during the optimization process. It also discusses potential challenges, future improvements, and alternative approaches to refining delta wing configurations for enhanced aerodynamic performance.

2. THEORETICAL BACKGROUND & METHODOLOGY

2.1. Flow Theory

In fluid mechanics, understanding the different types of flow is essential for analyzing, modeling, and optimizing fluid behavior across engineering systems. At the airflow around an aircraft wing, each flow exhibits distinct physical characteristics that influence pressure distribution, heat transfer, drag, and system stability.

The concept of flow can be classified according to different parameters such as time (steady - unsteady), viscosity (viscous - inviscid), flow nature (laminar - turbulent - transitional), compressibility and boundary condition (internal flow - external flow). In this study, while examining the optimization of delta wing profiles, it was preferred to proceed with a classification according to the "Mach Number" parameter for the purpose of analyzing the performance in flight conditions.

2.1.1. Subsonic Flow:

Subsonic flow refers to a flow regime where the fluid velocity is less than the speed of sound in the given medium. This corresponds to a Mach number (M) less than 1, where:

$$Mach\ Number(Ma) = \frac{V}{\alpha}$$

$V =$ Local flow velocity

$\alpha =$ Local speed of sound, which depends on temperature and fluid properties

Subsonic flow has some unique characteristics. In terms of compressibility, for M < 0.3 compressibility effects are negligible and the flow is generally considered to be incompressible. However, between M = 0.3 and 1 density changes become significant and compressible flow equations may be needed.

To evaluate the relationship between pressure and speed; in subsonic flow through a converging nozzle or duct, a decrease in area causes an increase in velocity, and hence a decrease in pressure (according to Bernoulli’s principle).

Pressure waves (sound waves) can travel upstream, meaning flow disturbances can affect upstream conditions, unlike in supersonic flow.

If evaluated in terms of stability and predictability, subsonic flows are generally more stable, easier to control, and less prone to instabilities such as shock waves.

Depending on Reynolds' number, the flow may be laminar or turbulent, but both can exist within subsonic regimes.

Commercial aircrafts (e.g., Airbus A320 or Boeing 737), HVAC Systems, automobile aerodynamics and pipeline flows operate at subsonic speeds.

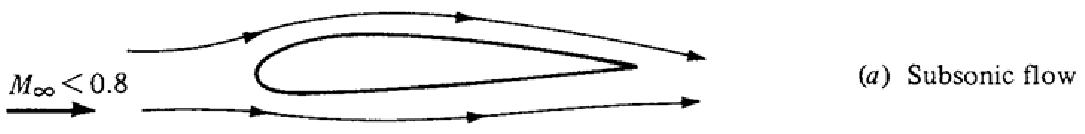


Figure 1: Subsonic Flow (Anderson, 2017).

2.1.2. Transonic Flow:

Transonic flow is the flow regime in which the Mach number of the fluid varies between approximately 0.8 and 1.2.

$$0.8 \leq Ma \leq 1.2$$

In transonic flow, the flow may be subsonic in some regions and supersonic in others. This is usually a complex flow regime due to the mixed structure.

In transonic flow, "local shock waves" occur when the speed is close to Mach 1. These shocks are usually thin, unstable, and produce severe pressure drops and energy loss in the flow. Due to shock waves, pressure increases suddenly, which can cause separation of the flow from the surface. This increases the friction force. A strong interaction occurs between these shock waves and the boundary layer. This interaction can destabilize the flow. Small geometric changes or velocity changes can create large differences in the flow field.

Most commercial jet aircraft (e.g., Boeing 737, Airbus A320) travel in the transonic Mach range (M ≈ 0.82–0.86). For this reason, airfoils are specially designed to reduce transonic effects.

The air passages in jet engines, especially turbojet and ramjet engines, include this regime.

The numerical solution of transonic flows is difficult because it involves both subsonic and supersonic regions simultaneously and requires complex algorithms and mesh structures.

Additionally, some aerodynamic difficulties can be mentioned for a complex flow structure such as transonic flow.

It is the free stream Mach number at the point where the wing surface first reaches Mach = 1. Transonic flow begins after this point. Immediately after this point, the drag force suddenly increases. This effect is known as "drag divergence" and is a critical limit for flight performance.

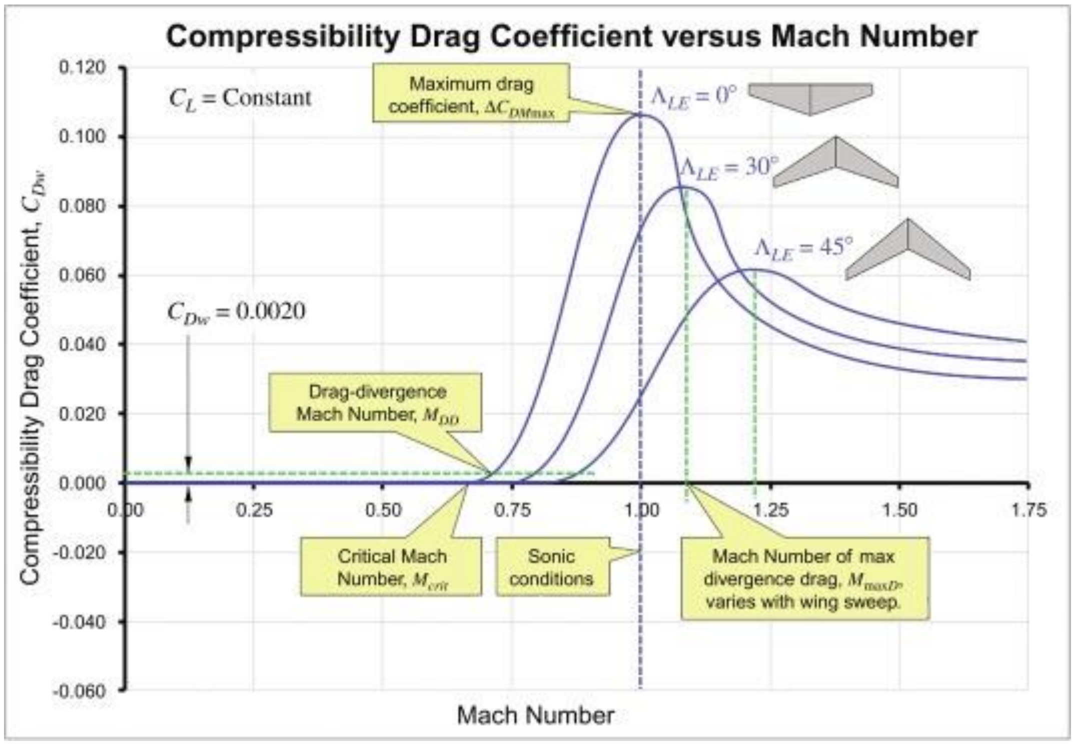


Figure 2: Drag Coefficient and Mach Number Relationship (Çengel & Cimbala, 2020).

2.1.3. Supersonic Flow:

Supersonic flow is a flow regime in which the speed of a fluid is greater than the speed of sound in the medium it passes through.

$$Mach\ Number\ (Ma) > 1$$

In supersonic flows, sound waves do not propagate back against the flow, but are only carried forward. The physical effects of this condition are quite different.

Shock waves are frequently seen in supersonic flows. The flow slows down suddenly, for example at the nose or wingtip of an aircraft, and passes into a high-pressure, high-temperature region. After the shock wave, the flow temperature increases, the velocity decreases, and the entropy increases.

As the flow passes over an expanding surface, it accelerates and reaches supercritical velocities. In these regions, temperature and pressure drop and the velocity increases.

Because pressure waves do not propagate backwards, the flow cannot simply respond to obstacles in front of it, so aerodynamic design is very sensitive.

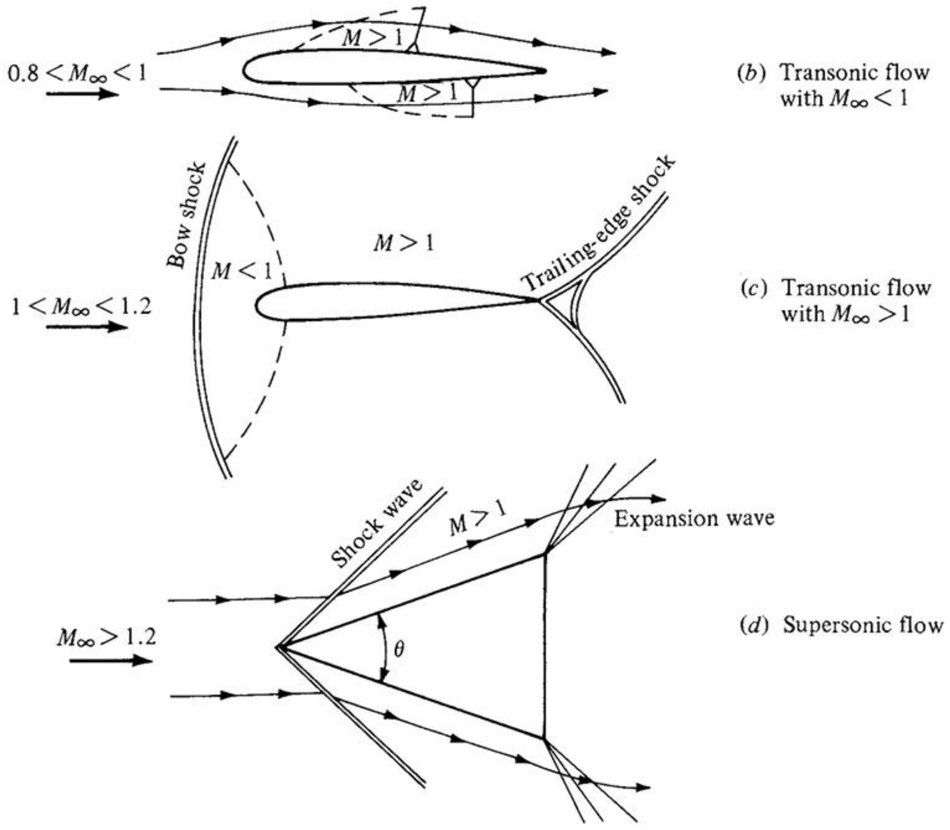


Figure 3: Transonic and Supersonic Flow (Anderson, 2017).

2.2. Wing Theory

2.2.1. Wing Shape

A delta wing is a wing which has a triangular shape when viewed from above (hence the name delta (Δ), often with its tip cut off. It sweeps sharply back from the front with the angle between the leading edge (the front) of the wing often as high as 60 degrees and the angle between the back of the plane and the trailing edge of the wing at around 90 degrees. Often delta-wing airplanes lack horizontal stabilizers. Even though paper airplanes have delta wings and appear to fly quite well when launched from a height, delta wings perform poorly at low speeds and often are unstable (i.e., they do not stay in level flight on their own). Their primary advantage is efficiency in super or hypersonic flight. The types of different widely used delta wings are represented in Figure 4.

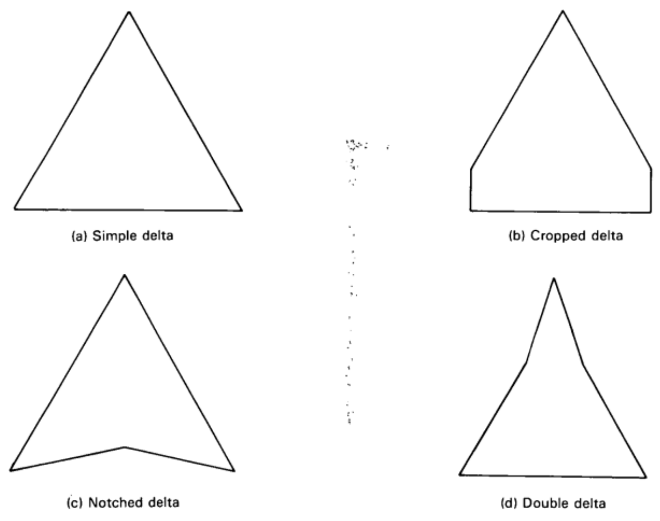


Figure 4: Types of delta wings (Loftin, 1985).

2.2.2. Aerofoil Shape

The shape of the aerofoil is what determines the pressure distribution around the aerofoil thereby determining the lift and drag forces generated.

A supersonic aerofoil is a cross-section geometry designed to generate lift efficiently at supersonic speeds. The need for such a design arises when an aircraft is required to operate consistently in the supersonic flight regime.

A lifting force that operates at a right angle to the airstream and a dragging force that acts in the same direction as the airstream are produced by an aerofoil, sometimes referred to as an aerofoil, which is the shape of an aircraft wing. The two types of aerofoils that are most frequently utilized in supersonic travel are double-wedge and bi-convex. The most basic and significant source of drag in areas of supersonic flow is wave drag. When an aerofoil's Mach number exceeds 1, it is referred to as supersonic. A supersonic aerofoil is made to efficiently provide lift at supersonic speeds. The leading and trailing edges of supersonic aerofoils are extremely sharp, and the thin section is made of either angled planes or opposing arcs. Usually described as double wedge aerofoils & a biconvex aerofoils.

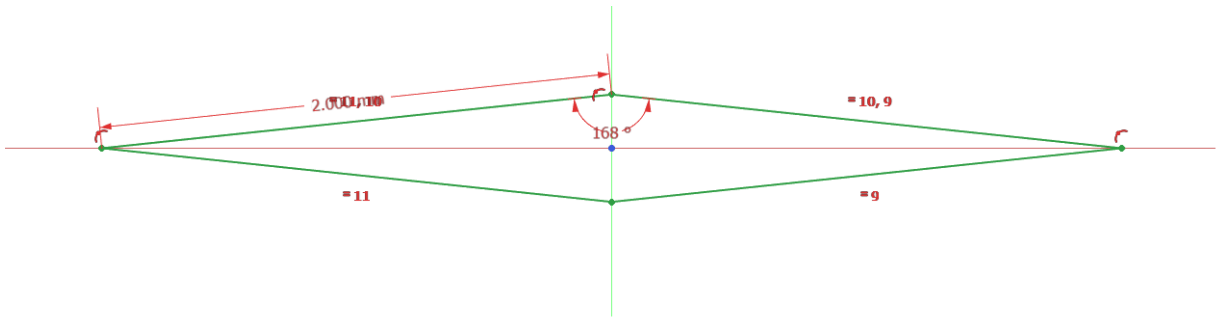


Figure 5: double wedge aerofoil.

2.2.3. Parametrization of wing

Parametrization is the process of expressing a system, function, or dataset in terms of a set of parameters. It gives a flexible control to the system and makes it easier to further analyse the system for different parameters. By the parametrization of the delta wing, we could generate the geometry of the system for different inputs. These inputs are the parameters of the system which will affect the geometry of the wing and are important for the analysis of these wings. The wing types, single delta and cropped delta are simple to parametrize. The parameters needed are:

- Sweep angle, θ_{sweep} [°]
- Wingspan, b [m]
- Cord length, c [m]

When these are obtained as an input, geometries for both types of wings could be generated. For the notched delta type, the trailing angle ($\theta_{trailing}$ [°]) is also needed as an input. For the double delta type, second sweep angle ($\theta_{sweep,2}$ [°]) and angle the percentage of the wingspan which has the first sweep angle should also be submitted. These parameters are visualized at Figure 6These are needed to generate the 2-D shape of the wing. To plot the 3-D shape, the max thickness (t [m]) is needed. If the aerofoil type is diamond shaped no other input is needed however, if the aerofoil type is biconvex, the radius of the upper and lower edges are also needed. Aerofoil parameters are visualized at Figure 7.

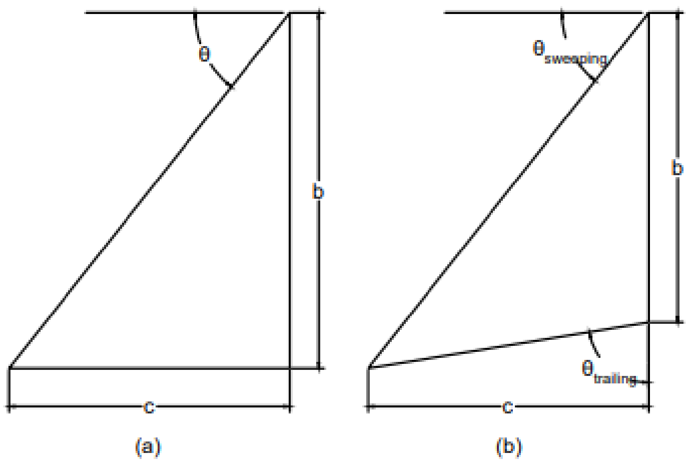


Figure 6: Parametrization diagrams for each wing type. (a) single and cropped (b) notched (c) double.

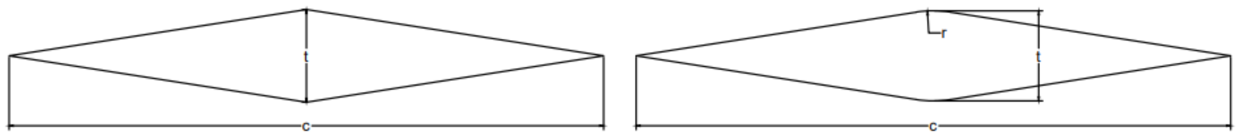


Figure 7: Parametrization of aerofoils. (a) diamond shaped (b) biconvex.

2.3. Sampling

2.3.1. What is Design of Experiment?

This work includes designing and optimization of delta wing profiles which are widely used in aircrafts. These processes (designing and optimization) are performed by obtaining theorical data based on books, articles, journals and results of specific experiments. During this process, specific methods are used to avoid noisy data. It is called “Design of Experiment”. The design of experiment (DOE) as a statistical method has been widely applied in different fields of science and industry, especially to support the design, development, and optimization of products and processes (B.Durakovic, 2017).

The design of experiments includes a series of applied statistics tools used to systematically classify and quantify the causes and effect relation between variables and outputs in the studied process or phenomenon. In this project, there are many geometric parameters related to the designing of a delta wing (wingspan, aerofoil thickness, swept angle, trailing angle, chord length etc.) including many parameters. Aim is the designing delta wing which gives optimum performance in supersonic flow and design off performance. Applying the design of experiment on a study or phenomenon may result in finding the settings and conditions under which the process optimized.

As mentioned above, many parameter values are generated to design delta wing profile. To give a sample case, there are four different design parameters, and each parameter has 10 valued range in specific intervals. Simply permutation calculation, it gives 10000 different geometries for designing a delta wing. To save time and avoid design which caused low performance, a sampling method must be used to decide the best combination to get optimum performance and support the optimization process.

Unfortunately, there is very little information in the literature that investigates and explains what types of procedures and steps need to be taken to find the optimal DOE among all the possible alternative options which have been developed and proposed (Aleksandar Jankovic, 2021). In this project, Latin hypercube sampling method is aimed at obtaining the best results in design process.

2.3.2. Latin Hypercube Sampling

Latin hypercube sampling (LHS) is a sampling method which generates random sample of parameter values from a wide range set. The sampling method is often used to construct the useful subset from obtained all data.

In statistical sampling, a square grid containing sample positions is a Latin Square. For instance, there is only one combination (or sample) in each row and each column. A Latin hypercube sampling is generated that each simple is one each axis aligned. Figure 8 shows what hyperplane is.

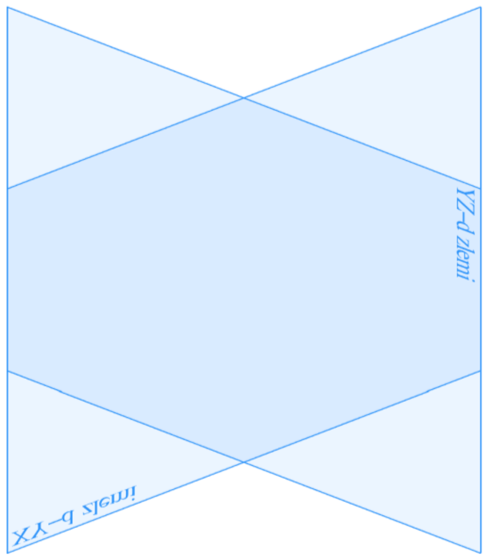


Figure 8: Figure of Hyperplanes.

The range of each variable is split up into M equally likely intervals when sampling a function of N variables. The number of divisions, M, must then be equal for each variable to meet the Latin hypercube requirements, which are satisfied by placing M sample points. One of the primary benefits of this sampling strategy is that it does not necessitate more samples for additional dimensions (variables). One other benefit is that random samples can be collected one at a time, making it easy to recall which samples were collected. The following figure explains the sampling method graphically.

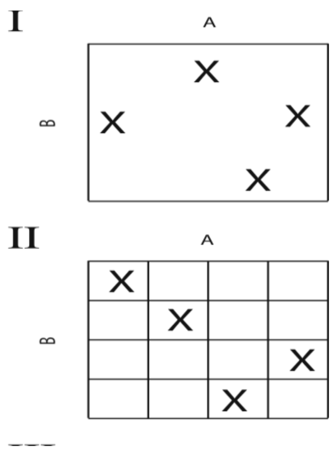


Figure 9: (a) random sampling, (b) LHS sampling.

Figure 9 (a) defines random sampling. All points are generated without considering any intersection in each row and column.

Figure 9(b) defines Latin hypercube sampling, each sample point must be unique and only one in each row and column. Such configuration is like Chessboard order. having N rooks on a chess board without threatening each other.

2.4. FreeCAD

CAD software such as SOLIDWORKS, ANSYS, and AutoCAD Inventor are widely used in the professional industrial world. However, in this project, since the developed application is based on a versatile and open-source software infrastructure, CAD programs that do not require a license were preferred.

Open-source CAD applications such as OpenSCAD, SolveSpace, LibreCAD, and FreeCAD were evaluated, and FreeCAD was preferred. FreeCAD is open-source parametric designer programme to design objects. The most important reason for FreeCAD to stand out is that it allows the user to make directly integrated designs with the Python language thanks to its "Macro" module. FreeCAD also includes "Record Macro" function built-in which records every action done by the user. It converts these actions into python script.

In this way, Python-based tools and automation continued to be used not only in the design phase but also in all processes after modelling.

2.5. GMSH

GMSH is an open-source program which has 4 main modules: geometry generation, mesh generation, solver and post-processing. The program has a built in GUI however it can also be used with its own scripting language (.geo) or other languages: C++, C, Python, Julia and Fortran.

For its geometry generation, it starts by adding points on the coordinate system from there; by connecting these points, one can make a line. By connecting closed loop of lines one can make a surface and by connecting a set of surfaces one can make a volume. Also, there are options to make simple 3D objects such as sphere, cylinder, box etc. To create a more complex 3D geometry, it is also possible to do extruding by translating, rotating and piping. GMSH also has built-in Boolean options where one can take the intersection, union, difference or fragments of multiple objects and tools. It is also possible to import other CAD outputs in other formats such as ".STEP". By this method one can still utilize the mesh generation of GMSH while having a more complex geometry which may not be drawable inside GMSH.

After the geometry generation and before mesh generation it is needed to create physical groups. These physical groups give definition to the elements inside the mesh. These definitions include mathematical definitions, functional definitions and material definitions. These definitions are very important for the CFD analysis for that some settings of the analysis demand to know them. And not having them may result in wrong analysis results.

After creating physical groups, it possible to mesh the geometry. However, the mesh generated would have evenly distributed mesh sizes along the geometry. This is not usually desirable while having a CFD analysis so to generate finer meshes at some locations inside a geometry there are multiple ways: to tweak with the setting of that location or create a field which is effective in that location. The first one is simplest way to do it in the case of using GMSH's own scripting language (.geo). It is done by editing the already created "geo" file and adding the desired mesh size right after the coordinated of the geometry element. After this tweaking the program will read the mesh sizes on each element and generate meshes with smooth transitions in between each element. Second way to do it would be to creating a field and picking the mesh size, the distance to original element where that mesh size would be effective. The second option is more complex but it gives the user more advanced control over the mesh generation especially at the transition of the mesh sizes. After the desired mesh is generated, it is possible to export in almost every format for a CFD analysis.

The advantages of GMSH are that it is a free and open-source mesh generation tool that runs on Windows, macOS, and Linux. It has a simple graphical user interface, making it easy to learn and use. One of its major strengths is the ability to create geometries and meshes using script files, allowing for full automation. GMSH also supports 1D, 2D, and 3D meshing with high flexibility and integrates well with many simulation tools. Additionally, it includes a Python API, which allows users to control and generate meshes directly from Python scripts, making it a powerful choice for advanced workflows and automation. Mesh refinement and local adaptation are also supported, which is essential for accurate numerical simulations.

2.5.1. Mesh independence

The CFD analysis should be mesh independent in order to have a closer to reality results. To reach mesh independence we generated a single sample wing, and we generated separate meshes which have different smallest mesh sizes. After mesh generation, we ran a CFD analysis on each of these meshes at both 0.5 and 1.5 Ma speeds and took the C_L/C_D ratios of each run. The analysis has resulted as:

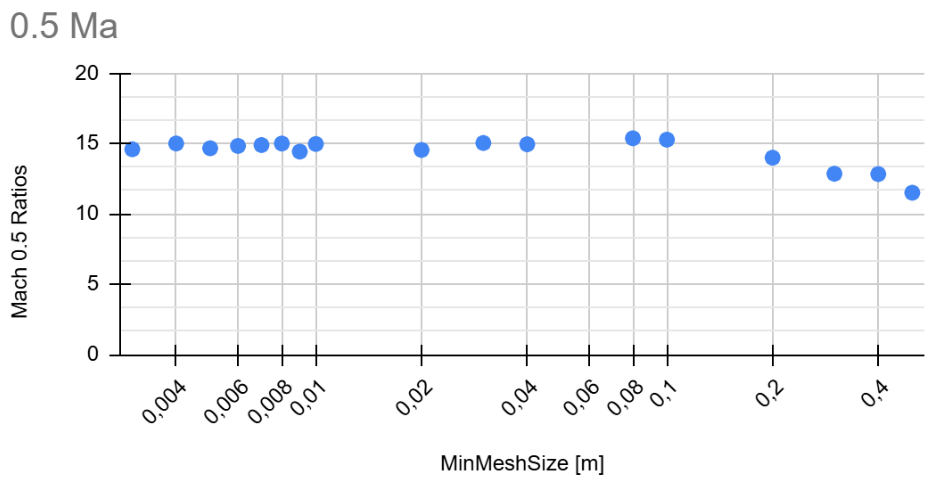


Figure 10: Graph of C_L/C_D ratios for different minimum mesh sizes at 0.5 Ma.

From Figure 10 at 0.1 m the analysis has an acceptable mesh independence. Since the ratios do not fluctuate after that point.

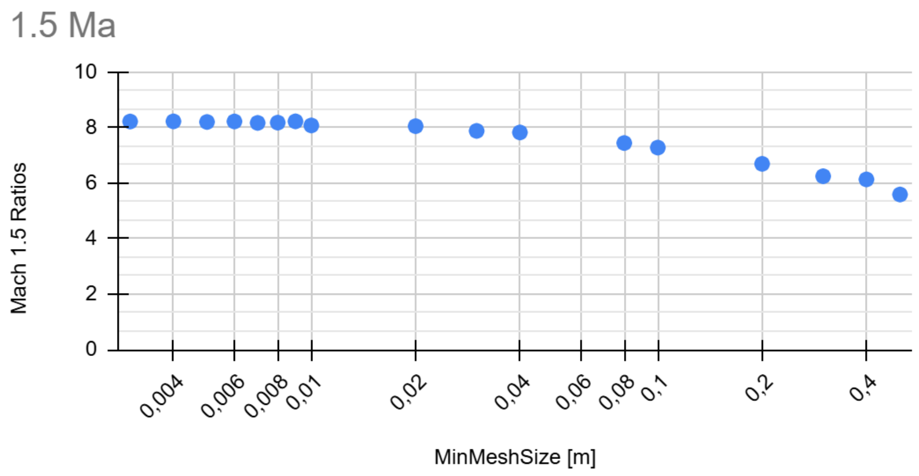


Figure 11: Graph of C_L/C_D ratios for different minimum mesh sizes at 1.5 Ma.

From Figure 11 at 0.01 m the analysis has an acceptable mesh independence. Since the ratios do not fluctuate after that point.

After these analyses we have selected to set the minimum mesh sizes to be 0.01 m since it is the minimum mesh size at both Mach numbers which our computing could undertake such analyses and still have mesh independence.

2.6. SU2

Computational analysis tools have revolutionized the way we design engineering systems, but most established codes are proprietary, unavailable, or prohibitively expensive for many users. SU2 makes Multiphysics analysis and design optimization software freely available.

This Computational Fluid Dynamics software, developed for academic studies by Stanford University's Aerospace Design Laboratory (ADL) and named SU2 from "Stanford University Unstructured Code", joined the Open-Source community within the framework of the decision taken and the first accessible version was published on January 19, 2012. The SU2 system, the development of which started in 2003, consists of open-source C++ based software components that have been shaped and developed to solve partial differential equations within the framework of aerodynamic engineering needs and on irregular meshes. The basic structure of the SU2 code can be summarized as follows:

- SU2_CFD- Main CFD solver
- SU2_PRT- Mesh partitioning
- SU2_SOL- Solution export
- SU2_MSH- Mesh adaptation
- SU2_DEF- Mesh shape modification
- SU2_GEO- Geometry definition

The main important capabilities of the software are:

- Ability to use in a very wide Mach range
- Plasma modeling
- Flow-Structure interaction

- Ability to work on optimization problems with the ability to change internal geometry/mesh
- Flow-compatible automatic mesh refinement
- Turbulence transition model

2.7. Python

Python is an object-oriented, interpreted, modular and interactive high-level programming language.

The simple syntax based on indentations makes it easy to learn and remember the language. This makes it a language that allows programming to begin without wasting time on the details of the syntax.

Its modular structure supports the class system (system) and all types of data field input. It can run on almost any platform (Unix, Linux, Mac, Windows, Amiga, Symbian). With Python, software can be developed in many areas such as system programming, user interface programming, network programming, web programming, application and database software programming.

The part of the Python program related to this study can be stated as follows:

There are many parameters that need to be evaluated during the study, and therefore, the adaptation of this study to automation makes an important contribution. The Python-based design program "FreeCAD" can be used while providing this adaptation. In addition, using Python will also make it easier during script writing. In the analysis part, SU2 will be preferred again in terms of automation and convenience.

3. PROCESS

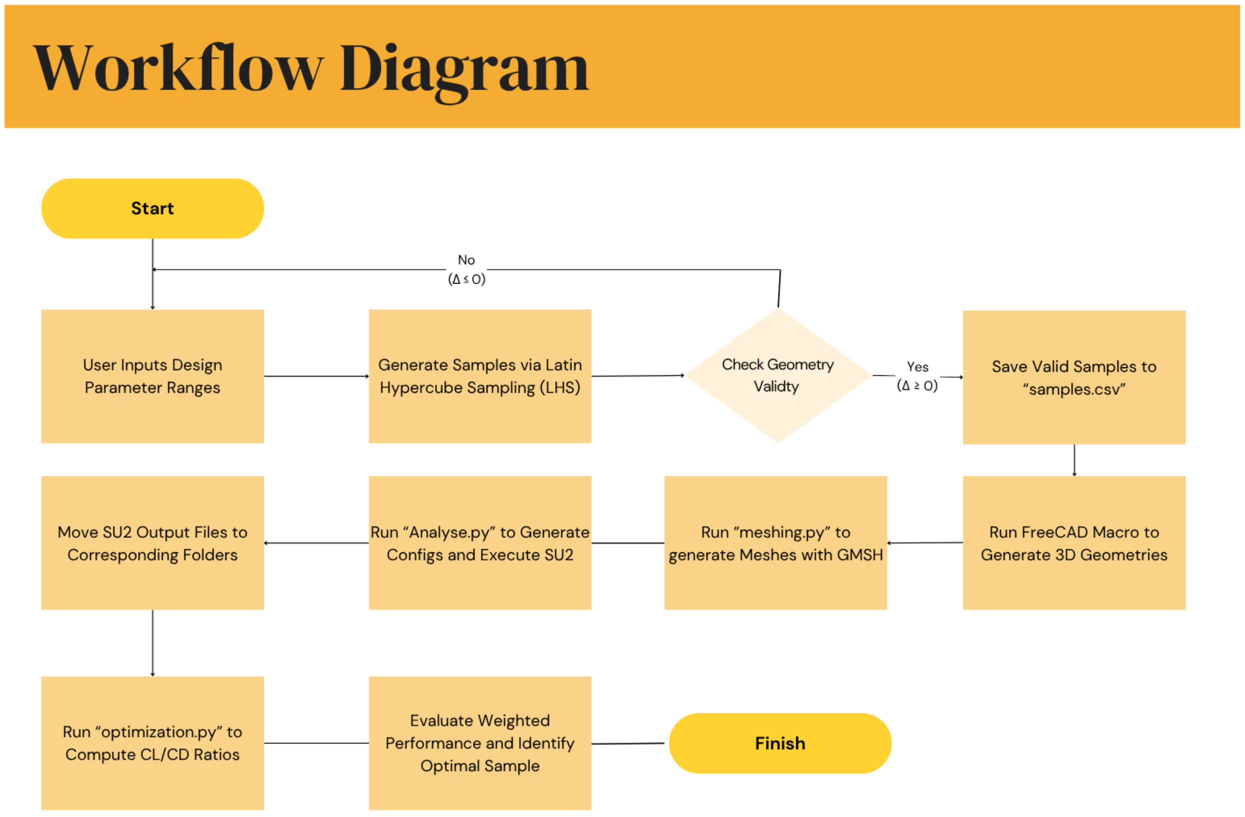

Workflow Diagram
Start
No (Δ ≤ 0)
User Inputs Design Parameter Ranges
Generate Samples via Latin Hypercube Sampling (LHS)
Check Geometry Validty
Yes (Δ ≥ 0)
Save Valid Samples to "samples.csv"
Move SU2 Output Files to Corresponding Folders
Run "Analyse.py" to Generate Configs and Execute SU2
Run "meshing.py" to generate Meshes with GMSH
Run FreeCAD Macro to Generate 3D Geometries
Run "optimization.py" to Compute CL/CD Ratios
Evaluate Weighted Performance and Identify Optimal Sample
Finish

Figure 12: Workflow diagram of the program.

3.1. GUI

This platform is based on Python and the working process starts when you give the "Run" command to the gui code for the prepared gui via a console such as PyCharm Community Edition.

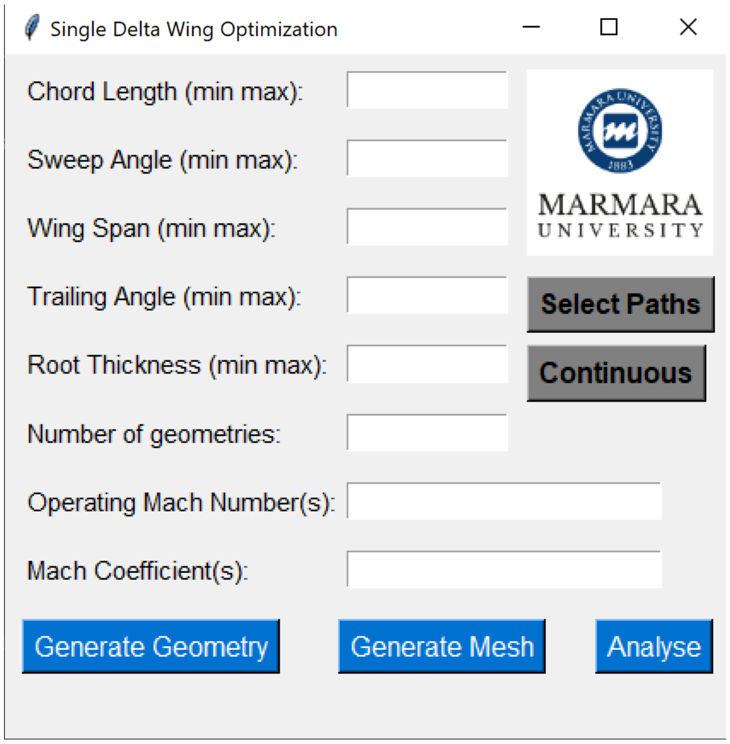
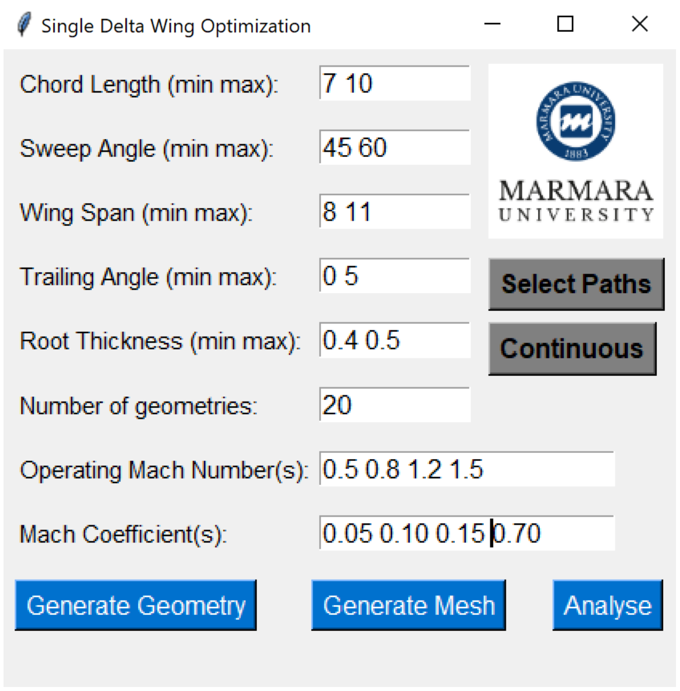


Figure 13: Gui Interface of the Platform

The GUI code starts running and the new interface seen in the figure above opens in front of the user. This interface contains input sections for parameters that are important for wing design. In order for the analysis process to take place, the desired ranges in the parameters must first be specified. Accordingly, the platform will automatically proceed with the process.



Chord Length (min max):	7 10
Sweep Angle (min max):	45 60
Wing Span (min max):	8 11
Trailing Angle (min max):	0 5
Root Thickness (min max):	0.4 0.5
Number of geometries:	20
Operating Mach Number(s):	0.5 0.8 1.2 1.5
Mach Coefficient(s):	0.05 0.10 0.15 0.70

Figure 14: The GUI interface with Input

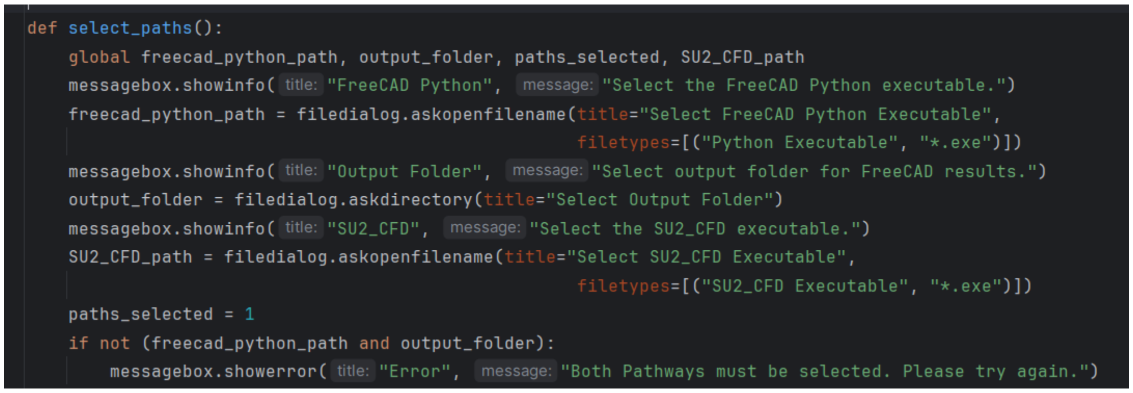
The input format in the GUI interface is very important. Because if the format seen in Figure 14 is not followed, the platform cannot detect this input and perform the necessary operations.

When processing inputs to the interface, the inputs must be entered in the "Chord Length, Sweep Angle, Wingspan, Trailing Angle and Root Thickness" input sections containing the phrase "(min max)" in the form of "x y". If more than 2 inputs are entered, the platform will give a warning. For decimal numbers, "." should be used, not ",".

The desired amount of geometry should be entered into the section specified as "Number of Geometries".

In the "Operating Mach Number(s)" section, the values of the Mach numbers you want to work with should be entered in the "x y z" format with a 1-character space in between.

In the "Mach Coefficient(s)" section, the same number of coefficients should be entered in the same format as the number of Mach numbers entered in the "Operating Mach Number(s)" section.

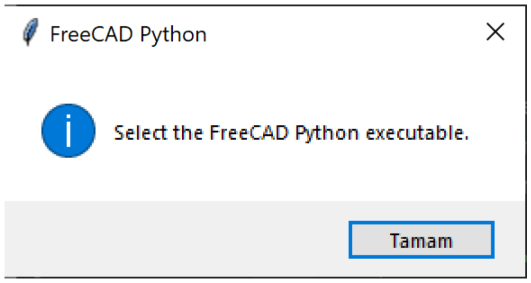


```
def select_paths():
    global freecad_python_path, output_folder, paths_selected, SU2_CFD_path
    messagebox.showinfo( title: "FreeCAD Python",  message: "Select the FreeCAD Python executable.")
    freecad_python_path = filedialog.askopenfilename(title="Select FreeCAD Python Executable",
                                                     filetypes=[("Python Executable", "*.exe")])
    messagebox.showinfo( title: "Output Folder",  message: "Select output folder for FreeCAD results.")
    output_folder = filedialog.askdirectory(title="Select Output Folder")
    messagebox.showinfo( title: "SU2_CFD",  message: "Select the SU2_CFD executable.")
    SU2_CFD_path = filedialog.askopenfilename(title="Select SU2_CFD Executable",
                                                     filetypes=[("SU2_CFD Executable", "*.exe")])
    paths_selected = 1
    if not (freecad_python_path and output_folder):
        messagebox.showerror( title: "Error",  message: "Both Pathways must be selected. Please try again.")
```

Figure 15: Python Code of "Select Paths" Button in Gui

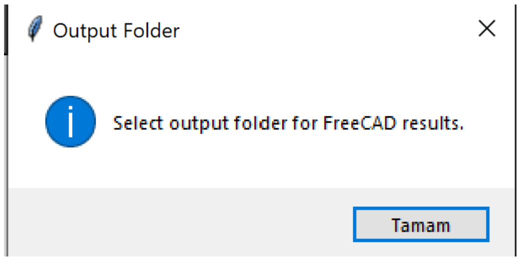
After all parameters are entered, first the "Select Paths" button must be pressed. In this button, the appropriate process files must be selected so that the necessary applications can perform their functions correctly. In order for FreeCAD macros to work, "python.exe" must be selected from the "bin" folder of the FreeCAD installation file. Then the appropriate folder for the samples is determined and finally, in order to perform CFD analysis, the "SU2_CFD" file must be selected from the "bin" folder among the program files of the SU2 application. The code in Figure 15 shows the operation of the "Select Paths" button.

The pop-ups appearing in Figure 16 and Figure 17 and Figure 18 respectively guide you through the process of selecting the necessary files.


FreeCAD Python

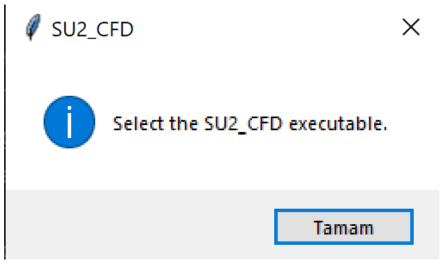
Select the FreeCAD Python executable.

Figure 16: Pop-up of Selecting FreeCAD Python Executable


Output Folder

Select output folder for FreeCAD results.

Figure 17: Pop-up of Selecting Folder for FreeCAD Results

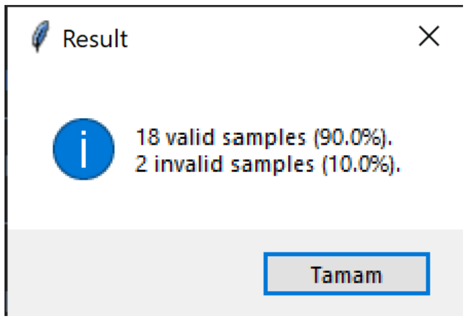

SU2_CFD

Select the SU2_CFD executable.

Figure 18: Pop-up of Selecting SU2_CFD Executable

After selecting the appropriate folder and file in the "Select Paths" section, click the "Generate Geometry" button. If the files were selected correctly in the previous step, the platform will start creating geometry samples according to the Latin Hypercube Sampling method.

The platform creates geometries and examines these geometries for theoretical correctness in its own control mechanism. Then, it removes the theoretically invalid ones and sends them to the trash. In this way, unnecessary analysis is avoided. An example is seen in the pop-up in "Figure 7".


Result

18 valid samples (90.0%).
2 invalid samples (10.0%).

Figure 19: Pop-up of Consistency of Sample Geometries

After the geometries are created, the "Generate Mesh" button is pressed in the interface in Figure 13.Mesh structures for the geometries created in this section are created by the GMSH program and saved in the folder in ".su2" format.

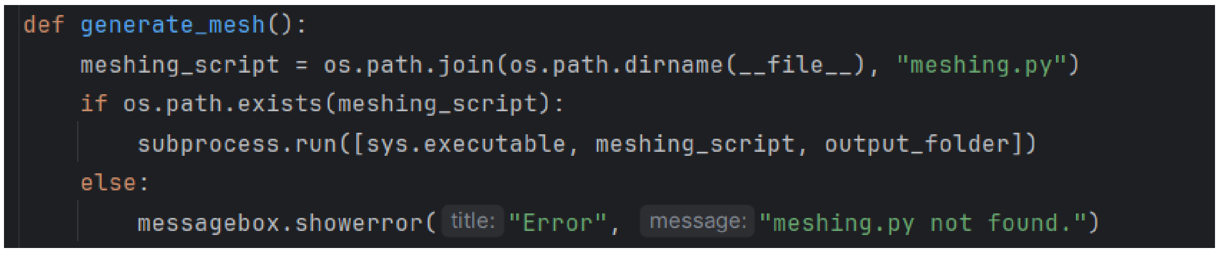
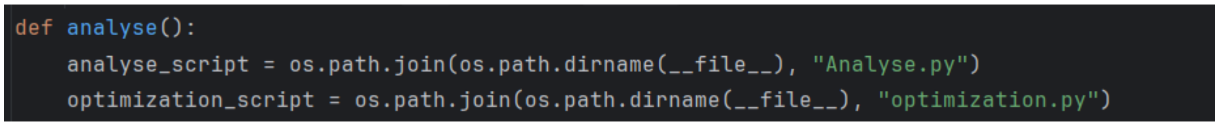

```
def generate_mesh():
    meshing_script = os.path.join(os.path.dirname(__file__), "meshing.py")
    if os.path.exists(meshing_script):
        subprocess.run([sys.executable, meshing_script, output_folder])
    else:
        messagebox.showerror( title: "Error",  message: "meshing.py not found.")
```

Figure 20: Code of Generate Mesh Button

The code in Figure 20 is the code for the generate mesh button. When the button is activated, it runs another code file of the code, the "meshing.py" file. The GUI code and the meshing.py code file can be examined in the "Appendix" section.

After the mesh structures are created, if the CFD analysis of the samples is to be examined, the "Analyse" button is pressed and the platform begins the analysis process. It is a somewhat long process. For this reason, it is preferred that the platform is optional during the process.

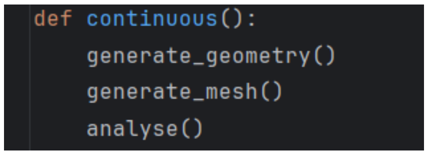


```
def analyse():
    analyse_script = os.path.join(os.path.dirname(__file__), "Analyse.py")
    optimization_script = os.path.join(os.path.dirname(__file__), "optimization.py")
```

Figure 21: Python Code of Analyse Button

As seen in Figure 21, when the "Analyse" button is activated, the Python files named "Analyse.py" and "optimization.py" are run and the CFD analysis and optimization of the samples are performed at this stage. The "Analyse.py" and "optimization.py" code files can be examined in the "Appendix" section.

After entering all the parameters in the interface in Figure 13,if the user does not want to bother pressing the buttons in order and wants to perform all the steps at once and obtain the CFD outputs, they can press the "Continuous" button and this button will perform all the operations at once, starting from the sample creation.

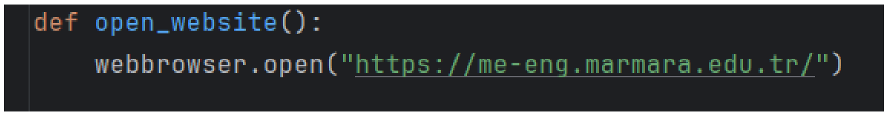


```
def continuous():
    generate_geometry()
    generate_mesh()
    analyse()
```

Figure 22: Python Code of Continuous Button

As seen in Figure 22, if the "Continuous" button is activated after the sample parameters are specified, geometry creation, mesh structure creation and analysis operations are performed with a single button.

Finally, if the user clicks on the Marmara University logo on the interface, the platform directs the user to the website of the Mechanical Engineering Department of Marmara University Faculty of Engineering, where we are students. The Python code required for this redirection is shown in Figure 23.

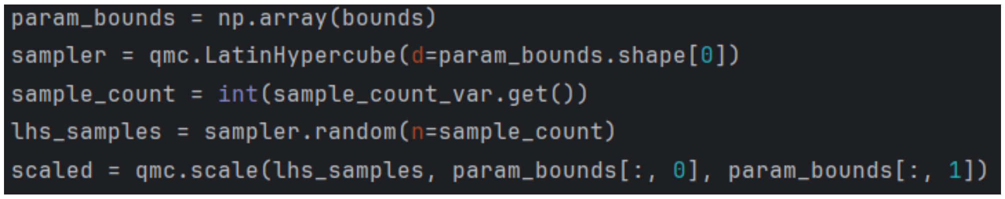


```python
def open_website():
    webbrowser.open("https://me-eng.marmara.edu.tr/")
```

Figure 23: Python Code of Marmara University Logo Button

3.2. Sampling

As explained before we used LHS method. This method was well interpreted into python via scipy.stats library qmc module. We select the dimensions as 5. First the code does sampling between 0 and 1. Then scales these numbers by multiplying them with parameter boundary which taken from the user as inputs. The code is as follows in Figure 24.



```python
param_bounds = np.array(bounds)
sampler = qmc.LatinHypercube(d=param_bounds.shape[0])
sample_count = int(sample_count_var.get())
lhs_samples = sampler.random(n=sample_count)
scaled = qmc.scale(lhs_samples, param_bounds[:, 0], param_bounds[:, 1])
```

Figure 24: Python code for LHS method.

3.3. Checking Drawability

To ensure that only physically meaningful and manufacturable wing geometries are retained, a control mechanism was implemented based on a geometric consistency check. After generating sample design parameters via Latin Hypercube Sampling (LHS), each sample is evaluated using a custom-defined function called delta (Δ), which is given by:

$$\Delta = c - b * (\tan(\theta_{\text{sweep}}) + \tan(\theta_{trail}))$$

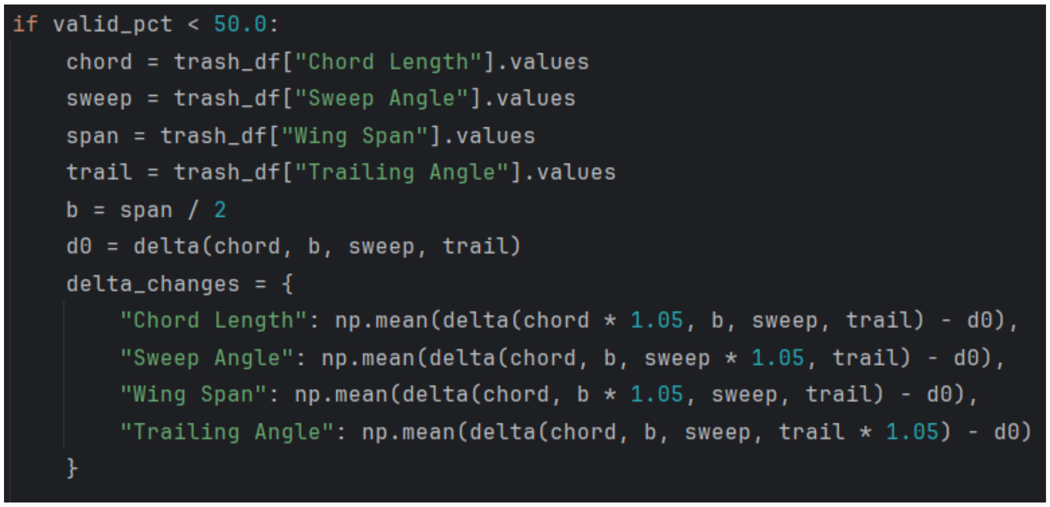
The sweeping and trailing angles are first converted from degrees to radians before evaluating the tangent function, as required by the numerical computation.

If the computed Δ value is negative, the corresponding geometry is considered invalid and excluded from the main dataset. Such geometries typically represent non-physical shapes where the leading edge may fall behind the trailing edge in planform projection.

All invalid samples are stored separately in a file named trash.csv, while the valid ones are saved in samples.csv.

The code monitors the proportion of valid samples. If less than 50% of the total generated geometries are valid, a sensitivity analysis is performed. Each of the primary input parameters

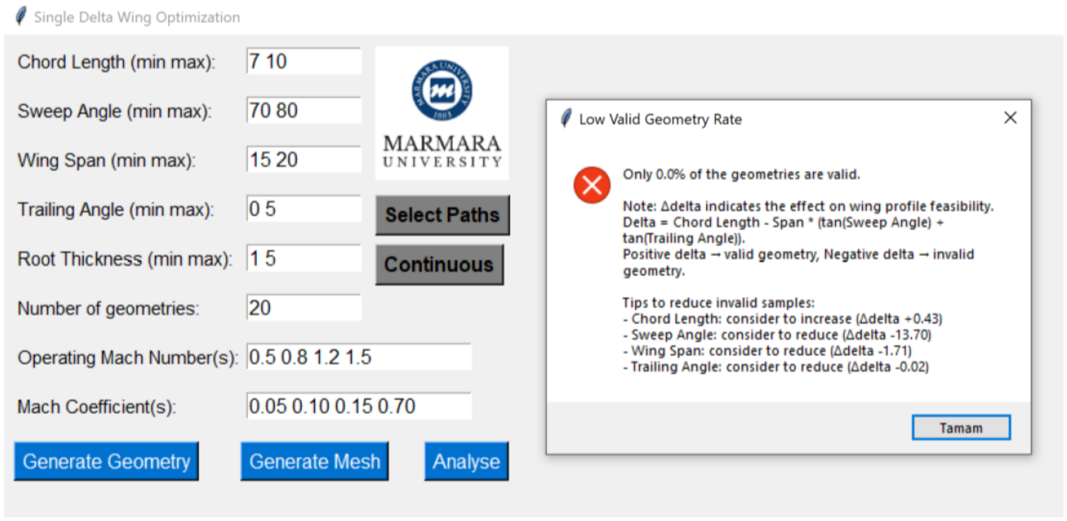
(Chord Length, Sweep Angle, Wingspan, Trailing Angle) is slightly increased (by 5%) to determine its effect on improving the Δ value. The mean change in Δ is computed for each case to guide future adjustments to input bounds. Calculation is done by the code Figure 25.



```
if valid_pct < 50.0:
    chord = trash_df["Chord Length"].values
    sweep = trash_df["Sweep Angle"].values
    span = trash_df["Wing Span"].values
    trail = trash_df["Trailing Angle"].values
    b = span / 2
    d0 = delta(chord, b, sweep, trail)
    delta_changes = {
        "Chord Length": np.mean(delta(chord * 1.05, b, sweep, trail) - d0),
        "Sweep Angle": np.mean(delta(chord, b, sweep * 1.05, trail) - d0),
        "Wing Span": np.mean(delta(chord, b * 1.05, sweep, trail) - d0),
        "Trailing Angle": np.mean(delta(chord, b, sweep, trail * 1.05) - d0)
    }
```

Figure 25: Code for calculating parameters' effect on delta.

After calculation program creates a error popup (Figure 26 on the right) and writes these found changes a suggestion.



Single Delta Wing Optimization

Chord Length (min max): 7 10
Sweep Angle (min max): 70 80
Wing Span (min max): 15 20
Trailing Angle (min max): 0 5
Root Thickness (min max): 1 5
Number of geometries: 20
Operating Mach Number(s): 0.5 0.8 1.2 1.5
Mach Coefficient(s): 0.05 0.10 0.15 0.70

Low Valid Geometry Rate

```
Only 0.0% of the geometries are valid.

Note: Δdelta indicates the effect on wing profile feasibility.
Delta = Chord Length - Span * (tan(Sweep Angle) +
tan(Trailing Angle)).
Positive delta → valid geometry, Negative delta → invalid
geometry.

Tips to reduce invalid samples:
- Chord Length: consider to increase (Δdelta +0.43)
- Sweep Angle: consider to reduce (Δdelta -13.70)
- Wing Span: consider to reduce (Δdelta -1.71)
- Trailing Angle: consider to reduce (Δdelta -0.02)
```

Figure 26: Error popup for small number of drawable wings.

These insights can be used to refine the sampling ranges and enhance the rate of valid geometry generation. Full code is in the section 7.1.

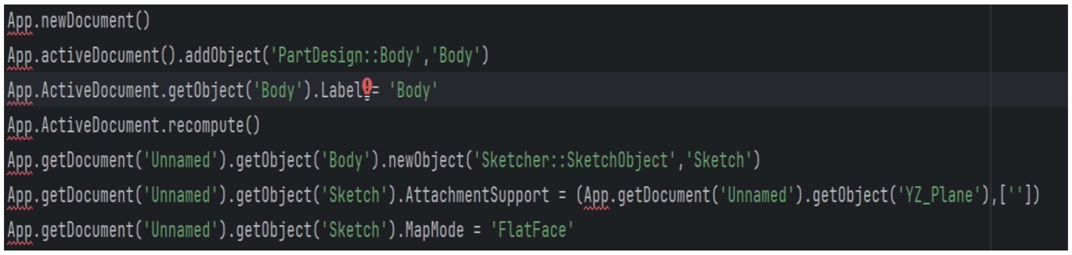
3.4. Wing Design and Modelling

While modelling the wing, the process started with 2D drawings in the FreeCAD environment. All drawing stages were recorded in a macro. The necessary adjustments were made in Python based on the resulting script and the process was completed.

Among the delta wing types mentioned in the theory section, "Cropped Delta" was preferred because it was relatively easy to draw. The most critical point in aerofoil design was the constraint assignments in the leading edge and trailing edge sections of the section. Since the drawing was started from the origin, the leading and trailing sections were prevented from interfering with each other in the inputs entered, or rather, they were limited and prevented from overlapping or shifting.

The macro code explained in the rest of the article is explained through a design draft that we tested during the project.

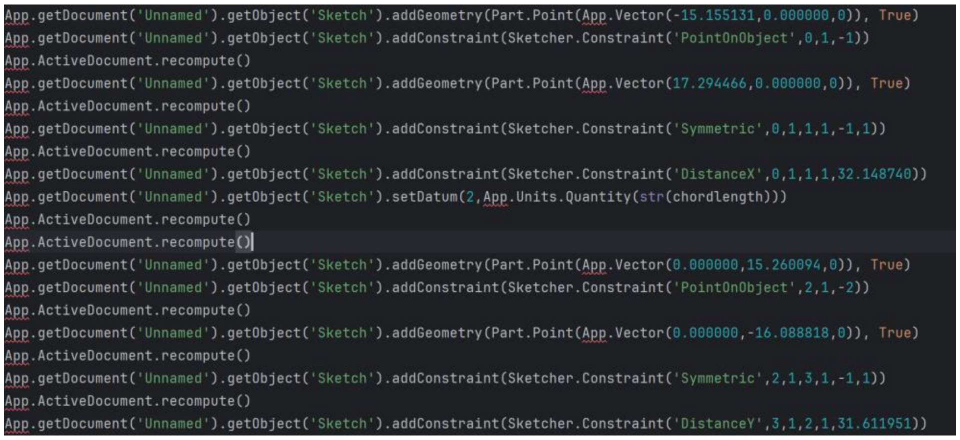
As seen in Figure 27, a body was first created and a sketch was created on the YZ axis.



```
App.newDocument()
App.activeDocument().addObject('PartDesign::Body','Body')
App.ActiveDocument.getObject('Body').Label = 'Body'
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Body').newObject('Sketcher::SketchObject','Sketch')
App.getDocument('Unnamed').getObject('Sketch').AttachmentSupport = (App.getDocument('Unnamed').getObject('YZ_Plane'),[''])
App.getDocument('Unnamed').getObject('Sketch').MapMode = 'FlatFace'
```

Figure 27: Creating sketch.

Later, four points were added—two on the Y axis and two near it. A symmetry constraint was applied between them. While drawing and recording the macro, the points were given random dimensions. These dimensions were later changed and adjusted to match the values needed by the application. Figure 28 shows these steps.

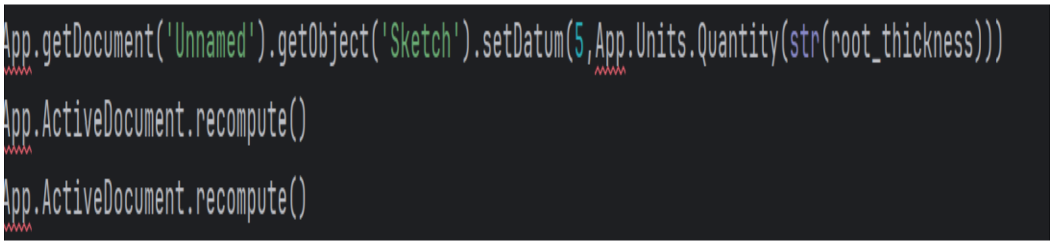


```
App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vector(-15.155131,0.000000,0)), True)
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('PointOnObject',0,1,-1))
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vector(17.294466,0.000000,0)), True)
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('Symmetric',0,1,1,1,-1,1))
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('DistanceX',0,1,1,1,32.148740))
App.getDocument('Unnamed').getObject('Sketch').setDatum(2,App.Units.Quantity(str(chordlength)))
App.ActiveDocument.recompute()
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vector(0.000000,15.260094,0)), True)
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('PointOnObject',2,1,-2))
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vector(0.000000,-16.088818,0)), True)
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('Symmetric',2,1,3,1,-1,1))
App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('DistanceY',3,1,2,1,31.611951))
```

Figure 28: Drawing the aerofoil.

The distance between points positioned along the Y axis is defined as chord length. A variable called "root_thickness" is assigned to this dimension. This variable contains the "Root Thickness" line in the csv file containing the input parameter values from the user.

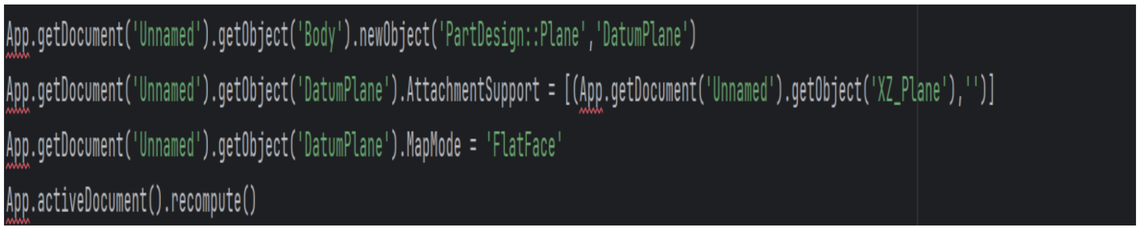
In the macro function, the "str" function is used because in the FreeCAD macro, it cannot be defined if it is not assigned as a string Figure 29.



```
App.getDocument('Unnamed').getObject('Sketch').setDatum(5,App.Units.Quantity(str(root_thickness)))
App.ActiveDocument.recompute()
App.ActiveDocument.recompute()
```

Figure 29: String coverting example.

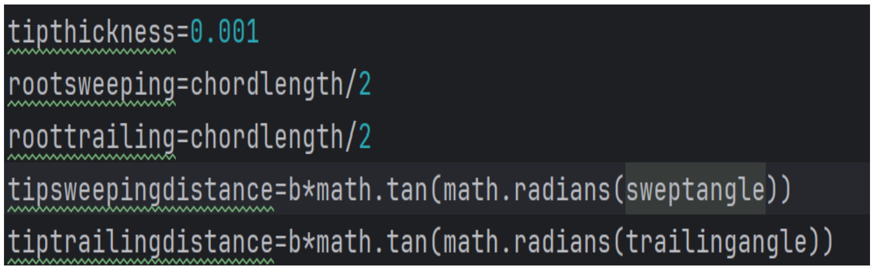
After the aerofoil section at the root was completed, the aerofoil section at the tip was drawn. A datum plane was created from the YZ plane where the root section was drawn.



```
App.getDocument('Unnamed').getObject('Body').newObject('PartDesign::Plane','DatumPlane')
App.getDocument('Unnamed').getObject('DatumPlane').AttachmentSupport = [(App.getDocument('Unnamed').getObject('XZ_Plane'),'')]
App.getDocument('Unnamed').getObject('DatumPlane').MapMode = 'FlatFace'
App.activeDocument().recompute()
```

Figure 30: Creating datum plane.

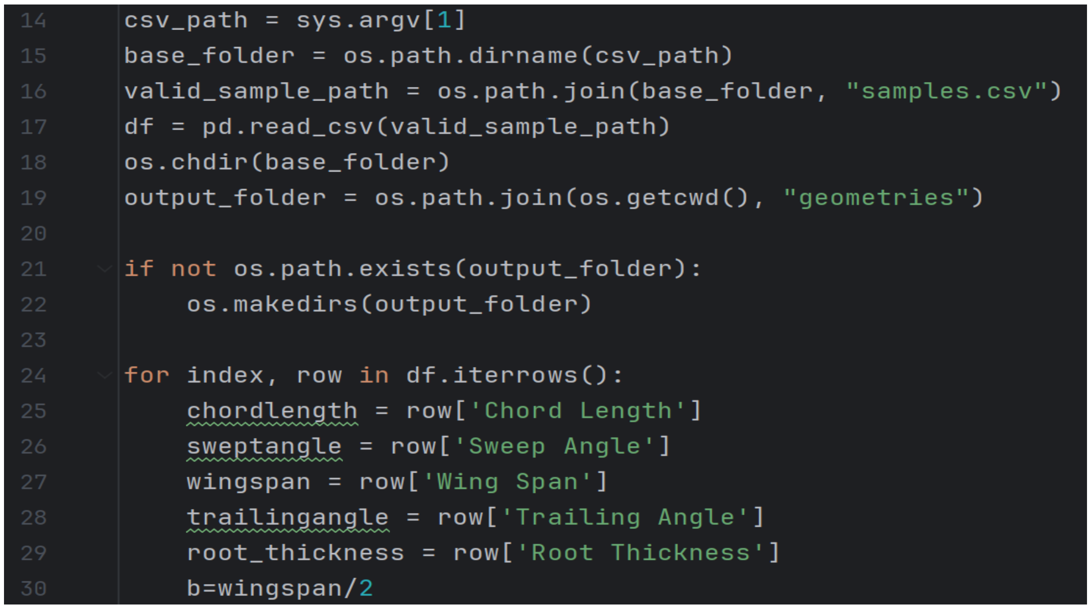
The steps followed while drawing the root section are repeated in this section. While creating a solid model from the root section to the tip section, constraints were added by considering the sweep angle and trailing angle. Points were assigned to the left and right of the leading and trailing edge, respectively. The lengths between these points and the edges are defined as “tipsweepdistance” and “tipstralingdistance.” In this way, a section image suitable for the wing geometry was obtained. Lines for calculations are in Figure 31.



```
tipthickness=0.001
rootsweeping=chordlength/2
roottrailing=chordlength/2
tipsweepingdistance=b*math.tan(math.radians(sweptangle))
tiptrailingdistance=b*math.tan(math.radians(trailingangle))
```

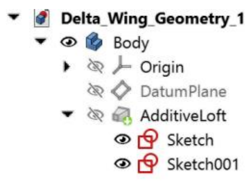
Figure 31: Calculations for tip drawing.

The variable assignments made in the macro throughout the drawing were taken from a CSV file. An array was created and written to the CSV file as a result of the parameters entered in the application screen, which will be discussed in detail in the rest of the project. The values there were transferred row by row in the macro in the lines Figure 32.



```
csv_path = sys.argv[1]
base_folder = os.path.dirname(csv_path)
valid_sample_path = os.path.join(base_folder, "samples.csv")
df = pd.read_csv(valid_sample_path)
os.chdir(base_folder)
output_folder = os.path.join(os.getcwd(), "geometries")

if not os.path.exists(output_folder):
    os.makedirs(output_folder)

for index, row in df.iterrows():
    chordlength = row['Chord Length']
    sweptangle = row['Sweep Angle']
    wingspan = row['Wing Span']
    trailingangle = row['Trailing Angle']
    root_thickness = row['Root Thickness']
    b=wingspan/2
```

Figure 32: Getting values from samples.csv and assigning needed parameters.


Delta_Wing_Geometry_1
Body
Origin
DatumPlane
AdditiveLoft
Sketch
Sketch001

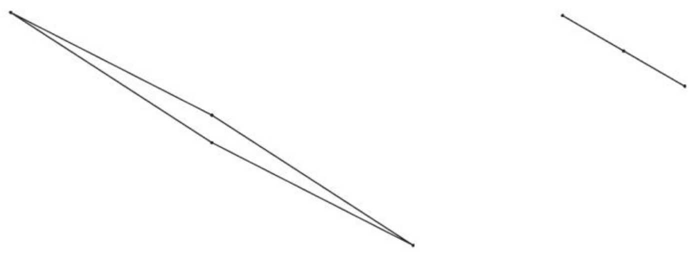


Figure 33: Isometric view of aerofoil sketches of an example wing.

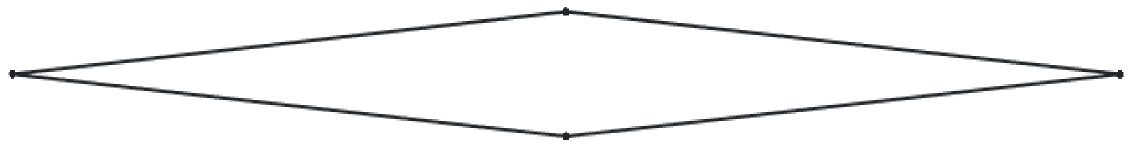
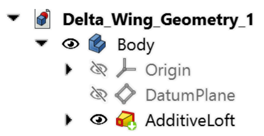


Figure 34: Root Aerofoil sketch of an example wing.

After the 2D sketches were finished, both sections were joined using the "Additive Loft" feature to form a 3D solid model. Figure 35 and Figure 36 represent a wing design created from random input parameter values.



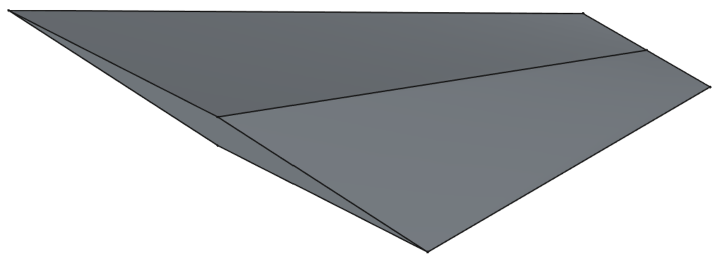


Figure 35: Isometric view of an example wing.

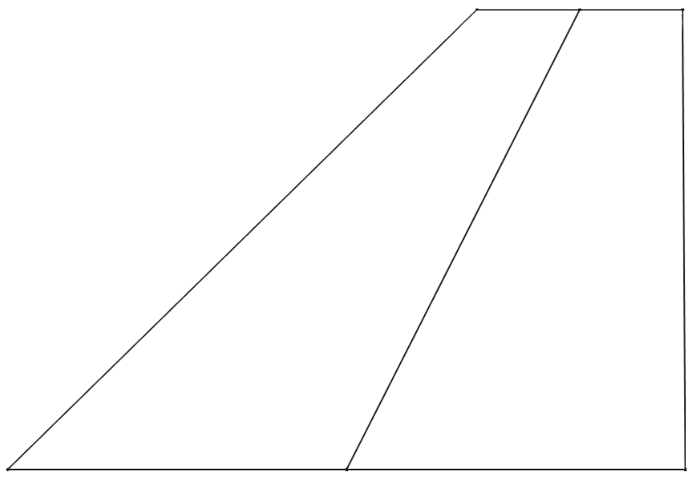
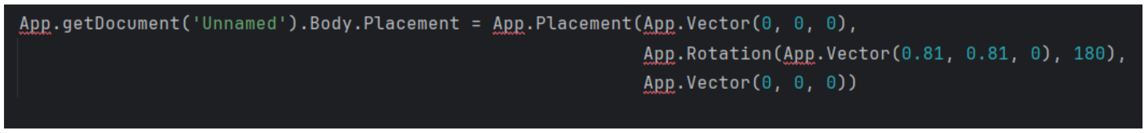


Figure 36: Top view of an example wing.

When selecting the plane to be drawn, since the SU2 analysis setup accepts the X axis as the flow direction, the drawing was performed on the XZ axis as spanwise Y for convenience. To change the direction of the wing we used the line in Figure 37.

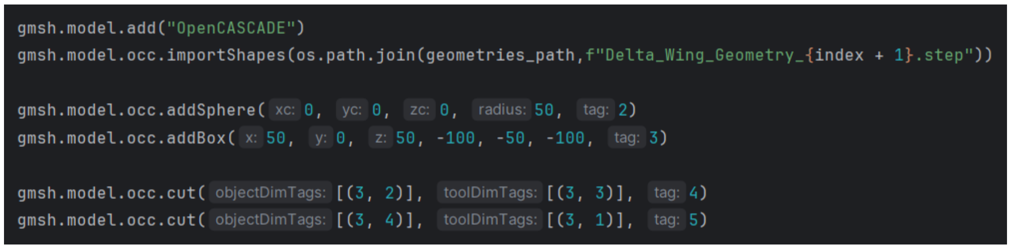


```
App.getDocument('Unnamed').Body.Placement = App.Placement(App.Vector(0, 0, 0),
                                                          App.Rotation(App.Vector(0.81, 0.81, 0), 180),
                                                          App.Vector(0, 0, 0))
```

Figure 37: Direction changing codes.

3.5. Generating mesh

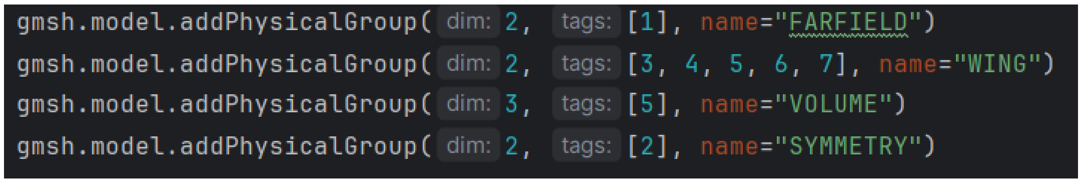
In order to automate the meshing process, we used the Python API of GMSH. Our code first imports the geometry generated in the previous step as a ".STEP" file. Then we add a sphere which has a diameter of approximately 12 times the cord length. Since we will be only analysing a single wing, we need a hemisphere. To get a sphere, we add a box which covers the non-needed part of the sphere and then do a 3D Boolean difference operation where object is the sphere and tool is the box. This gives us a hemisphere. But we also need to subtract the wing from the hemisphere, so we again do a 3D Boolean difference operation where object is the hemisphere and the tool is the wing. After this operation the geometry is ready. The code responsible for these actions are in Figure 38.



```
gmsh.model.add("OpenCASCADE")
gmsh.model.occ.importShapes(os.path.join(geometries_path,f"Delta_Wing_Geometry_{index + 1}.step"))

gmsh.model.occ.addSphere( xc: 0,  yc: 0,  zc: 0,  radius: 50,  tag: 2)
gmsh.model.occ.addBox( x: 50,  y: 0,  z: 50, -100, -50, -100,  tag: 3)

gmsh.model.occ.cut( objectDimTags: [(3, 2)],  toolDimTags: [(3, 3)],  tag: 4)
gmsh.model.occ.cut( objectDimTags: [(3, 4)],  toolDimTags: [(3, 1)],  tag: 5)
```

Figure 38: GMSH geometry.

After geometry is ready in GMSH we need to define physical groups. In this project we have 4 different physical groups: FARFIELD, WING, SYMMETRY, VOLUME. First 3 of these physical groups are 2D and last one is 3D (hence the name). The group FARFIELD represents the edges of the domain, WING represents the surfaces of the wing, SYMMETRY represents the symmetry plane of the system (in our case it the surface at which the sphere is cut), VOLUME represent the domain at which the air will flow. To find which tag is responsible for which element, we hand examined a test case by generating the same geometry by hand. In our test case we saw that surface 1 is the "FARFIELD", surfaces 3,4,5,6,7 is the "WING", surface 2 is the "SYMMETRY", and volume 5 is the "VOLUME". The code responsible for defining physical groups are Figure 39.



```
gmsh.model.addPhysicalGroup( dim: 2,  tags: [1], name="FARFIELD")
gmsh.model.addPhysicalGroup( dim: 2,  tags: [3, 4, 5, 6, 7], name="WING")
gmsh.model.addPhysicalGroup( dim: 3,  tags: [5], name="VOLUME")
gmsh.model.addPhysicalGroup( dim: 2,  tags: [2], name="SYMMETRY")
```

Figure 39: Physical groups.

At this stage the mesh is ready to generate however the mesh generated would be very coarse and unusable. To get a finer mesh we decreased the mesh sizes along some parts (some of these parts are shown in Figure 40) of the wing. These parts include: each corner of the wing and each edge of the wing along parallel to the wingspan axis (y axis). These corner points are: 5, 6, 7, 8, 9, 10, 11, 12. These edges are 11, 13, 15 ,17. This method would be enough for most of the edges but in some cases where the edges are longer, GMSH clusters fine sized meshes around equally spaced points on these edges. As a workaround, we assigned the possible long edges as transfinite curves and told the program to split these curves equally along the edge. The number of mesh amount is calculated by dividing the length of the curve to the mesh size. After doing this we see the desired uniform mesh distribution along the edges.

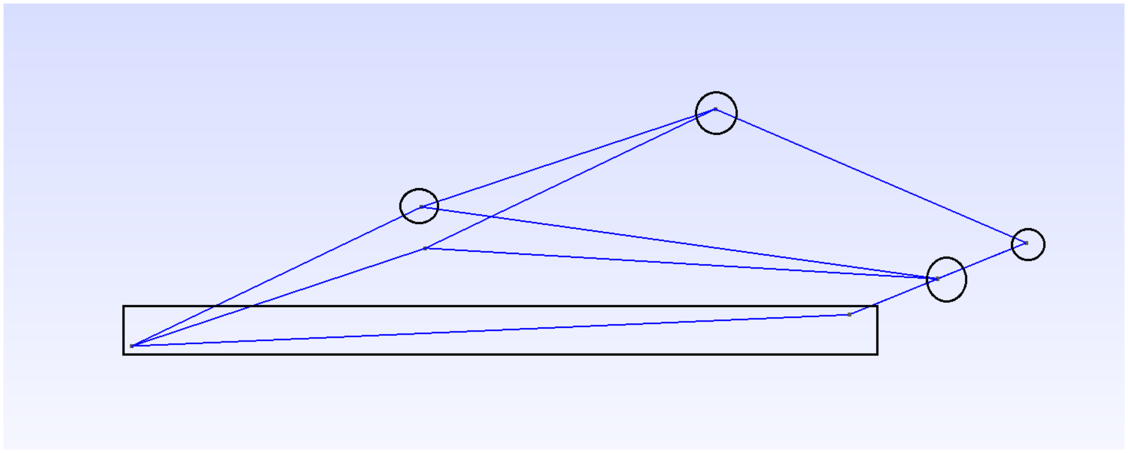
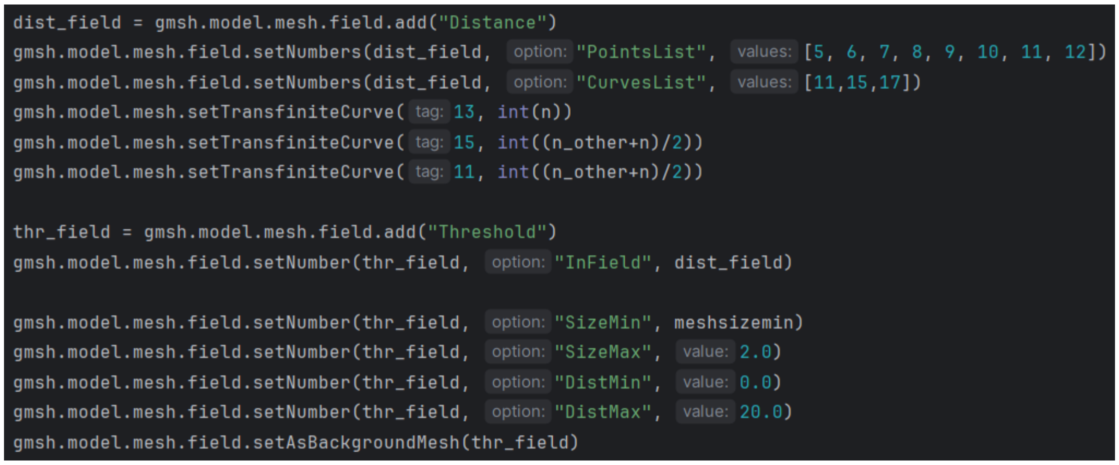


Figure 40: Some of the locations at which finer mesh sizes needed.

Then we assign a threshold field to select the distance at which we want finer mesh and to select the distance where a coarse mesh would suffice. We selected this distance to be 20m. So, our code starts to mesh on the wing with the mesh size of 0.01m and this mesh size increases to 2m at the distance of 20m from the wing. This process finalizes our efforts in making a finer mesh. This process is done by the code in Figure 41.

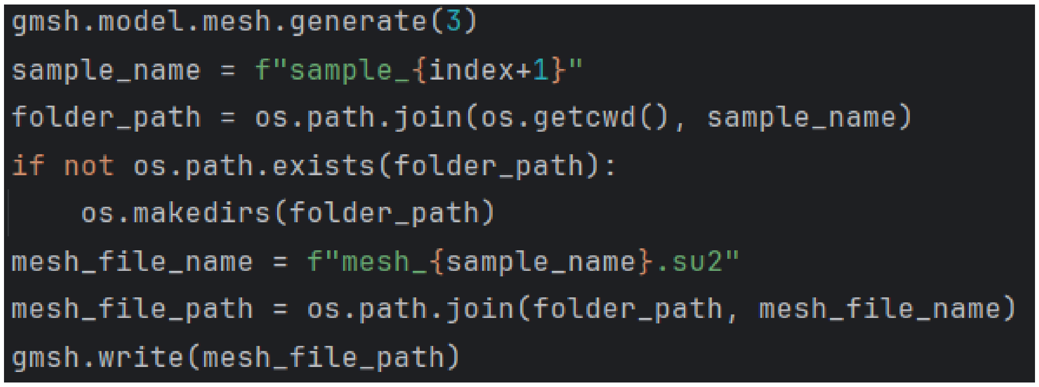


```
dist_field = gmsh.model.mesh.field.add("Distance")
gmsh.model.mesh.field.setNumbers(dist_field,  option: "PointsList",  values: [5, 6, 7, 8, 9, 10, 11, 12])
gmsh.model.mesh.field.setNumbers(dist_field,  option: "CurvesList",  values: [11,15,17])
gmsh.model.mesh.setTransfiniteCurve( tag: 13, int(n))
gmsh.model.mesh.setTransfiniteCurve( tag: 15, int((n_other+n)/2))
gmsh.model.mesh.setTransfiniteCurve( tag: 11, int((n_other+n)/2))

thr_field = gmsh.model.mesh.field.add("Threshold")
gmsh.model.mesh.field.setNumber(thr_field,  option: "InField", dist_field)

gmsh.model.mesh.field.setNumber(thr_field,  option: "SizeMin", meshsizemin)
gmsh.model.mesh.field.setNumber(thr_field,  option: "SizeMax",  value: 2.0)
gmsh.model.mesh.field.setNumber(thr_field,  option: "DistMin",  value: 0.0)
gmsh.model.mesh.field.setNumber(thr_field,  option: "DistMax",  value: 20.0)
gmsh.model.mesh.field.setAsBackgroundMesh(thr_field)
```

Figure 41: Field creation.

Before meshing, we had to change "angle tolerance facet overlap" setting because we had errors of facet overlap.

After this we mesh the geometry, then assign a name to the sample, create a folder in the output folder path in the name of the sample and save the mesh in the .su2 format inside this folder. It is done by the code in Figure 42.



```
gmsh.model.mesh.generate(3)
sample_name = f"sample_{index+1}"
folder_path = os.path.join(os.getcwd(), sample_name)
if not os.path.exists(folder_path):
    os.makedirs(folder_path)
mesh_file_name = f"mesh_{sample_name}.su2"
mesh_file_path = os.path.join(folder_path, mesh_file_name)
gmsh.write(mesh_file_path)
```

Figure 42: Saving the mesh.

For full code of this chapter see section 7.3.

3.6. SU2 Analysis

For the analysis part of delta wings, SU2 is used. To do an analysis in SU2, there should be 2 files. (configuration file, mesh files in the .su2 format).

The config file that is used in this project is ONERAM6. This config file can be found on the SU2 tutorials website. The reason for selecting ONERAM6 is, it is a wing that flies at transition

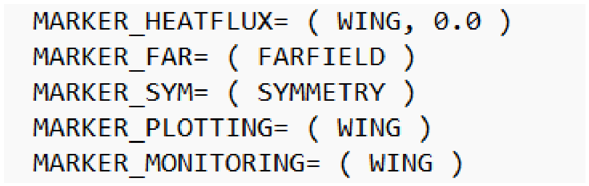
Mach numbers, and it fits the wings that are generated during this project. Also, after making the analysis the results were reasonable.

In the config file, the Mach number, CFL number, markers and the mesh input file name is changed for this project. Markers are determined as ‘WING, FARFIELD, SYMMETRY’.

WING: Wing marker is applied to the surface of the wing.

FARFIELD: Farfield marker is applied for the free-stream conditions.

SYMMETRY: Symmetry marker is applied to the hemisphere’s outer surface where the wing is placed into. (normal to the -y direction)



```
MARKER_HEATFLUX= ( WING, 0.0 )
MARKER_FAR= ( FARFIELD )
MARKER_SYM= ( SYMMETRY )
MARKER_PLOTTING= ( WING )
MARKER_MONITORING= ( WING )
```

Figure 43: Markers.

Here (in Figure 43) the MARKER_ HEATFLUX is applied such that the heat flux of wing is 0 and constant. The project does not deal with heat transfer effects.

MARKER_FAR and MARKER_SYM are explained above.

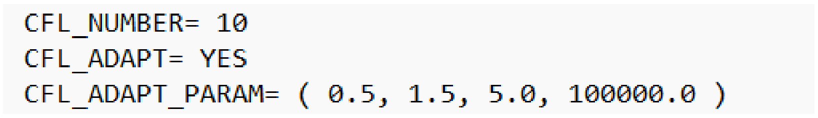
MARKER_PLOTTING is used for writing the flow data to the output files. Since the scope of this project is aerodynamic optimization, it is applied to the wing. This marker is for the visualization of the analysis.

MARKER_MONITORING is used to write the parameters (e.g Cl and Cd) to the history.csv file.

Determination of the Mach numbers are left to the user, but to see the on and off design behavior of the wings it is suggested to the user that enter both supersonic and subsonic flow conditions.

$$CFL = \frac{U * \Delta t}{\Delta h}$$

According to (Courant, Friedrichs, & Lewy, 1967) , the CFL number quantifies how much information travels (U) across a computational grid cell (Δh) in a unit of time (Δt).



```
CFL_NUMBER= 10
CFL_ADAPT= YES
CFL_ADAPT_PARAM= ( 0.5, 1.5, 5.0, 100000.0 )
```

Figure 44: CFL numbers.

CFL number was determined by trial and error.

Here (in Figure 44), CFL adapt option is used. First term is factor down; it decreases the CFL number if the residual increases. Second term is factor up it increases the CFL number if the residual decrease to certain value. Third term is the minimum value that CFL can take. Fourth term is the maximum value that CFL can take.

The other file needed is .SU2. This is the mesh file that is obtained from the gmsh program. It needs to be as in .SU2 extension to be work in SU2. For full code see section 7.4.

3.7. Optimization

The optimization is done by taking the weighted average of different Mach number cl/cd ratios. These weights are expected from the user. After the analyses is completed, program multiplies the cl/cd ratios of each Mach number with its corresponding coefficient and substitutes the ratios for a sample. Program records each substitute and decides which one is better. Code for this part is in the section 7.5.

4. RESULTS & DISCUSSION

We tested our program in a test case with a sample size of 20. Our parameters were as in Figure 14.

One of the geometries is in Figure 45 and Figure 46.

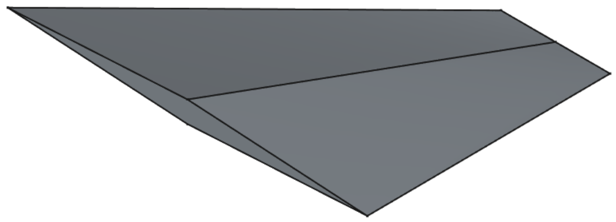


Figure 45: Isometric view a wing.

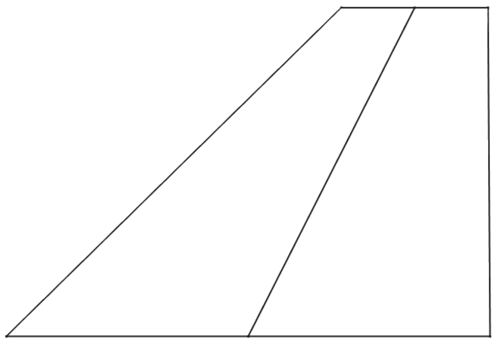


Figure 46: Top view of a wing.

After meshing we got Figure 47 and Figure 48

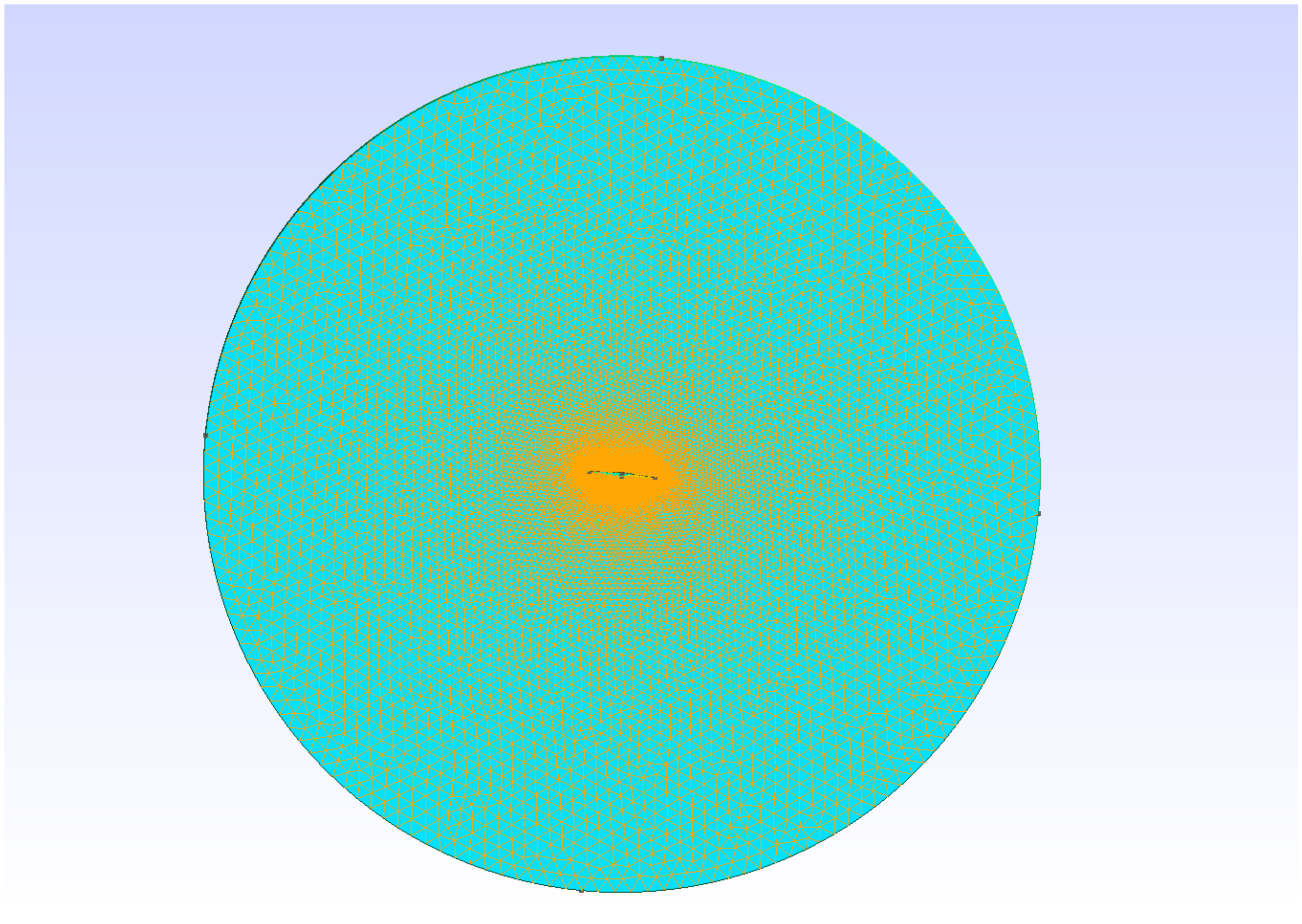


Figure 47: Domain mesh at the symmetry plane.

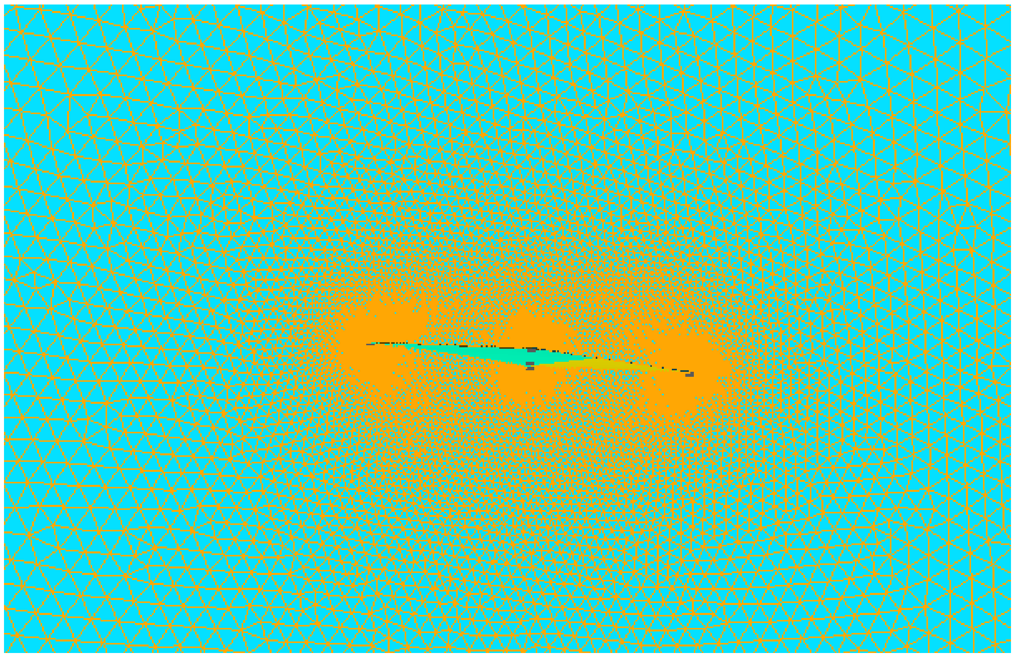


Figure 48: Mesh around the wing.

The finest meshing could be seen generated along the previously chosen elements (in Figure 49 it is the leading edge of the wing).

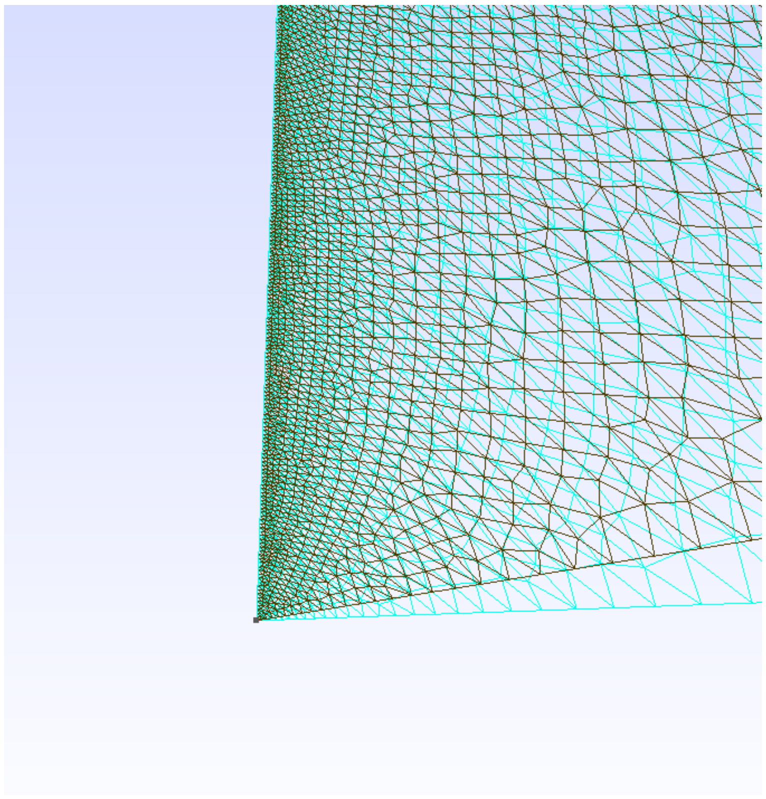
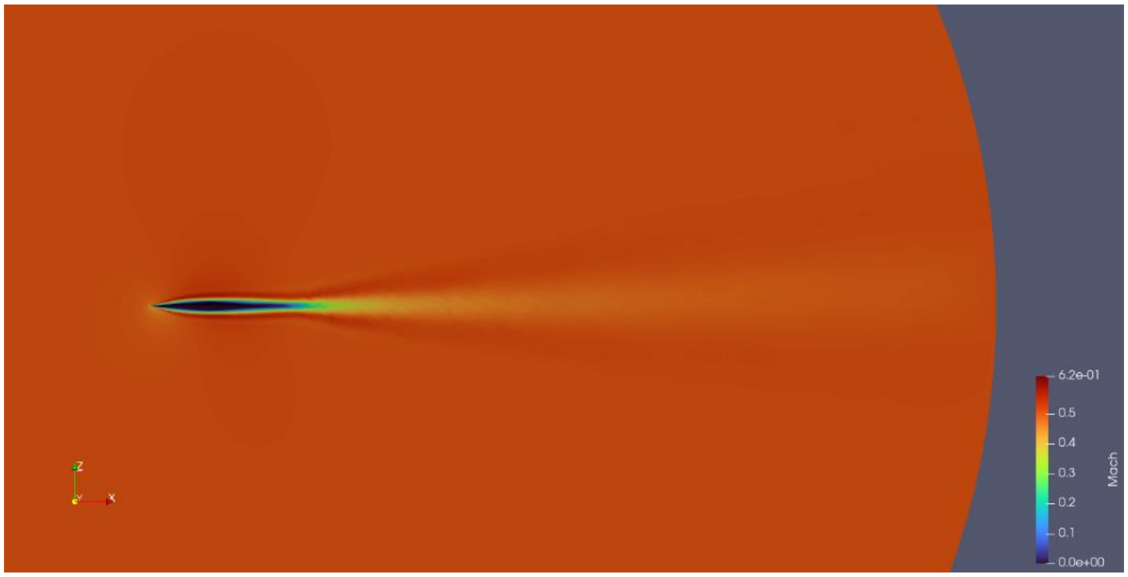
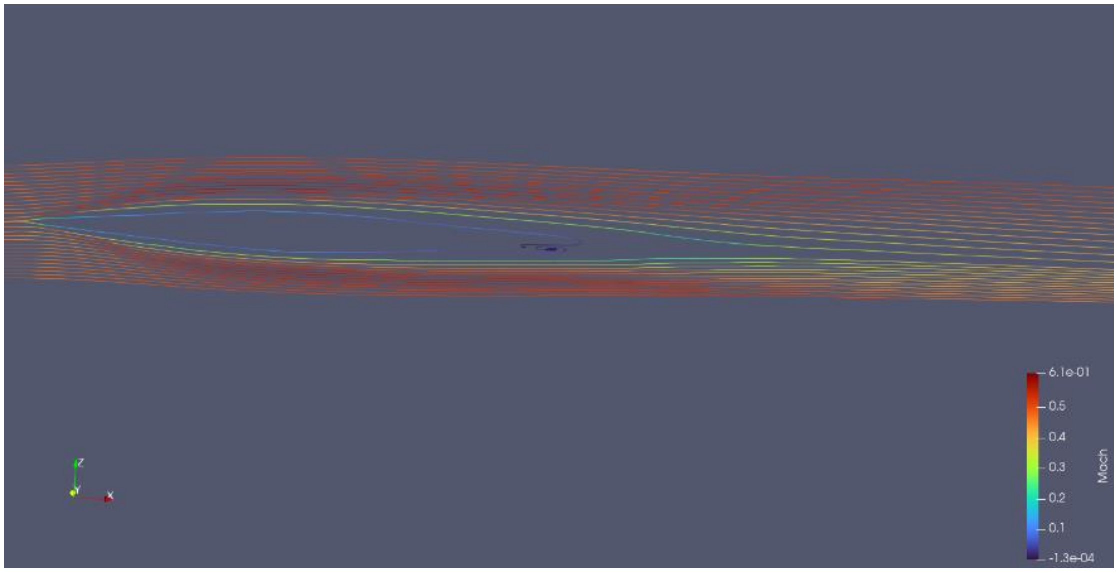


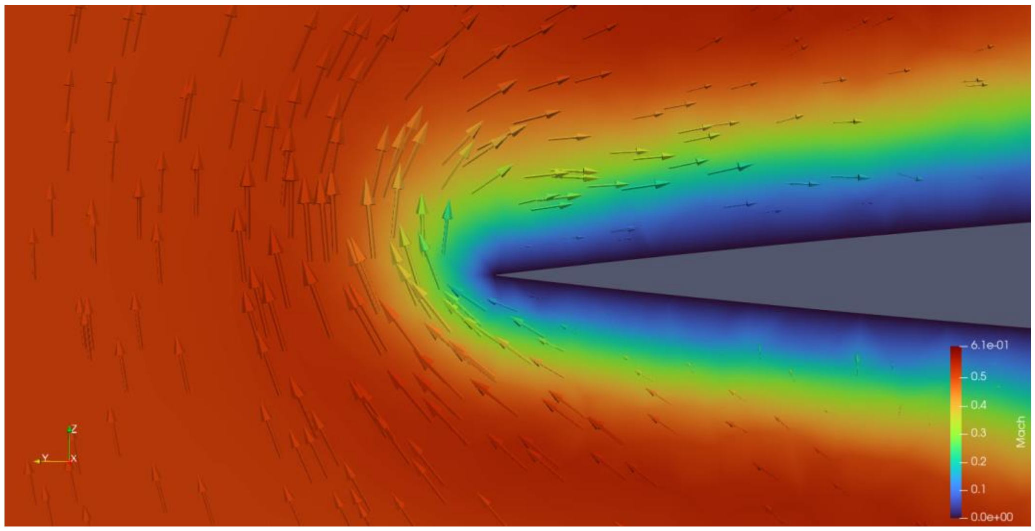
Figure 49: Meshing at tip of the upper wing surface.

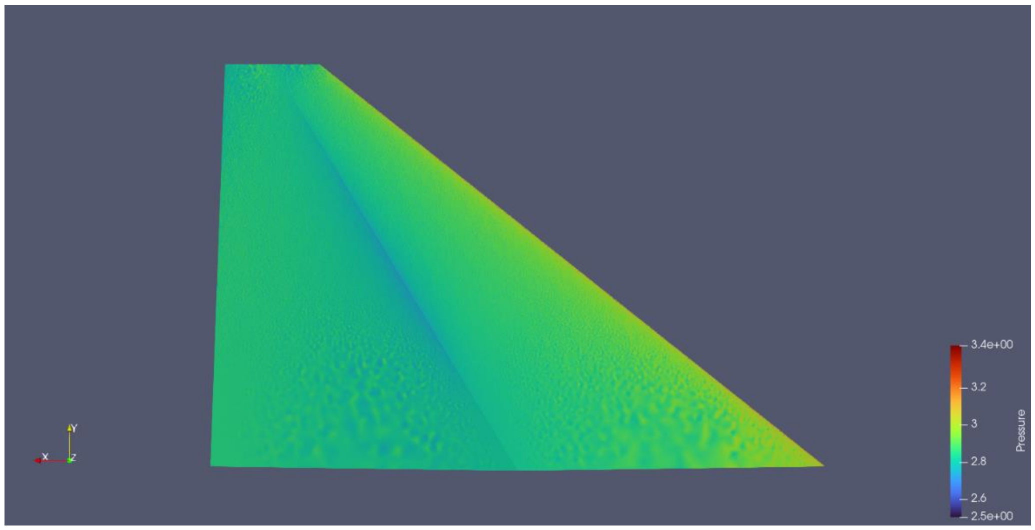
The mesh results were in line with our goals. It generated the desired fine meshes near the desired elements and became coarser towards the domain edges.

Analyses results for 0.5 Ma (Figure 50). The C_L/C_D ratio is found to be 16,97967662.









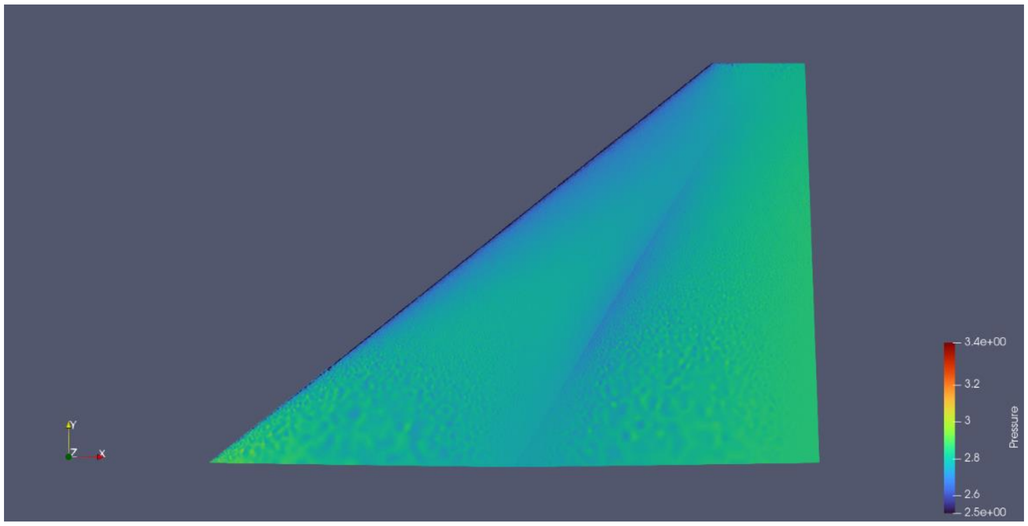
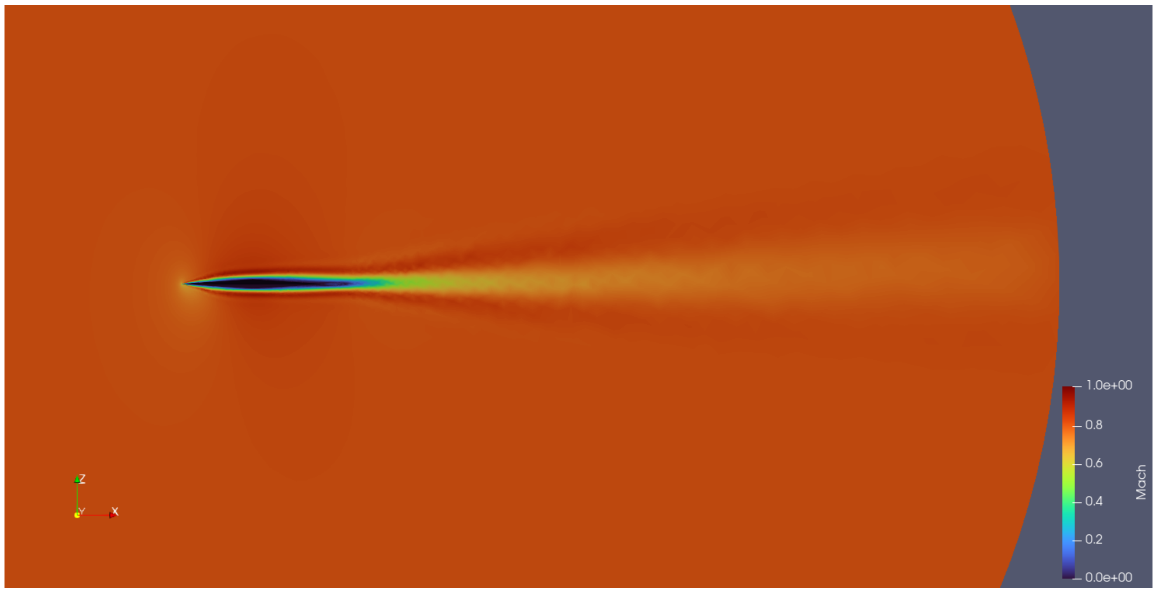
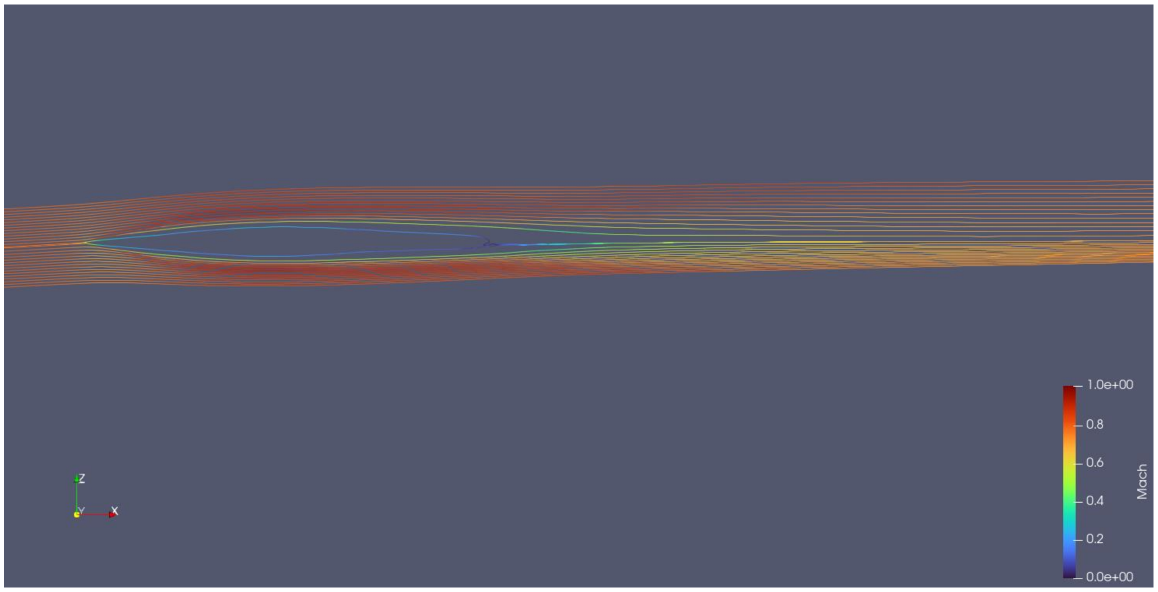


Figure 50: Analyses results for 0.5 Ma. (a) Mach number distribution, (b) streamlines along the wing, (c) Tip vortex, (d) Pressure distribution at the bottom side of the wing, (e) Pressure distribution at the top side of the wing.

Analyses results for 0.8 Ma (Figure 51). The C_L/C_D ratio is found to be 15,29286289.





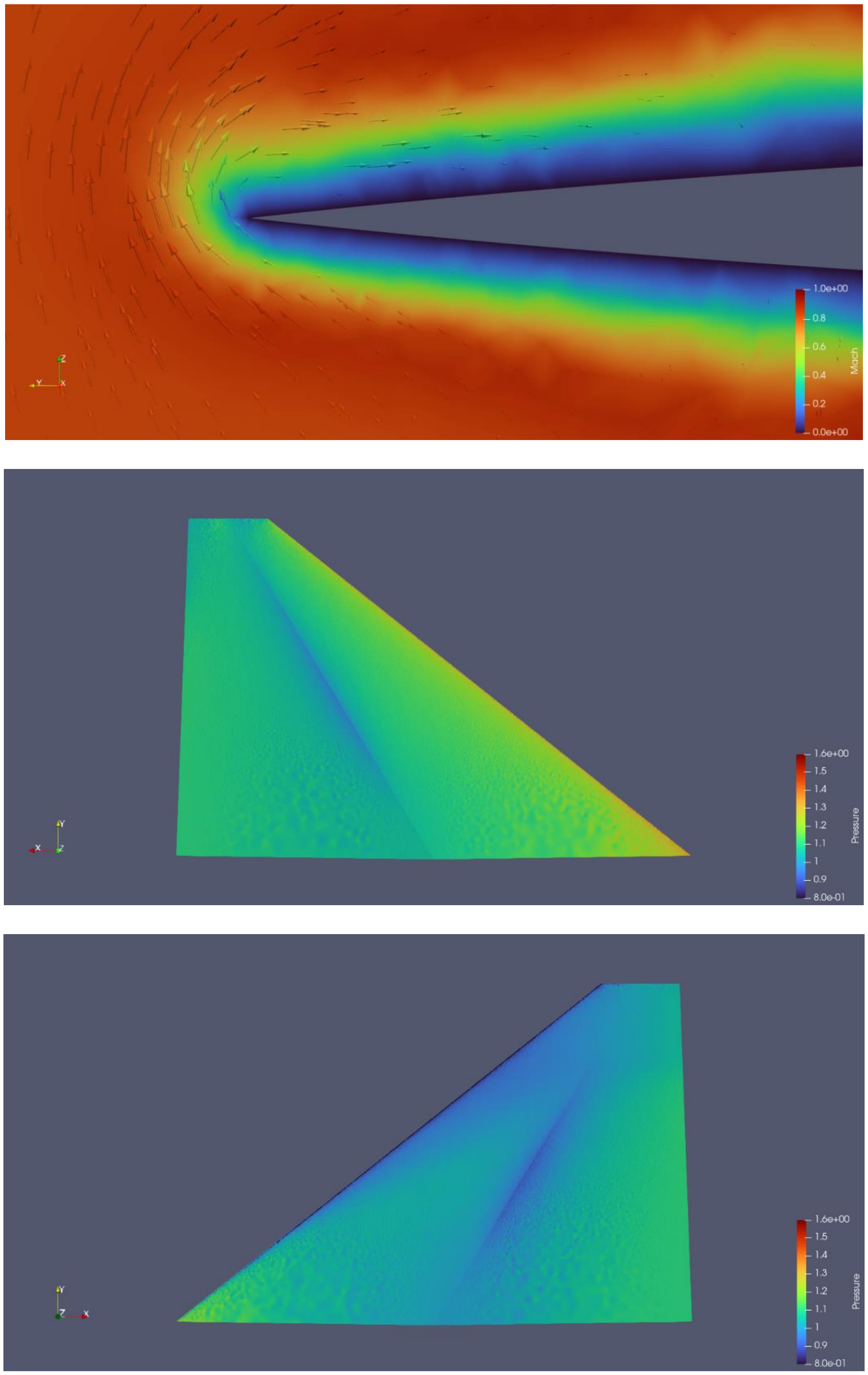
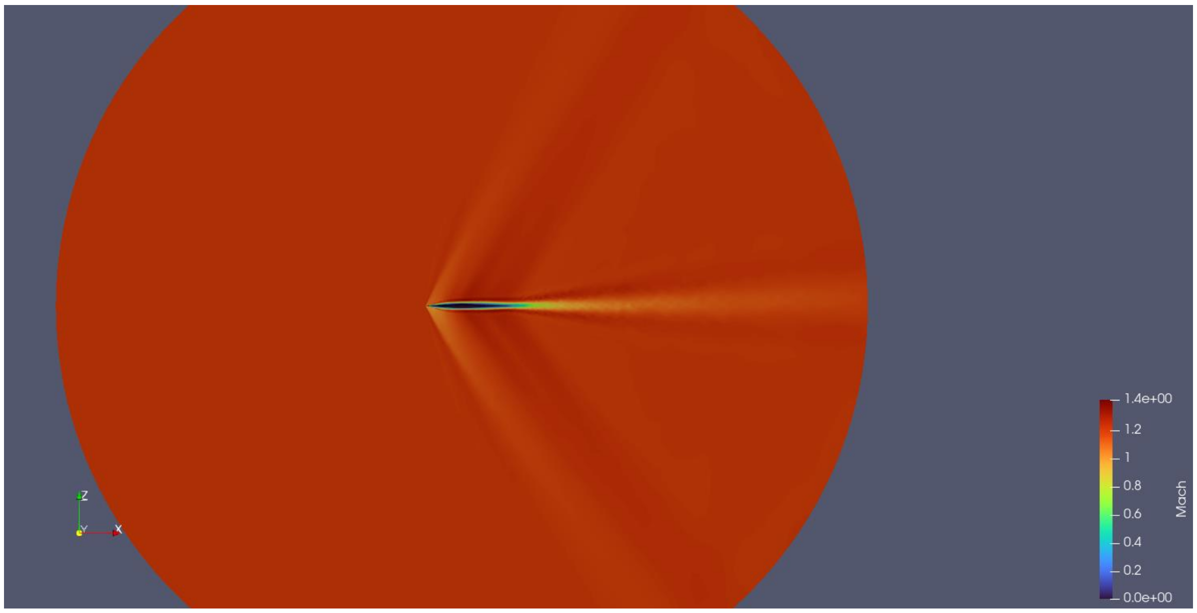
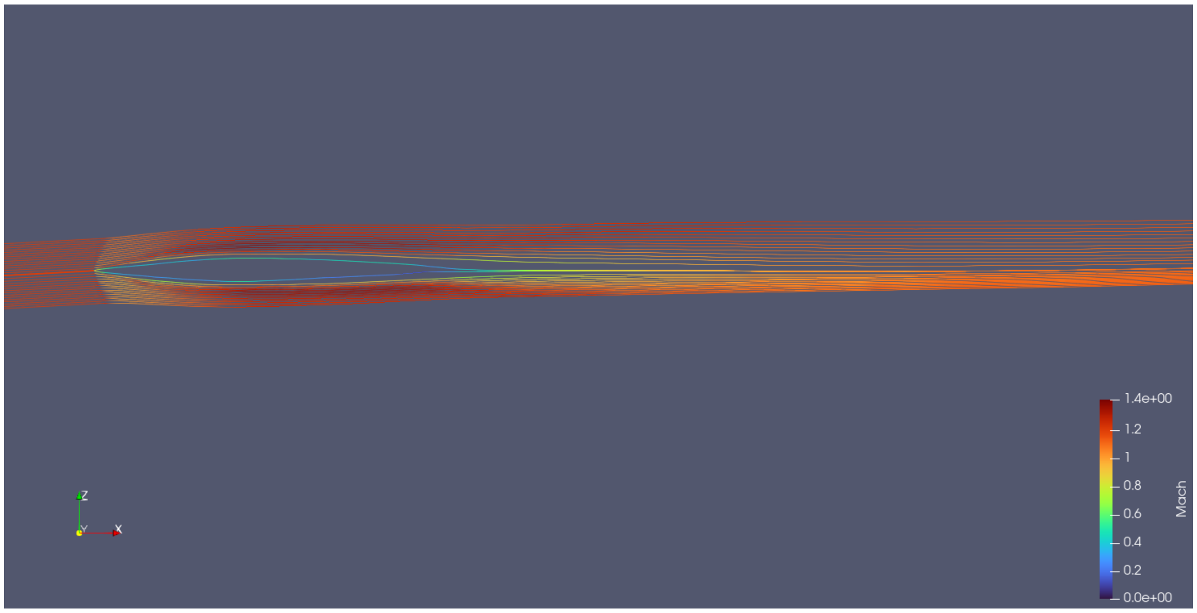


Figure 51: Analyses results for 0.8 Ma. (a) Mach number distribution, (b) streamlines along the wing, (c) Tip vortex, (d) Pressure distribution at the bottom side of the wing, (e) Pressure distribution at the top side of the wing.

Analyses results for 1.2 Ma (Figure 52). The C_L/C_D ratio is found to be 7,524693153.





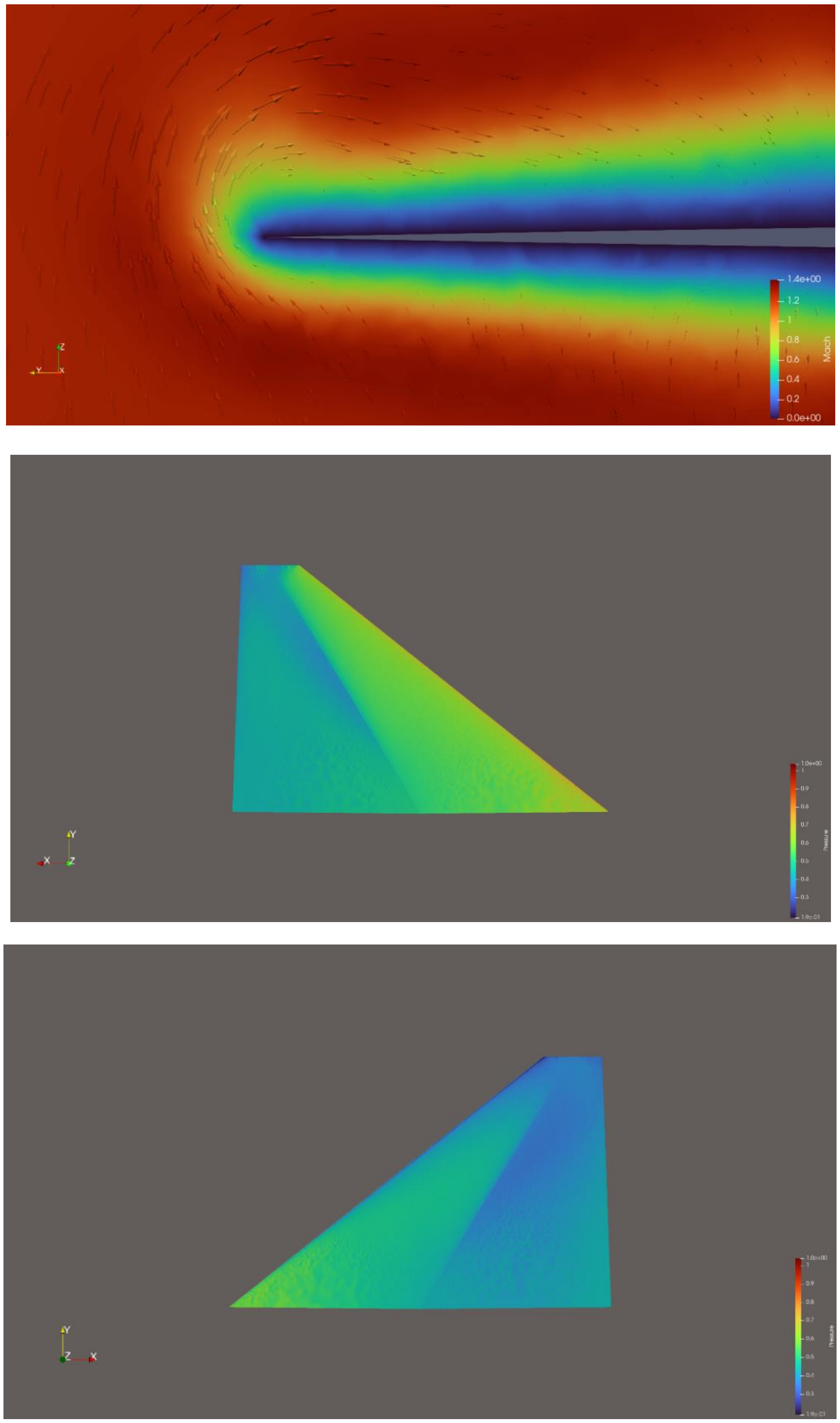
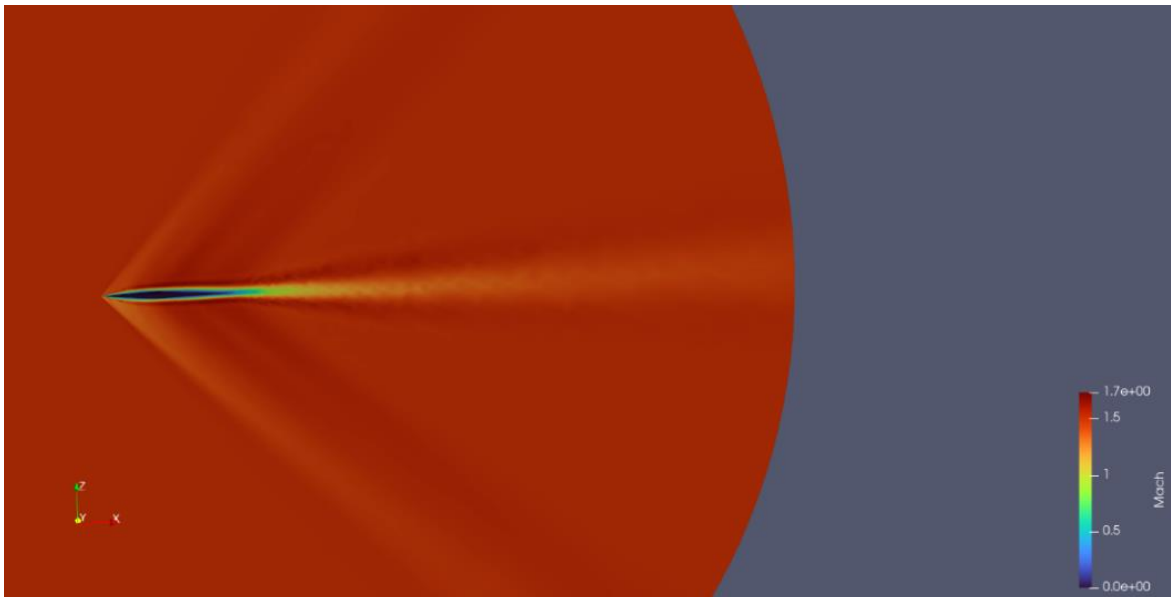
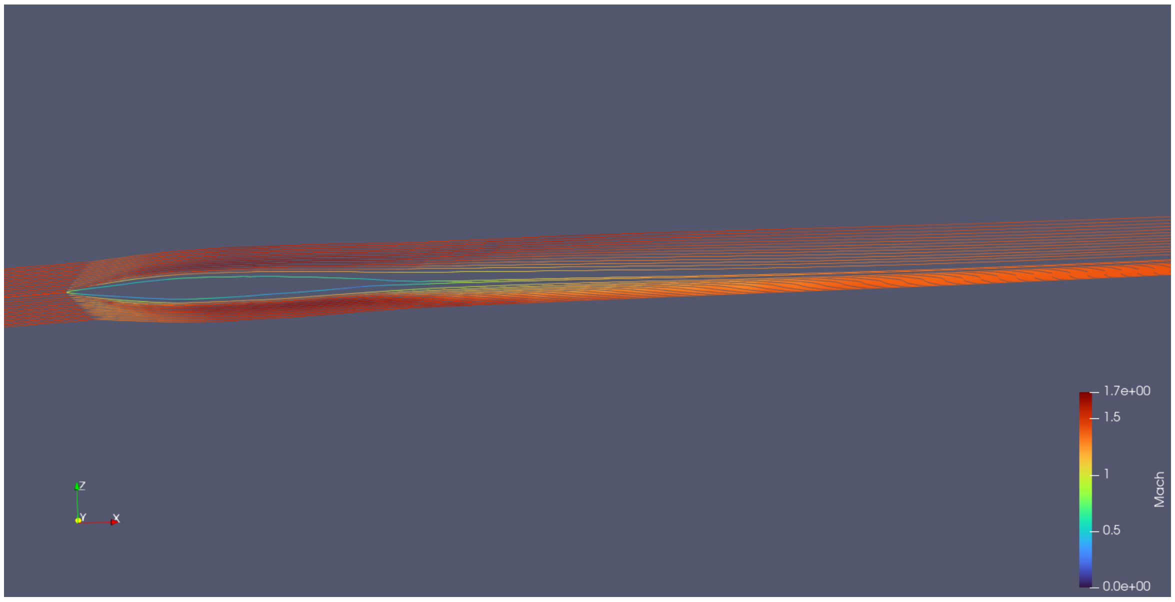
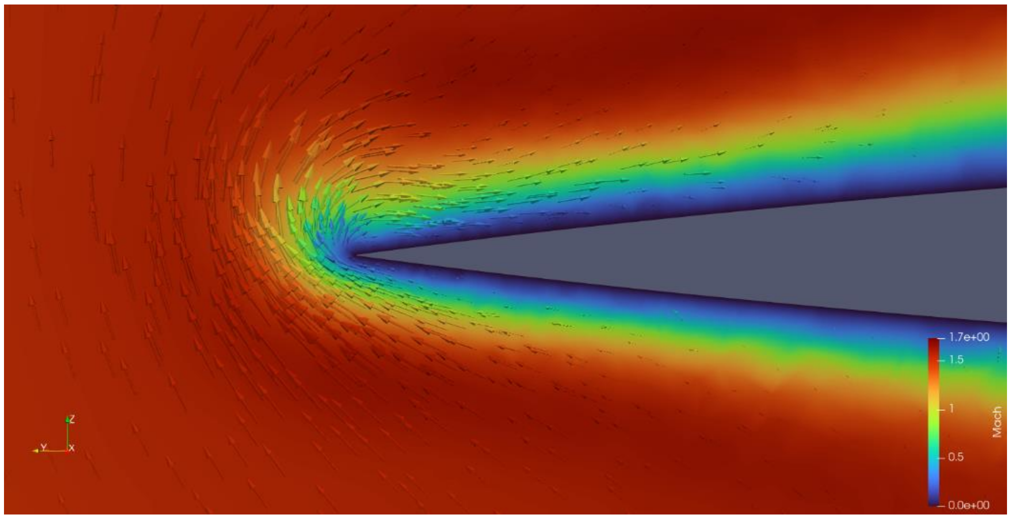


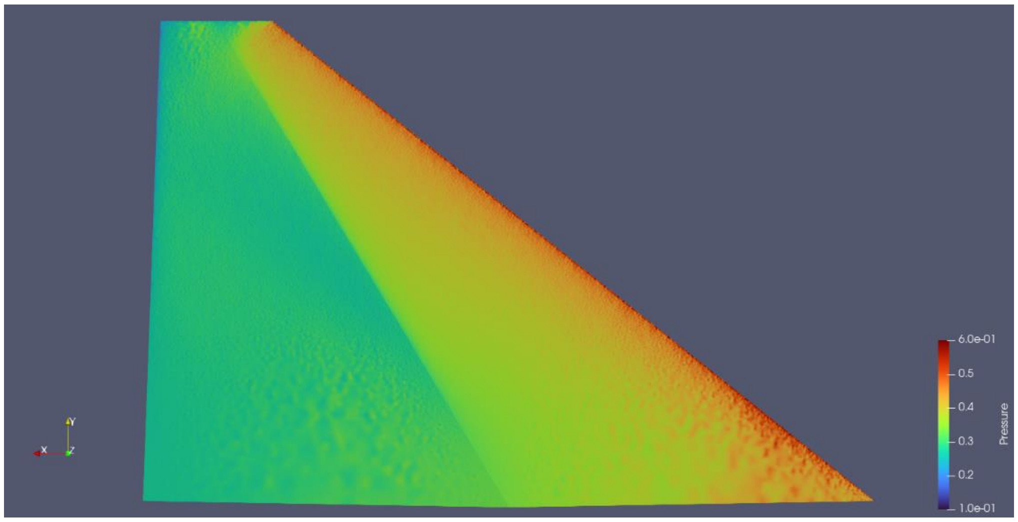
Figure 52: Analyses results for 1.2 Ma. (a) Mach number distribution, (b) streamlines along the wing, (c) Tip vortex, (d) Pressure distribution at the bottom side of the wing, (e) Pressure distribution at the top side of the wing.

Analyses results for 1.5 Ma (Figure 53). The C_L/C_D ratio is found to be 7,891703681.









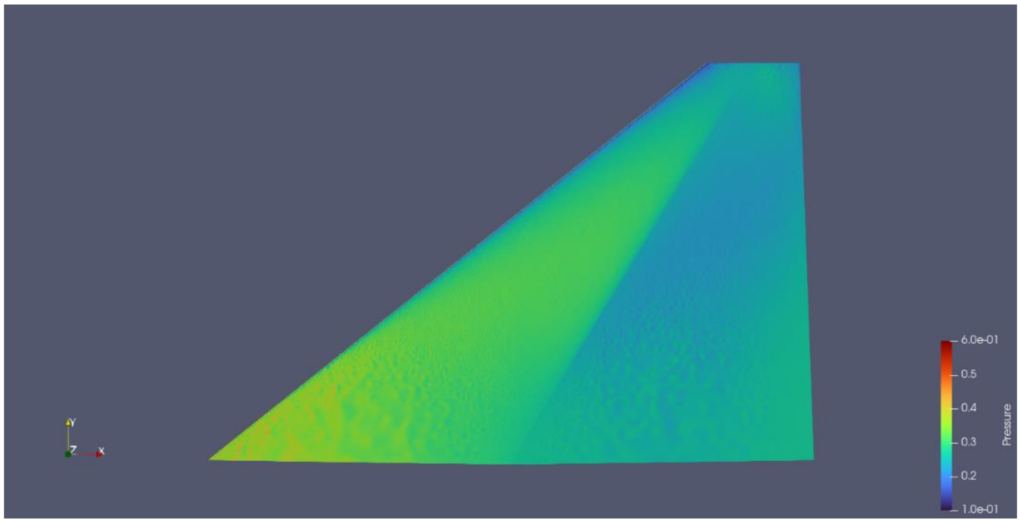


Figure 53: Analyses results for 1.5 Ma. (a) Mach number distribution, (b) streamlines along the wing, (c) Tip vortex, (d) Pressure distribution at the bottom side of the wing, (e) Pressure distribution at the top side of the wing.

From the Mach domain figures ((a) Figure 52, Figure 53), Oblique shocks are seen around the wing which spreads after the wing tip. This shock also increases as the domain Mach number increases to 1.5 from 1.2 as expected.

From the streamline figures ((b) of Figure 50, Figure 51, Figure 52, Figure 53), there is a separation at the trailing end of the wing. This separation can be commented to decrease at higher speeds. Due to this separation vortexes occur at these areas.

The tip vortexes ((c) of Figure 50, Figure 51, Figure 52, Figure 53) occurred as expected along the tip of the wing.

From these figures ((d) and (e) of Figure 50, Figure 51, Figure 52, Figure 53), it can be observed that there is a pressure difference between the top and bottom side of the wing. This pressure difference causes the lift as the Bernoulli Principle explains.

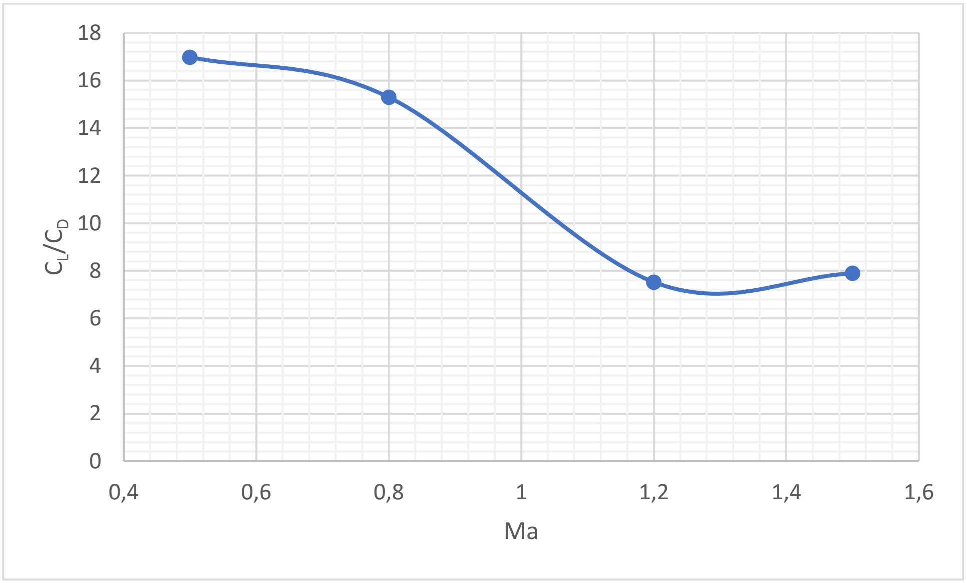


Figure 54: Graph of CL/CD ratios vs Mach number.

From Figure 54 we see that the C_L/C_D ratio decreases in supersonic flow compared to subsonic flow. This is expected to happen since C_D is expected to skyrocket after 0.8 Ma, climax in 1.0 Ma and start to decrease. It is also seen that C_D at 1.5 Ma is still greater than C_D at 0.5 Ma

Our test case could be interpreted as successful since it ran flawlessly from start to finish. It used each program as it was aimed at. The aerodynamic analyses were done at 0.5, 0.8, 1.2, 1.5

Ma. The analyses results were in line with the results published at (Devasurya, Nair, & Zubair, 2022). After optimizing, we found that sample 15 has the optimal wing shape with a weighted average of these ratios as 9,03116667. The parameters of sample 15 were:

- c=7.053965322252924 [m]
- Θ_{sweep}=51.48553849529085 [°]
- b=9.235578622070541 [m]
- $\Theta_{trailing}$ =2.0839590460390816 [°]
- t=0.47451889966833144 [m]

5. CONCLUSION

This final thesis project aimed to develop a Python based software that can design delta wings with input parameters, mesh, analyze and optimize delta wing shapes automatically with basic method which takes weighted averages of Cl/Cd ratios. Since the software is Python based, we used open-source programmers for each step of the project: FreeCAD for solid wing modelling, GMSH for mesh generation and SU2 for CFD analysis. These tools were combined into a single python program with a user-friendly interface. The program is run from gui.py file. The GUI allowed the user to enter input parameters for wing design. Many types of wings can be designed on this software, mesh and analyze them with small changes in configuration file.

In our sample case, sample size is determined as 20. First, we applied Latin Hypercube Sampling method to create samples with parameters which are in between the input numbers. Then, we checked whether they are feasible values to draw a wing. After that, we generated geometries using FreeCAD. FreeCAD macro modules provided us to easily compile python codes to generate geometry and output CAD files in the ".step" format. These files then were imported into GMSH Python API and meshed. These meshes were done in a way that we had finer meshes along the important elements. Mesh was saved in the ".su2" format. CFD analyses were run on these meshes at different Mach numbers. Results were optimized using a weighted average method where the weights were taken as an input. After optimization, sample 15 was seen as the most optimized wing.

To get better results, the sample size could be increased. This would mean that more combinations of parameters could be tested. The mesh sizes could be decreased, by doing this a more accurate results could be achieved. Optimization could also be done by SU2’s ADJOINT optimization. This adjoint optimization could be done on the already picked most optimize wing and further improve its efficiency.

6. REFERENCES

Aleksandar Jankovic, G. C. (2021, Febuary). Designing the design of experiments (DOE)– An investigation on the influence of different factorial designs on the characterization of complex systems. *ELSEVIER*.

Anderson, J. D. (2017). *Fundamentals of Aerodynamics (6th ed.).* McGraw-Hill Education.

B.Durakovic. (2017, December). Design of experiments application,concepts,examples:State of the art. *Periodicals of Engineering and Natural Sciences*, pp. 421-439.

Courant, R., Friedrichs, K., & Lewy, H. (1967). On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 215-234.

Çengel, Y. A., & Cimbala, J. M. (2020). *Fluid mechanics: Fundamentals and applications (4th ed.).* McGraw-Hill Education.

Devasurya, D., Nair, A. P., & Zubair, A. (2022). Analytical Study of Double Wedge Airfoil for Supersonic Application and Shape Modification for Subsonic Applications. *IJARSCT*.

GMSH. (2025, 06 22). Retrieved from Library: https://gmsh.info/doc/texinfo/gmsh.html

Loftin, L. K. (1985). *Quest for Performance: the Evolution of Modern Aircraft.* Washington, D.C.: NASA.

Mendenhall, C. A. (1983). *Delta Wings Convair's high-speed planes of the fifties & sixties.* Osceola, Wis: WI: Motorbooks International.

Modo, U. P. (2017). Analyses of diamond-shaped and circular arc airfoils in supersonic wind tunnel airflows. *International Journal of Engineering and Applied Sciences*.

Pasha, M. S. (2023). CFD Analysis of Diamond Shape Airfoils at Supersonic Speec. *International Journal of Advanced Research in Science*, 311-378.

7. APPENDIX

7.1. Code of GUI

```
import sys
import tkinter as tk
from tkinter import filedialog, messagebox, PhotoImage
import numpy as np
from scipy.stats import qmc
import pandas as pd
import subprocess
import os
import webbrowser

freecad_python_path = ""
output_folder = ""
SU2_CFD_path = ""
paths_selected = 0
geometries_generated = 0

def select_paths():
    global freecad_python_path, output_folder, paths_selected, SU2_CFD_path
    messagebox.showinfo("FreeCAD Python", "Select the FreeCAD Python
executable.")
    freecad_python_path = filedialog.askopenfilename(title="Select FreeCAD
Python Executable",
                                                     filetypes=[("Python
Executable", "*.exe")])
    messagebox.showinfo("Output Folder", "Select output folder for FreeCAD
results.")
    output_folder = filedialog.askdirectory(title="Select Output Folder")
    messagebox.showinfo("SU2_CFD", "Select the SU2_CFD executable.")
    SU2_CFD_path = filedialog.askopenfilename(title="Select SU2_CFD
Executable",
                                                     filetypes=[("SU2_CFD
Executable", "*.exe")])
    paths_selected = 1
    if not (freecad_python_path and output_folder):
        messagebox.showerror("Error", "Both Pathways must be selected. Please
try again.")
def run_freecad_macro(csv_path):
    macro_path = os.path.join(os.path.dirname(__file__), "macro.FCMacro")
    if macro_path:
        subprocess.run([freecad_python_path, macro_path, csv_path])
def delta(c, b, sweep, trail):
    return c - b * (np.tan(np.radians(sweep)) + np.tan(np.radians(trail)))
def generate_and_run(entries, labels, sample_count_var, window):
    global geometries_generated
    try:
        bounds = []
        for entry in entries:
            vals = list(map(float, entry.get().split()))
            if len(vals) != 2:
                raise ValueError("Each field must contain min and max
values.")
            bounds.append(sorted(vals))
        param_bounds = np.array(bounds)
        sampler = qmc.LatinHypercube(d=param_bounds.shape[0])
        sample_count = int(sample_count_var.get())
        lhs_samples = sampler.random(n=sample_count)
        scaled = qmc.scale(lhs_samples, param_bounds[:, 0], param_bounds[:,
1])

        df = pd.DataFrame(scaled, columns=labels)
        df["Output Folder"] = output_folder
        csv_path = os.path.join(output_folder, "samples.csv")
        df.to_csv(csv_path, index=False)
        samples, trash = [], []
        for i in range(len(df)):
            c, sweep, span, trail, root = df.iloc[i, :5]
            b = span / 2
            d_val = delta(c, b, sweep, trail)
            row = list(df.iloc[i])
            if d_val >= 0:
                samples.append(row)
            else:
                trash.append(row)
        pd.DataFrame(samples,
columns=df.columns).to_csv(os.path.join(output_folder, "samples.csv"),
index=False)
        trash_df = pd.DataFrame(trash, columns=df.columns)
        if not trash_df.empty:
            trash_df.to_csv(os.path.join(output_folder, "trash.csv"),
index=False)
        total = len(samples) + len(trash)
        valid_pct = 100 * len(samples) / total if total > 0 else 0
        trash_pct = 100 * len(trash) / total if total > 0 else 0
        if valid_pct < 50.0:
            chord = trash_df["Chord Length"].values
            sweep = trash_df["Sweep Angle"].values
            span = trash_df["Wing Span"].values
            trail = trash_df["Trailing Angle"].values
            b = span / 2
            d0 = delta(chord, b, sweep, trail)
            delta_changes = {
                "Chord Length": np.mean(delta(chord * 1.05, b, sweep, trail) -
d0),
                "Sweep Angle": np.mean(delta(chord, b, sweep * 1.05, trail) -
d0),
                "Wing Span": np.mean(delta(chord, b * 1.05, sweep, trail) -
d0),
                "Trailing Angle": np.mean(delta(chord, b, sweep, trail * 1.05)
- d0)
            }
            suggestions = []
            for param, change in delta_changes.items():
                direction = "reduce" if change < 0 else "increase"
                suggestions.append(f"- {param}: consider to {direction}
(Δdelta {change:+.2f})")
            delta_tip = (
                    f"Only {valid_pct:.1f}% of the geometries are valid.\n"
                    "\nNote: Δdelta indicates the effect on wing profile
feasibility.\n"
```

```
                    "Delta = Chord Length - Span * (tan(Sweep Angle) +
tan(Trailing Angle)).\n"
                    "Positive delta → valid geometry, Negative delta → invalid
geometry.\n"
                    "\nTips to reduce invalid samples:\n" +
"\n".join(suggestions)
            )
            messagebox.showerror("Low Valid Geometry Rate", delta_tip)
            window.lift()
            return
        else:
            msg = f"{len(samples)} valid samples
({valid_pct:.1f}%).\n{len(trash)} invalid samples ({trash_pct:.1f}%)."
            messagebox.showinfo("Result", msg)
            run_freecad_macro(csv_path)
            geometries_generated = 1
            #window.destroy()
    except Exception as e:
        messagebox.showerror("Error", str(e))


labels = ["Chord Length", "Sweep Angle", "Wing Span", "Trailing Angle", "Root
Thickness"]
form = tk.Tk()
form.title("Single Delta Wing Optimization")
form.geometry("423x400")
form.resizable(True, False)
sample_count_var = tk.StringVar()
entries = []
column_width = 200
row_width = 40
for i, label in enumerate(labels):
    tk.Label(form, text=f"{label} (min max):", font='Arial 11').place(x=10,
y=10+row_width*i)
    ent = tk.Entry(form, font='Arial 12', width=10)
    ent.place(x=column_width, y=10+row_width*i)
    entries.append(ent)
tk.Label(form, text="Number of geometries:", font="Arial
11").place(x=10,y=10+row_width*len(labels))
tk.Entry(form, textvariable=sample_count_var, font="Arial 12",
width=10).place(x=column_width,y=10+row_width*len(labels))
mach_label = tk.Label(form, text="Operating Mach Number(s):", font="Arial 11")
mach_label.place(x=10, y=10+row_width*6)
mach_entry = tk.Entry(form, font="Arial 12", width=20)
mach_entry.place(x=column_width, y=10+row_width*6)
coefficients_label = tk.Label(form, text="Mach Coefficient(s):", font="Arial
11")
coefficients_label.place(x=10, y=10+row_width*7)
coefficients_entry = tk.Entry(form, font="Arial 12", width=20)
coefficients_entry.place(x=column_width, y=10+row_width*7)
def generate_geometry():
    if paths_selected == 0:
        messagebox.showerror("Error", "Please select paths by clicking the
'Select Paths' button.")
    else:
        generate_and_run(entries, labels, sample_count_var, form)
def generate_mesh():
    meshing_script = os.path.join(os.path.dirname(__file__), "meshing.py")
    if os.path.exists(meshing_script):
        subprocess.run([sys.executable, meshing_script, output_folder])
    else:
        messagebox.showerror("Error", "meshing.py not found.")
def analyse():
    analyse_script = os.path.join(os.path.dirname(__file__), "Analyse.py")
    optimization_script = os.path.join(os.path.dirname(__file__),
"optimization.py")
    if os.path.exists(analyse_script):
        mach_input=mach_entry.get()
        mach_list=mach_input.split()
        mach_arg=",".join(mach_list)
        subprocess.run([sys.executable, analyse_script, output_folder,
SU2_CFD_path,mach_arg])
        if os.path.exists(optimization_script):
            subprocess.run([sys.executable, optimization_script,
output_folder, mach_arg])
            optimization_script = os.path.join(os.path.dirname(__file__),
"optimization.py")
            if os.path.exists(optimization_script):
                coefficients_input = coefficients_entry.get()
                coefficients_list = coefficients_input.split()
                coefficients_arg = ",".join(coefficients_list)
                mach_input = mach_entry.get()
                mach_list = mach_input.split()
                mach_arg = ",".join(mach_list)
                subprocess.run([sys.executable, optimization_script,
output_folder, mach_arg, coefficients_arg])
    else:
        messagebox.showerror("Error", "Analyse.py not found.")
def continuous():
    generate_geometry()
    generate_mesh()
    analyse()
def open_website():
    webbrowser.open("https://me-eng.marmara.edu.tr/")

button_y= y=10+row_width*8
tk.Button(form, text="Generate Geometry", bg="#0071CE", fg="white",
          font="TimesNewRoman 12", command=generate_geometry).place(x=10,
y=button_y)
tk.Button(form, text="Generate Mesh", bg="#0071CE", fg="white",
          font="TimesNewRoman 12", command=generate_mesh).place(x=195,
y=button_y)
tk.Button(form, text="Analyse", bg="#0071CE", fg="white",
          font="TimesNewRoman 12", command=analyse).place(x=345, y=button_y)
tk.Button(form, text="Select Paths", bg="gray", fg="black",
          font="TimesNewRoman 12 bold", command=select_paths).place(x=305,
y=130)
photo=PhotoImage(file=os.path.join(os.path.dirname(__file__), "University of
Marmara.png"))
tk.Button(form, image=photo, command=open_website,bd=0).place(x=304,y=8)
tk.Button(form, text="Continuous", bg="gray", fg="black",
          font="TimesNewRoman 12 bold", command=continuous).place(x=305,
y=170)
```

```
form.mainloop()
```

7.2. Code of FreeCAD macro

```
import FreeCAD
import PartDesign
# import PartDesignGui
import Sketcher
import math
import Part
import pandas as pd
import os
import sys
import Mesh

csv_path = sys.argv[1]
base_folder = os.path.dirname(csv_path)
valid_sample_path = os.path.join(base_folder, "samples.csv")
df = pd.read_csv(valid_sample_path)
os.chdir(base_folder)
output_folder = os.path.join(os.getcwd(), "geometries")

if not os.path.exists(output_folder):
    os.makedirs(output_folder)

for index, row in df.iterrows():
    chordlength = row['Chord Length']
    sweptangle = row['Sweep Angle']
    wingspan = row['Wing Span']
    trailingangle = row['Trailing Angle']
    root_thickness = row['Root Thickness']
    b = wingspan / 2
    App.newDocument()
    App.activeDocument().addObject('PartDesign::Body', 'Body')
    App.ActiveDocument.getObject('Body').Label = 'Body'
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Body').newObject('Sketcher::SketchObject
', 'Sketch')
    App.getDocument('Unnamed').getObject('Sketch').AttachmentSupport = (
    App.getDocument('Unnamed').getObject('YZ_Plane'), [''])
    App.getDocument('Unnamed').getObject('Sketch').MapMode = 'FlatFace'
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vect
or(-15.155131, 0.000000, 0)), True)

App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constrai
nt('PointOnObject', 0, 1, -1))
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vect
or(17.294466, 0.000000, 0)), True)
    App.ActiveDocument.recompute()
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constrai
nt('Symmetric', 0, 1, 1, 1, -1, 1))
    App.ActiveDocument.recompute()
    App.getDocument('Unnamed').getObject('Sketch').addConstraint(
        Sketcher.Constraint('DistanceX', 0, 1, 1, 1, 32.148740))
    App.getDocument('Unnamed').getObject('Sketch').setDatum(2,
App.Units.Quantity(str(chordlength)))
    App.ActiveDocument.recompute()
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vect
or(0.000000, 15.260094, 0)), True)

App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constrai
nt('PointOnObject', 2, 1, -2))
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch').addGeometry(Part.Point(App.Vect
or(0.000000, -16.088818, 0)), True)
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constrai
nt('Symmetric', 2, 1, 3, 1, -1, 1))
    App.ActiveDocument.recompute()
    App.getDocument('Unnamed').getObject('Sketch').addConstraint(
        Sketcher.Constraint('DistanceY', 3, 1, 2, 1, 31.611951))
    App.getDocument('Unnamed').getObject('Sketch').setDatum(5,
App.Units.Quantity(str(root_thickness)))
    App.ActiveDocument.recompute()
    App.ActiveDocument.recompute()
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(0.000000, 7.500000, 0.000000),
App.Vector(50.000000, -0.000000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch').addGeometry(geoList, False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 4, 1, 2, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 4, 2, 1, 1))

App.getDocument('Unnamed').getObject('Sketch').addConstraint(constraintList)
    del constraintList
    App.ActiveDocument.recompute()
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(50.000000, -0.000000, 0.000000),
App.Vector(0.000000, -7.500000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch').addGeometry(geoList, False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 5, 1, 1, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 5, 2, 3, 1))

App.getDocument('Unnamed').getObject('Sketch').addConstraint(constraintList)
    del constraintList
    App.ActiveDocument.recompute()
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(-50.000000, 0.000000, 0.000000),
App.Vector(0.000000, 7.500000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch').addGeometry(geoList, False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 6, 1, 0, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 6, 2, 2, 1))

App.getDocument('Unnamed').getObject('Sketch').addConstraint(constraintList)
    del constraintList
    App.ActiveDocument.recompute()
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(0.000000, -7.500000, 0.000000),
App.Vector(-50.000000, 0.000000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch').addGeometry(geoList, False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 7, 1, 3, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 7, 2, 0, 1))

App.getDocument('Unnamed').getObject('Sketch').addConstraint(constraintList)
    del constraintList
    App.ActiveDocument.recompute()
    tipthickness = 0.001
    rootsweeping = chordlength / 2
    roottrailing = chordlength / 2
    tipsweepingdistance = b * math.tan(math.radians(sweptangle))
    tiptrailingdistance = b * math.tan(math.radians(trailingangle))

App.getDocument('Unnamed').getObject('Body').newObject('PartDesign::Plane',
'DatumPlane')
    App.getDocument('Unnamed').getObject('DatumPlane').AttachmentSupport = [
        (App.getDocument('Unnamed').getObject('XZ_Plane'), '')]
    App.getDocument('Unnamed').getObject('DatumPlane').MapMode = 'FlatFace'
    App.activeDocument().recompute()
    App.getDocument('Unnamed').getObject('DatumPlane').AttachmentOffset =
App.Placement(
        App.Vector(0.0000000000, 0.0000000000, b), App.Rotation(0.0000000000,
0.0000000000, 0.0000000000))
    App.getDocument('Unnamed').getObject('DatumPlane').MapReversed = False
    App.getDocument('Unnamed').getObject('DatumPlane').AttachmentSupport = [
        (App.getDocument('Unnamed').getObject('YZ_Plane'), '')]
    App.getDocument('Unnamed').getObject('DatumPlane').MapPathParameter =
0.000000
    App.getDocument('Unnamed').getObject('DatumPlane').MapMode = 'FlatFace'
    App.getDocument('Unnamed').getObject('DatumPlane').recompute()

App.getDocument('Unnamed').getObject('Body').newObject('Sketcher::SketchObject
', 'Sketch001')
    App.getDocument('Unnamed').getObject('Sketch001').AttachmentSupport = (
    App.getDocument('Unnamed').getObject('DatumPlane'), [''])
    App.getDocument('Unnamed').getObject('Sketch001').MapMode = 'FlatFace'
    App.ActiveDocument.recompute()

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(Part.Point(App.V
ector(-25.093729, 0.000000, 0)), True)

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(Sketcher.Const
raint('PointOnObject', 0, 1, -1))
    App.getDocument('Unnamed').getObject('Sketch001').addConstraint(
        Sketcher.Constraint('DistanceX', 0, 1, -1, 1, 25.093729))
    App.getDocument('Unnamed').getObject('Sketch001').setDatum(1,
App.Units.Quantity(str(rootsweeping)))

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(Part.Point(App.V
ector(24.770006, 0.000000, 0)), True)

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(Sketcher.Const
raint('PointOnObject', 1, 1, -1))
    App.getDocument('Unnamed').getObject('Sketch001').addConstraint(
        Sketcher.Constraint('DistanceX', -1, 1, 1, 1, 24.770006))
    App.getDocument('Unnamed').getObject('Sketch001').setDatum(3,
App.Units.Quantity(str(roottrailing)))

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(Part.Point(App.V
ector(-11.975457, 0.000000, 0)), True)

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(Sketcher.Const
raint('PointOnObject', 2, 1, -1))
    App.getDocument('Unnamed').getObject('Sketch001').addConstraint(
        Sketcher.Constraint('DistanceX', 0, 1, 2, 1, 13.024543))
    App.getDocument('Unnamed').getObject('Sketch001').setDatum(5,
App.Units.Quantity(str(tipsweepingdistance)))

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(Part.Point(App.V
ector(16.674252, 0.000000, 0)), True)

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(Sketcher.Const
raint('PointOnObject', 3, 1, -1))
    App.getDocument('Unnamed').getObject('Sketch001').addConstraint(
        Sketcher.Constraint('DistanceX', 3, 1, 1, 1, 8.325748))
    App.getDocument('Unnamed').getObject('Sketch001').setDatum(7,
App.Units.Quantity(str(tiptrailingdistance)))
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    constrGeoList = []
    constrGeoList.append(
        Part.LineSegment(App.Vector(-5.000000, 0.000000, 0.000000),
App.Vector(15.000000, 0.000000, 0.000000)))
App.getDocument('Unnamed').getObject('Sketch001').addGeometry(constrGeoList,
True)
    del constrGeoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 4, 1, 2, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 4, 2, 3, 1))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    constrGeoList = []
    constrGeoList.append(
        Part.LineSegment(App.Vector(5.000000, 0.000000, 0.000000),
App.Vector(5.120581, 4.705888, 0.000000)))

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(constrGeoList,
True)
    del constrGeoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Symmetric', 4, 1, 4, 2, 5, 1))
    constraintList.append(Sketcher.Constraint('Vertical', 5))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList

App.getDocument('Unnamed').getObject('Sketch001').addGeometry(Part.Point(App.V
ector(4.709064, -3.730186, 0)), True)

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(Sketcher.Const
raint('Symmetric', 5, 2, 6, 1, 4))
    App.getDocument('Unnamed').getObject('Sketch001').addConstraint(
        Sketcher.Constraint('DistanceY', 6, 1, 5, 2, 7.840572))
    App.getDocument('Unnamed').getObject('Sketch001').setDatum(13,
App.Units.Quantity(str(tipthickness)))
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(-5.000000, 0.000000, 0.000000),
App.Vector(5.000000, 2.500000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch001').addGeometry(geoList,
False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 7, 1, 2, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 7, 2, 5, 2))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(5.000000, 2.500000, 0.000000),
App.Vector(15.000000, 0.000000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch001').addGeometry(geoList,
False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 8, 1, 5, 2))
    constraintList.append(Sketcher.Constraint('Coincident', 8, 2, 3, 1))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(15.000000, 0.000000, 0.000000),
App.Vector(5.000000, -2.500000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch001').addGeometry(geoList,
False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 9, 1, 3, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 9, 2, 6, 1))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList
    ActiveSketch = App.getDocument('Unnamed').getObject('Sketch001')
    lastGeoId = len(ActiveSketch.Geometry)
    geoList = []
    geoList.append(
        Part.LineSegment(App.Vector(5.000000, -2.500000, 0.000000),
App.Vector(-5.000000, 0.000000, 0.000000)))
    App.getDocument('Unnamed').getObject('Sketch001').addGeometry(geoList,
False)
    del geoList
    constraintList = []
    constraintList = []
    constraintList.append(Sketcher.Constraint('Coincident', 10, 1, 6, 1))
    constraintList.append(Sketcher.Constraint('Coincident', 10, 2, 2, 1))

App.getDocument('Unnamed').getObject('Sketch001').addConstraint(constraintList
)
    del constraintList
    App.ActiveDocument.recompute()
    App.ActiveDocument.recompute()
    loft =
App.getDocument('Unnamed').getObject('Body').newObject('PartDesign::AdditiveLo
ft', 'AdditiveLoft')
    App.getDocument('Unnamed').getObject('AdditiveLoft').Profile =
App.getDocument('Unnamed').getObject('Sketch')
```

```
    App.getDocument('Unnamed').getObject('AdditiveLoft').Sections = 
App.getDocument('Unnamed').getObject('Sketch001')
    App.ActiveDocument.recompute()
    App.getDocument('Unnamed').getObject('Sketch').Visibility = False
    loft.Ruled = False
    loft.Closed = True
    App.ActiveDocument.recompute()
    App.getDocument('Unnamed').recompute()
    App.getDocument('Unnamed').getObject('Sketch').Visibility = False
    App.getDocument('Unnamed').getObject('Sketch001').Visibility = False
    App.getDocument('Unnamed').getObject('DatumPlane').Visibility = False

    save_path_fcstd = os.path.join(output_folder, f"Delta_Wing_Geometry_{index 
+ 1}.FCStd")
    save_path_step = os.path.join(output_folder, f"Delta_Wing_Geometry_{index 
+ 1}.step")

    App.getDocument('Unnamed').Body.Placement = App.Placement(App.Vector(0, 0, 
0),

App.Rotation(App.Vector(0.81, 0.81, 0), 180),
                                                              App.Vector(0, 0, 
0))

    App.ActiveDocument.recompute()

    App.getDocument('Unnamed').saveAs(save_path_fcstd)

    step_part = App.getDocument('Unnamed').getObject('Body')
    Part.export([step_part], save_path_step)

    FreeCAD.closeDocument(App.getDocument('Unnamed').Name)
```

7.3. Code of Meshing

```
import time
import gmsh
import os
import numpy as np
import pandas as pd
import math
import sys

output_folder = sys.argv[1]
base_folder = os.path.dirname(output_folder)
valid_path = output_folder + "/samples.csv"
geometries_path =os.path.join(output_folder, "geometries")
os.chdir(output_folder)
df = pd.read_csv(valid_path)

gmsh.initialize()
mesh_sizes = [0.01]

for index in range(0,4):
    for meshsizemin in mesh_sizes:
        sweep_ang = df.iat[index,1]
        b = df.iat[index,2]/2
        leading_edge= b/math.cos(math.radians(sweep_ang))
        n=leading_edge/meshsizemin
        n_other=b/meshsizemin
        start=time.time()

        gmsh.model.add("OpenCASCADE")

gmsh.model.occ.importShapes(os.path.join(geometries_path,f"Delta_Wing_Geometry
_{index + 1}.step"))

        gmsh.model.occ.addSphere(0, 0, 0, 50, 2)
        gmsh.model.occ.addBox(50, 0, 50, -100, -50, -100, 3)

        gmsh.model.occ.cut([(3, 2)], [(3, 3)], 4)
        gmsh.model.occ.cut([(3, 4)], [(3, 1)], 5)

        gmsh.model.occ.synchronize()

        gmsh.model.addPhysicalGroup(2, [1], name="FARFIELD")
        gmsh.model.addPhysicalGroup(2, [3, 4, 5, 6, 7], name="WING")
        gmsh.model.addPhysicalGroup(3, [5], name="VOLUME")
        gmsh.model.addPhysicalGroup(2, [2], name="SYMMETRY")

        dist_field = gmsh.model.mesh.field.add("Distance")
        gmsh.model.mesh.field.setNumbers(dist_field, "PointsList", [5, 6, 7,
8, 9, 10, 11, 12])
        gmsh.model.mesh.field.setNumbers(dist_field, "CurvesList", [11,15,17])
        gmsh.model.mesh.setTransfiniteCurve(13, int(n))
        gmsh.model.mesh.setTransfiniteCurve(15, int((n_other+n)/2))
        gmsh.model.mesh.setTransfiniteCurve(11, int((n_other+n)/2))

        thr_field = gmsh.model.mesh.field.add("Threshold")
        gmsh.model.mesh.field.setNumber(thr_field, "InField", dist_field)

        gmsh.model.mesh.field.setNumber(thr_field, "SizeMin", meshsizemin)
        gmsh.model.mesh.field.setNumber(thr_field, "SizeMax", 2.0)
        gmsh.model.mesh.field.setNumber(thr_field, "DistMin", 0.0)
        gmsh.model.mesh.field.setNumber(thr_field, "DistMax", 20.0)
        gmsh.model.mesh.field.setAsBackgroundMesh(thr_field)

        gmsh.option.setNumber("Mesh.AngleToleranceFacetOverlap", 0.001)

        gmsh.model.mesh.generate(3)
        sample_name = f"sample_{index+1}"
        folder_path = os.path.join(os.getcwd(), sample_name)
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)
        mesh_file_name = f"mesh_{sample_name}.su2"
        mesh_file_path = os.path.join(folder_path, mesh_file_name)
        gmsh.write(mesh_file_path)

gmsh.finalize()
```

7.4. Code of Analysis

```
import sys
import os
import subprocess
import shutil
import time

output_folder = sys.argv[1]
SU2_path = sys.argv[2]
mach_arg = sys.argv[3]
mach_numbers = [float(x) for x in mach_arg.split(",") if x.strip() != '']
list=os.listdir(output_folder)
sample_names = [file for file in list if "sample_" in file]


for index in range(len(sample_names)):
    sample_name = sample_names[index]
    sample_folder = os.path.join(output_folder, sample_name)

    mesh_file_name = f"mesh_{sample_name}.su2"
    mesh_file_path = os.path.join(sample_folder, mesh_file_name)

    for ma in mach_numbers:
        mach_folder_name = f"{sample_name}_at_{ma}_Ma"
        mach_folder = os.path.join(sample_folder, mach_folder_name)
        if not os.path.exists(mach_folder):
            os.makedirs(mach_folder)

        mach_str = f"{ma:.1f}".replace(".", "_")

        cfg_content =
f"""%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
%
%
% SU2 configuration file
%
% Case description: Turbulent flow past the ONERA M6 wing
%
% Author: Thomas D. Economon
%
% Institution: Stanford University
%
% Date: 2014.06.14
%
% File Version 5.0.0 "Raven"
%
%
%

SOLVER= RANS
KIND_TURB_MODEL= SA
MATH_PROBLEM= DIRECT
RESTART_SOL= NO
```

```
MACH_NUMBER= {ma}
AOA= 3.06
SIDESLIP_ANGLE= 0.0
INIT_OPTION= REYNOLDS
FREESTREAM_OPTION= TEMPERATURE_FS
FREESTREAM_TEMPERATURE= 288.15
REYNOLDS_NUMBER= 11.72E6
REYNOLDS_LENGTH= 0.64607
FLUID_MODEL= STANDARD_AIR
GAMMA_VALUE= 1.4
GAS_CONSTANT= 287.058
VISCOSITY_MODEL= SUTHERLAND
MU_REF= 1.716E-5
MU_T_REF= 273.15
SUTHERLAND_CONSTANT= 110.4
CONDUCTIVITY_MODEL= CONSTANT_PRANDTL
PRANDTL_LAM= 0.72
PRANDTL_TURB= 0.90
REF_ORIGIN_MOMENT_X = 0.25
REF_ORIGIN_MOMENT_Y = 0.00
REF_ORIGIN_MOMENT_Z = 0.00
REF_LENGTH= 0.64607
REF_AREA= 0
REF_DIMENSIONALIZATION= FREESTREAM_VEL_EQ_ONE
MARKER_HEATFLUX= ( WING, 0.0 )
MARKER_FAR= ( FARFIELD )
MARKER_SYM= ( SYMMETRY )
MARKER_PLOTTING= ( WING )
MARKER_MONITORING= ( WING )
NUM_METHOD_GRAD= GREEN_GAUSS
CFL_NUMBER= 100
CFL_ADAPT= YES
CFL_ADAPT_PARAM= ( 0.5, 1.5, 10.0, 100000.0 )
RK_ALPHA_COEFF= ( 0.66667, 0.66667, 1.000000 )
ITER= 999999
LINEAR_SOLVER= FGMRES
LINEAR_SOLVER_PREC= ILU
LINEAR_SOLVER_ERROR= 1E-10
LINEAR_SOLVER_ITER= 10
MGLEVEL= 0
MGCYCLE= V_CYCLE
MG_PRE_SMOOTH= ( 1, 1, 1, 1 )
MG_POST_SMOOTH= ( 0, 0, 0, 0 )
MG_CORRECTION_SMOOTH= ( 0, 0, 0, 0 )
MG_DAMP_RESTRICTION= 0.7
MG_DAMP_PROLONGATION= 0.7
CONV_NUM_METHOD_FLOW= JST
JST_SENSOR_COEFF= ( 0.5, 0.02 )
TIME_DISCRE_FLOW= EULER_IMPLICIT
CONV_NUM_METHOD_TURB= SCALAR_UPWIND
MUSCL_TURB= NO
SLOPE_LIMITER_TURB= VENKATAKRISHNAN
TIME_DISCRE_TURB= EULER_IMPLICIT
CONV_FIELD= DRAG
CONV_STARTITER= 10
CONV_CAUCHY_ELEMS= 100
CONV_CAUCHY_EPS= 1E-6
```

```
MESH_FILENAME= {mesh_file_name}
MESH_FORMAT= SU2
MESH_OUT_FILENAME= mesh_out.su2
SOLUTION_FILENAME= {sample_name}_solution_flow_{mach_str}.dat
SURFACE_ADJ_FILENAME= {sample_name}_surface_adjoint_{mach_str}.dat
TABULAR_FORMAT= CSV
CONV_FILENAME= {sample_name}_history_{mach_str}Ma
RESTART_FILENAME= restart_flow.dat
RESTART_ADJ_FILENAME= restart_adj.dat
VOLUME_FILENAME= {sample_name}_flow_{mach_str}Ma
VOLUME_ADJ_FILENAME= adjoint
GRAD_OBJFUNC_FILENAME= of_grad.dat
SURFACE_FILENAME= {sample_name}_surface_flow_{mach_str}Ma
OUTPUT_FILES= (RESTART, PARAVIEW, SURFACE_PARAVIEW)
HISTORY_OUTPUT= (ITER, RMS_RES, LIFT, DRAG)
SCREEN_OUTPUT= (INNER_ITER, WALL_TIME, RMS_DENSITY, RMS_NU_TILDE, LIFT, DRAG)
"""

        config_name = f"config_for_{mach_str}Ma.cfg"
        cfg_path = os.path.join(sample_folder, config_name)

        with open(cfg_path, "w") as cfg_file:
            cfg_file.write(cfg_content)

        print(f"[INFO] Running SU2 for {sample_name} at Mach {ma}...")
        subprocess.run([SU2_path, config_name], cwd=sample_folder)
        print(f"[INFO] SU2 run finished for {sample_name} at Mach {ma}.")

        time.sleep(2)

        files_to_move = [
            f"{sample_name}_history_{mach_str}Ma.csv",
            f"{sample_name}_flow_{mach_str}Ma.vtu",
            f"{sample_name}_surface_flow_{mach_str}Ma.vtu",
            f"config_for_{mach_str}Ma.cfg",
            f"{sample_name}_solution_flow_{mach_str}.dat",
            f"{sample_name}_surface_adjoint_{mach_str}.dat"
        ]

        print(f"[DEBUG] Looking for files to move for {sample_name} at Mach
{ma}:")
        for filename in files_to_move:
            print(f"  Searching for: {filename}")

        for filename in files_to_move:
            src = os.path.join(sample_folder, filename)
            dst = os.path.join(mach_folder, filename)

            if os.path.exists(src):
                try:
                    shutil.move(src, dst)
                    print(f"[INFO] Moved {filename} to {mach_folder}")
                except Exception as e:
                    print(f"[ERROR] Failed to move {filename}: {e}")
            else:
                print(f"[WARNING] {filename} not found in {sample_folder}")
```

7.5. Code of Optimization

```python
import sys
import os
import pandas as pd
import numpy as np

output_folder = sys.argv[1]
mach_arg = sys.argv[2]
mach_numbers = [float(x) for x in mach_arg.split(",") if x.strip() != '']
list=os.listdir(output_folder)
sample_names = [file for file in list if "sample_" in file]
coefficients=sys.argv[3]
coefficients_numbers = [float(x) for x in coefficients.split(",") if x.strip()
!= '']
data=[]

for index in range(len(sample_names)):
    sample_name = sample_names[index]
    current_directory=os.path.join(output_folder, sample_name)
    ratio=[]
    row=[]
    for i in range(len(mach_numbers)):
        ma=mach_numbers[i]
        mach_folder_name = f"{sample_name}_at_{ma}_Ma"
        mach_folder = os.path.join(current_directory, mach_folder_name)
        mach_str = f"{ma:.1f}".replace(".", "_")
        history_name=f"{sample_name}_history_{mach_str}Ma.csv"
        history_path=os.path.join(mach_folder,
f"{sample_name}_history_{mach_str}Ma.csv")
        hist=pd.read_csv(history_path)
        hist.columns = hist.columns.str.strip().str.strip('"')
        cl=hist["CL"].iloc[-1]
        cd=hist["CD"].iloc[-1]
        ratio.append(cl/cd)
    total=np.dot(ratio,coefficients_numbers)
    row.append(total)
    data.append(row)

df = pd.DataFrame(data,index=sample_names,columns=["TOTAL RATIO"])
df.to_csv(os.path.join(output_folder,"sample points.csv"), index_label="SAMPLE
NAME")
max_value = df["TOTAL RATIO"].max()
max_sample = df["TOTAL RATIO"].idxmax()
print(f"Max Ratio: {max_value}")
print(f"Most optimize Sample: {max_sample}")
```