

Rocket Fin Design and Optimization with Automation in Computational Fluid Dynamics

a project presented to
The Faculty of the Department of Aerospace Engineering
San Jose State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Vasant V. Chalemcherla

December 2025

approved by

Dr. Periklis Papadopoulos

Faculty Advisor

ABSTRACT

Rocket Fin Design and Optimization with Automation in Computational Fluid Dynamics

by Vasant V. Chalemcherla

Computational fluid dynamics simulations take large amounts of time to complete. This issue is compounded when multiple consecutive simulations are required, especially with optimization problems. Through the use of modern tools, this process can be accelerated to increase the yield of results. The PyAnsys Python package in conjunction with Ansys software can be used in such a manner to achieve this goal. The purpose of this project is to develop a Python executable that implements the above mentioned tools to design and optimize the fins of a rocket and compare them to existing results for accuracy.

Table of Contents

1.	Introduction	9
1.1	Motivation	9
1.2	Literature Review	9
1.3	Project Proposal.....	15
1.4	Methodology.....	15
2.	Understanding Computational Fluid Dynamics.....	16
2.1	Fundamental Equations	16
2.2	Assumptions and Simplifications	17
2.3	Mesh Generation	18
2.4	Convergence	19
2.5	Analytical System of Equations with Applied Assumptions	19
3.	Baseline Rocket Design	21
3.1	3D Model.....	21
3.2	Control Volume.....	22
4.	Meshing Approach	23
4.1	Meshing in Computational Fluid Dynamics.....	23
4.2	Meshing the Baseline Rocket Control Volume.....	23
5.	Baseline Rocket Simulation and Results.....	27
5.1	Simulation.....	27
5.2	Results	27
6.	CFD Software Automation.....	33
6.1	Ansys PyAnsys.....	33
6.2	Meshing with PyAnsys.....	33
6.3	PyFluent Core Package.....	34
6.4	PyFluent Parametric Package	35
6.5	Data Handling and Display.....	36
6.6	Drawbacks and Limitations	36
7.	Python Executable Usage and Demonstration	38
7.1	Requirements	38
7.2	Operation	38
8.	Optimization Trials.....	40
8.1	Fin Geometry Variations.....	40
8.2	Simulation Performance and Optimization Results.....	40
8.3	Analysis	42
9.	Conclusion.....	43

References.....	44
Appendix.....	46
Appendix A — Python Executable.....	46

List of Tables

Table 1.1 — Most impactful design parameters summarized [9].	12
Table 2.1 — Assumptions and their effects for the mathematics of CFD.....	18
Table 4.1 — Baseline rocket control volume mesh statistics.....	26
Table 8.1 — Fin design parameters.	40
Table 8.2 — CFD results for each fin tested at Mach 0.5	41
Table 8.3 — CFD results for each fin tested at Mach 1	41
Table 8.4 — CFD results for each fin tested at Mach 3	42

List of Figures

Figure 1.1 — Fin designs tested by Warsaw University of Technology students [1].	9
Figure 1.2 — MDO-SONDA interaction and process loop [6].	10
Figure 1.3 — Fin optimization results indicating drag force [6].	11
Figure 1.4 — Flutter mode example [8].	12
Figure 1.5 — PG-7 unguided rocket with stabiliazer fins deployed [11].	13
Figure 1.6 — Simple rocket free-body diagram with all forces labeled [10].	13
Figure 1.7 — Trapezoidal fin structured-multiblock fin [12].	14
Figure 1.8 — Lift-to-drag ratio plots from CFD results [12].	14
Figure 2.1 — Screen capture of Ansys Fluent solver and viscous model setup.	17
Figure 2.2 — Mesh convergence study example using von Mises stress [14].	19
Figure 3.1 — Power series curve for nose cone design.	21
Figure 3.2 — Isometric trapezoidal fin design.	21
Figure 3.3 — Complete 3D model of the baseline design.	22
Figure 3.4 — Control volume of the baseline design.	22
Figure 4.1 — Maximum element size specification.	24
Figure 4.2 — Sphere of influence element size specification.	24
Figure 4.3 — Match control periodic boundary specification.	25
Figure 4.4 — Initial mesh of quarter section of the baseline rocket design control volume.	25
Figure 5.1 — Velocity contour for the baseline rocket at Mach 0.5.	27
Figure 5.2 — Pressure contour for the baseline rocket at Mach 0.5.	28
Figure 5.3 — Velocity contour for the baseline rocket at Mach 1.0.	29
Figure 5.4 — Pressure contour for the baseline rocket at Mach 1.0.	30
Figure 5.5 — Velocity contour for the baseline rocket at Mach 1.5.	31
Figure 5.6 — Pressure contour for the baseline rocket at Mach 1.5.	32
Figure 6.1 — PyFluent TUI method meshing terminal results.	34
Figure 6.2 — PyAnsys PyFluent core package Settings API method demonstration.	35
Figure 6.3 — PyAnsys PyFluent parametric package Settings API method demonstration.	36
Figure 8.1 — Four fin designs tested using the Python executable.	40

SYMBOLS PAGE

Symbol	Definition	Units [SI]
ρ	Density	kg/m ³
V	Volume	m ³
$\vec{v}$	Velocity vector	m/s
v	Velocity magnitude	m/s
$\vec{A}$	Area vector	m ²
t	Time	s
$\vec{F}$	Force vector	N
e	Energy	J
$\dot{Q}$	Change in heat over time	J
W	Work	J
p	Pressure	Pa
R	Gas constant	J/kg-K
T	Temperature	K
ε	Error	
M	Mach number	
C	Coefficient	
Subscripts		
$()_x$	x -direction	
$()_y$	y -direction	
$()_z$	z -direction	
$()_p$	Pressure	
$()_\tau$	Skin-friction	
$()_g$	Gravity	
$()_{EM}$	Electromagnetic	
$()_{ext}$	External	
$()_D$	Drag	
$()_L$	Lift	
Acronyms		
CFD	Computational fluid dynamics	
MDO	Multidisciplinary design optimization	
FOS	Factor of safety	
FEA	Finite element analysis	
CV	Control volume	
CS	Control surface	
TUI	Text user interface	
API	Application programming interface	
GUI	Graphical user interface	
CSV	Comma separated values	
VS	Visual Studio	

1. Introduction

1.1 Motivation

Rocket stability design is a pivotal step in the development of a new rocket system. One of the simplest methods is fin stabilization. Rocket fins serve as a passive control method while not adding a large amount of mass to the total system. The effectiveness of rocket fins is directly connected to their size, since larger fins allow for more deflection of airflow. However, increasing the size of the fins can introduce large amounts of drag, negatively affecting the system. Adjusting the positioning of the rocket fins relative to the center of mass of the rocket can serve as another method of increasing effectiveness [1].

There are multiple variables that affect the performance of rocket fins, and the target performance varies based on the design goals of the rocket system. As such, it becomes necessary to simulate a wide variety of rocket fin design solutions to understand the benefits and drawbacks of each solution. To achieve a desirable balance, careful design and optimization must be completed to enhance the performance of the rocket system in its entirety. In the case of long-range guided rockets, optimization models have shown improvements in performance of up to 9% [2].

Rocket fin designs can be simulated using computational fluid dynamics (CFD) computer programs such as Ansys Fluent. The simulations can provide an understanding of the performance of each solution, but the accuracy of the results is dependent on the quality of the mesh used in the simulation. Increasing mesh quality also increases the complexity of the simulation, leading to longer computation times. The large number of solutions available in combination with the long computation times creates a problem when trying to perform design and optimization in a timely manner. By utilizing the recently developed Ansys PyAnsys Python library, Fluent simulations can benefit from automation allowing for a more efficient use of time during simulations [3]. Furthermore, automation can be implemented to assist with the selection and parameterization of designs to simulate, reducing the total number of solutions to be tested.

1.2 Literature Review

A group of students from the Warsaw University of Technology conducted many computational fluid dynamics (CFD) simulations to optimize the fin design for the Grot 1 supersonic sounding rocket. The experiment was constructed with 48 fixed fin designs and the original fin design shown in Figure 1.1, each requiring its own CFD simulation. Fin designs with a taper ratio of 0.3 tended to perform better, and the fins with a shape like Fin 21 or Fin 48 performing the best [1].

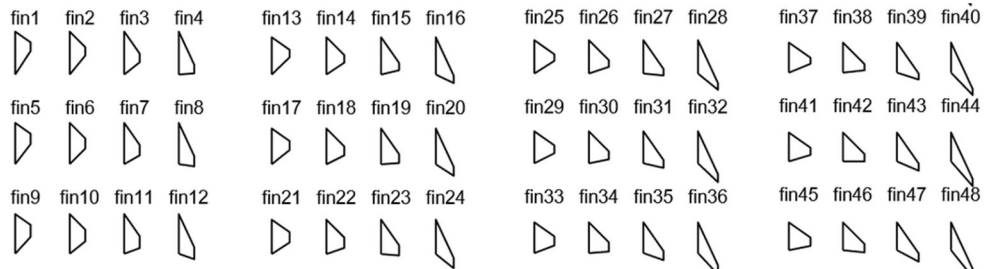


Figure 1.1 — Fin designs tested by Warsaw University of Technology students [1].

In a study conducted at the Nanjing University of Science and Technology, a set of aerodynamic shape parameters of long-range guided rockets were derived to optimize a general shape. The lift-to-drag ratio, maneuverability, and stability were prioritized as the metrics of performance, with a secondary focus

on the range and accuracy of the guided rockets. As mentioned previously, the results of the study indicated approximately 9-15% increase in performance over the general shape [2].

An optimization method involving the placement of the front fins or canards of a rocket was developed at Beihang University. The method utilized a variable step scheme and CFD to analyze the turbulence and flow field with various fin positions. The method was proven to work well when prioritizing equal weight coefficients, lateral aerodynamic force influence, or aerodynamic moment [4].

In an investigation completed by the Shijiazhuang Mechanical Engineering College, a meshing method was utilized for a CFD simulation to study the aerodynamic characteristics of canards on a rocket. The meshing method was based on the multi-block method using a structured grid, and it was proven to yield accurate results. The method was also utilized to test varying nose cones and fins with quicker convergence rates due to scoping allowed by the multi-block method [5].

In a case study presented by the Institute of Aeronautics and Space in Brazil, multidisciplinary design optimization (MDO) was applied to the fins on a sounding rocket using a developed tool called MDO-SONDA. The tool can prioritize the aerodynamic characteristics and the trajectory as shown in Figure 1.2 below [6].

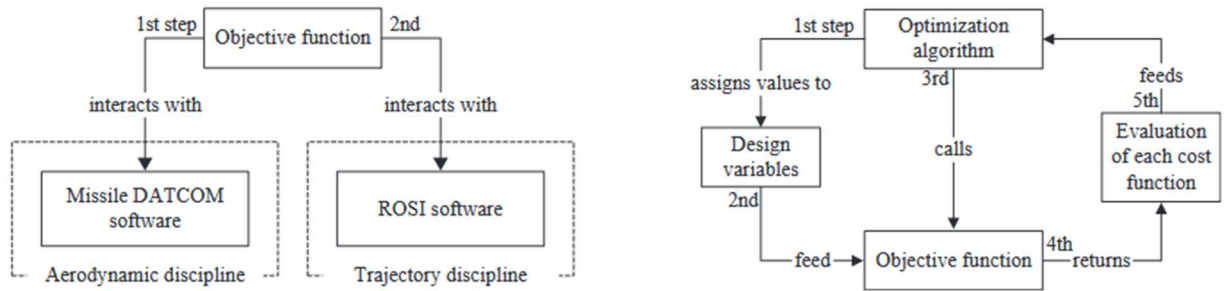


Figure 1.2 — MDO-SONDA interaction and process loop [6].

The design variables indicated in Figure 1.2 accurately represent the fin configuration tested. The case study found that the drag force caused by the rocket fins could be reduced by up to 30% without causing stability issues using the solutions produced by the tool. Figure 1.3 below demonstrates this, showing eleven solutions compared to the original fin design on the sounding rocket [6].

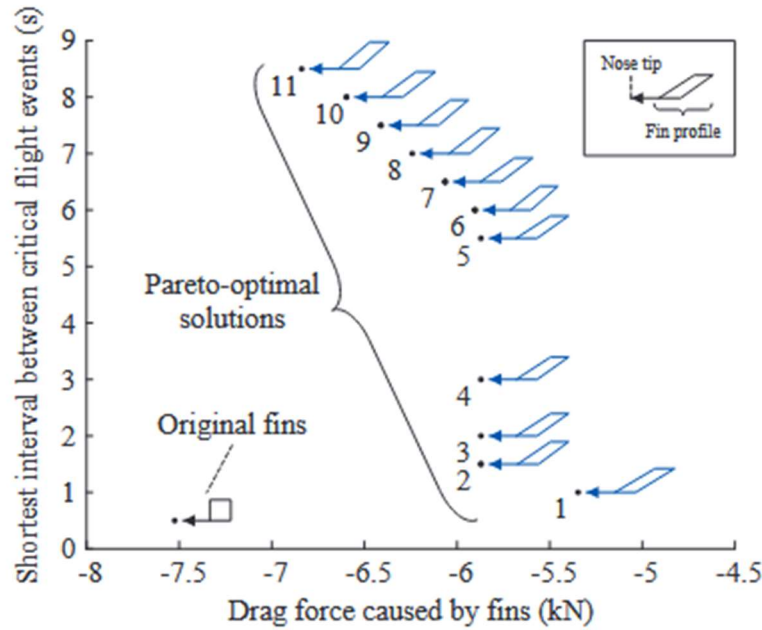


Figure 1.3 — Fin optimization results indicating drag force [6].

A research team from the Indian Institute of Technology Madras organized another MDO study on sounding rocket fins. The aerodynamic, structural, and flight performances were constrained during the optimization process while maximizing the maximum altitude. The aerodynamic static margin, payload mass, fin tip torsion angle, and factors of safety (FOS) were each considered as secondary objectives in a separate optimization problem to understand how the objectives influenced the resulting design. Conducting CFD and finite element analysis (FEA) simulations yielded expected varying results:

- Payload mass can be increased significantly with a minor loss in maximum altitude.
- Maximum altitude and aerodynamic static margin are inversely related.
- Similarly, maximum altitude and the FOS are also inversely related.
- Fin tip torsion angle increases significantly at maximum altitudes above 50 km.
- Constraining the fin tip torsion angle ensures that the FOS is maintained.

The aerodynamic static margin and fin tip torsion angle were found to be the most constraining design variables with a clear and obvious inverse relation between the payload mass and the maximum altitude [7].

The flutter induced by the design of rocket fins was studied at the Istanbul Technical University. Specifically, the thickness, taper ratio, and aspect ratio were treated as the controlling variables while observing the flutter speed. The fins of a model rocket were studied in 15 configurations using FEA and modal analysis. Figure 1.4 below shows an example of the resulting flutter mode when a fin with a thickness of 0.5 mm, taper ratio of 0.35, and aspect ratio of 0.75 was tested. The study found that the thickness of the fin had the greatest impact as a design parameter on the flutter velocity. Furthermore, the thickness had a direct relation, and the aspect ratio had an indirect relation with the flutter velocity. The taper ratio seemed to have a minor effect on the flutter velocity [8].

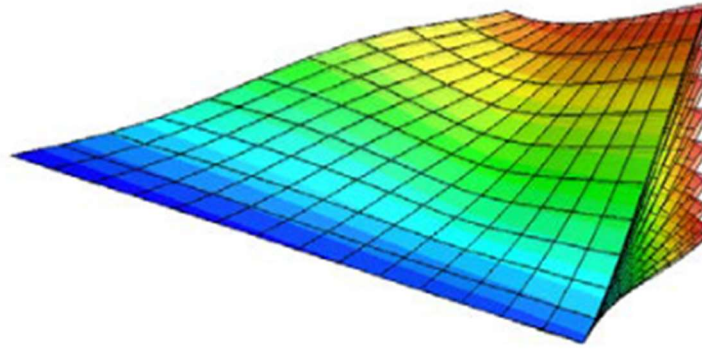


Figure 1.4 — Flutter mode example [8].

In another study conducted at the University of Turkish Aronautical Association, different rocket fin shapes were tested on a model rocket to understand their effect on the maximum altitude and overall stability of the rocket. A clipped delta fin, swept fin, trapezoidal fin, and triangular fin were tested with varying design parameters to learn which parameter was the most important for each shape. Table 1.1 below summarizes the most important design parameter for each fin type for the maximum altitude and stability. The span of the fin was the most impactful parameter for all of the fins for stability, with thickness being the most impactful for the maximum altitude for the swept fin, trapezoidal fin, and triangular fin [9].

Table 1.1 — Most impactful design parameters summarized [9].

		Most Impactful Design Parameter
Clipped Delta Fin	<i>Maximum Altitude</i>	Span
	<i>Stability</i>	Span
Swept Fin	<i>Maximum Altitude</i>	Thickness
	<i>Stability</i>	Span
Trapezoidal Fin	<i>Maximum Altitude</i>	Span/Thickness
	<i>Stability</i>	Span
Triangular Fin	<i>Maximum Altitude</i>	Span/Thickness
	<i>Stability</i>	Span

Researchers from Chosun University and the Korea Aerospace University studied the aspects of attitude control for a simple rocket with four controllable fins. Figure 1.6 below shows the modeled rocket and all the forces acting on the body and fins of the rocket. Although the goal was to develop a linear-state space model, the large effect and importance of the fins on a rocket was discussed. The viability of the fins as a control method was postulated and investigated [10].

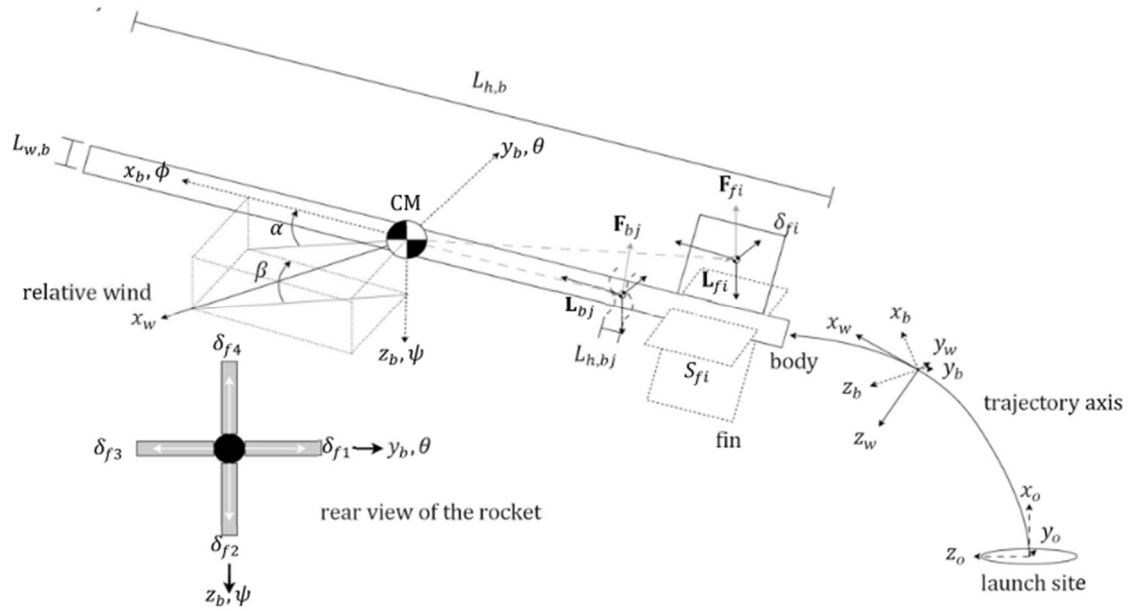


Figure 1.6 — Simple rocket free-body diagram with all forces labeled [10].

The effect of stabilizer fins and the related structure on the performance of the PG-7 unguided rocket shown in Figure 1.5 was observed by a team of researchers at the Le Quy Don Technical University. The performance was quantified by the maximum speed, deflection angle, attack angle, and spin speed of the rocket. The inclined surface area of the fin has a large effect on the spin speed of the rocket, but the force of drag increases due to the increased surface area normal to the flow. The increased drag reduces the overall maximum speed and range of the rocket. Therefore, the balance between the inclined surface area and resulting force of drag must be maintained to generate a more optimal rocket design. For the PG-7 unguided rocket in particular, an angle of tilt between 9 to 11 degrees provides the best stability during flight [11].



Figure 1.5 — PG-7 unguided rocket with stabilizer fins deployed [11].

In a study conducted at the National Research and Innovation Agency of Indonesia, the aerodynamic forces on trapezoidal and rectangular rocket fins were investigated CFD. The two fins were modeled with the same root chord length and span. They were then meshed with a structured-multiblock grid shown in Figure 1.7 below [12].

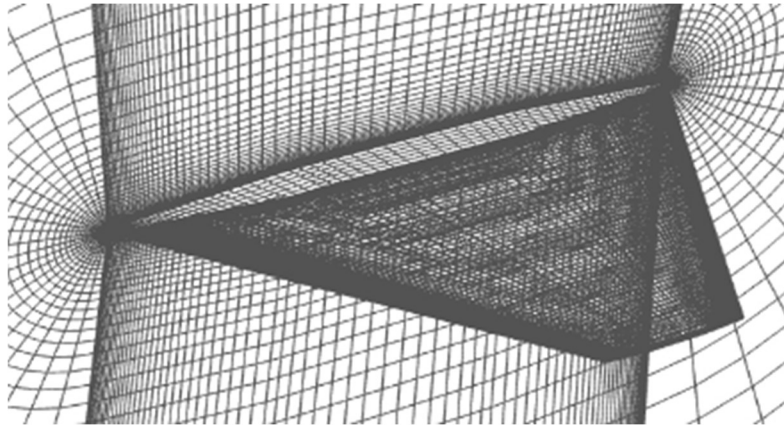


Figure 1.7 — Trapezoidal fin structured-multiblock fin [12].

It is important to note that there is a high density of points perpendicular to the leading and trailing edges of the fin at the root chord. This could be improved using a comprehensive O-grid instead of a multiblock grid made up of H-grids and C-grids. Figure 1.8 shows the resulting lift-to-drag ratio plots for varying angles of attack and Mach numbers for both fin shapes. The trapezoidal fins were found to be the better shape for most tested angles of attack, but at 15 degrees, the rectangular fin yields comparable results [12].

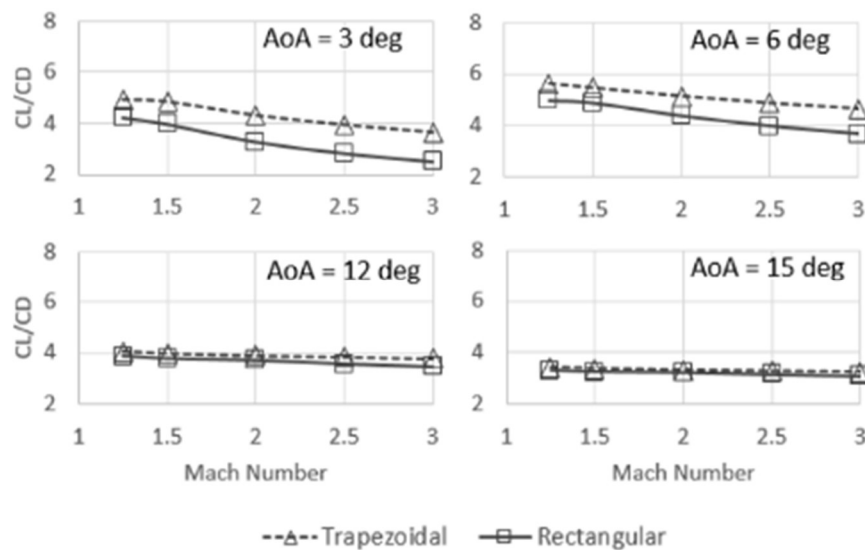


Figure 1.8 — Lift-to-drag ratio plots from CFD results [12].

Missile Aerodynamics by Jack N. Nielsen discusses the physics that affect slender-bodied rockets and how the various components influence the performance. Nielsen introduces the slender-body theory at different speeds before covering the aerodynamics of rockets. He then shows the effects of the different rocket design components including the nose cone and wings (fins and canards) on the flow surrounding

the body of the rocket. The flow following wings placed closer to the nose cone of the rocket can affect the performance of the tail of the rocket, influencing the total stability of the rocket [13].

1.3 Project Proposal

The objective of this report is to investigate the effects of rocket fin design on the entire rocket and design a method of optimization for the rocket fins using computational fluid dynamics (CFD). Then, the usefulness of automation can be determined when implemented to improve the numerical simulation run times.

1.4 Methodology

The first step is to study the effects of rocket fin designs in different use cases. This can be accomplished with the use of the Ansys Fluent CFD software. Then, the performance of a baseline design will be determined, and the variables that can be adjusted for optimal performance will be collated in order of influence. These variables will be incorporated into a simple optimization method to understand the trend towards optimal design. After performing adjustments and monitoring the change in performance of the rocket, the new Ansys PyAnsys Python library will be introduced to verify its accuracy with the previously collected data from the baseline model. With the automation model verified, it can be used to accelerate the simulation process in a more complex optimization method. The resulting optimized designs from the simple and complex methods can be compared to determine the efficacy of the automation model.

2. Understanding Computational Fluid Dynamics

Computational fluid dynamics (CFD) is a mathematical implementation of fluid mechanics with the purpose of simulating and analyzing the behavior of fluids in or around a physical object. While simulations might not be perfect, they serve as a good starting point when designing systems for specific performance parameters. CFD simulations also help reduce the cost of iterative testing with physical prototypes. Understanding how CFD works and the algorithms that are used helps ensure that the results can be interpreted to be accurate to a satisfactory degree.

2.1 Fundamental Equations

The math of computational fluid dynamics (CFD) is based on the Navier-Stokes equations in conjunction with the ideal gas law. The Navier-Stokes equations are five equations that embody the laws of conservation of:

- Mass
- Momentum
 - In the x -direction
 - In the y -direction
 - In the z -direction
- Energy

The equations are all structured similarly with the sum of the changes occurring in the control volume and the changes across the control surface on one side of the equation and the external applications on the control surface on the other side of the equation. The equation for the conservation of mass is defined in Eq. (2.1), where ρ is the density, V is the volume, $\vec{v}$ is the velocity vector, and $\vec{A}$ is the area vector.

$$\frac{d}{dt} \iiint_{CV} \rho dV + \iint_{CS} \rho \vec{v} \cdot d\vec{A} = 0 \quad (2.1)$$

The equations for the conservation of momentum in the x -, y -, and z -directions are defined in Eq. (2.2), Eq. (2.3), and Eq. (2.4), respectively. In these equations, v is the velocity, $\vec{F}_p$ is the force of pressure, $\vec{F}_\tau$ is the force of skin friction, $\vec{F}_g$ is the force of gravity, and $\vec{F}_{EM}$ is the force of electromagnetism. Each of these variables have a subscript denoting the component for the equation of each direction.

$$\frac{d}{dt} \iiint_{CV} \rho v_x dV + \iint_{CS} v_x (\rho \vec{v} \cdot d\vec{A}) = \vec{F}_{p_x} + \vec{F}_{\tau_x} + \vec{F}_{g_x} + \vec{F}_{EM_x} \quad (2.2)$$

$$\frac{d}{dt} \iiint_{CV} \rho v_y dV + \iint_{CS} v_y (\rho \vec{v} \cdot d\vec{A}) = \vec{F}_{p_y} + \vec{F}_{\tau_y} + \vec{F}_{g_y} + \vec{F}_{EM_y} \quad (2.3)$$

$$\frac{d}{dt} \iiint_{CV} \rho v_z dV + \iint_{CS} v_z (\rho \vec{v} \cdot d\vec{A}) = \vec{F}_{p_z} + \vec{F}_{\tau_z} + \vec{F}_{g_z} + \vec{F}_{EM_z} \quad (2.4)$$

The equation for the conservation of energy is defined in Eq. (2.5), where e is the energy, $\dot{Q}_{\text{ext}}$ is the change in external heat, and W_{ext} is the external work done.

$$\frac{d}{dt} \iiint_{\text{CV}} e dV + \iint_{\text{CS}} e(\rho \vec{v} \cdot d\vec{A}) = \dot{Q}_{\text{ext}} + W_{\text{ext}} \quad (2.5)$$

The ideal gas law is defined in Eq. (2.6), where p is the pressure, R is the gas constant, and T is the temperature.

$$p = \rho RT \quad (2.6)$$

2.2 Assumptions and Simplifications

The mathematics of computational fluid dynamics (CFD) can be simplified depending on the assumptions made about the system being studied. These assumptions appear in the form of a plethora of settings within CFD software as shown in Figure 2.1.

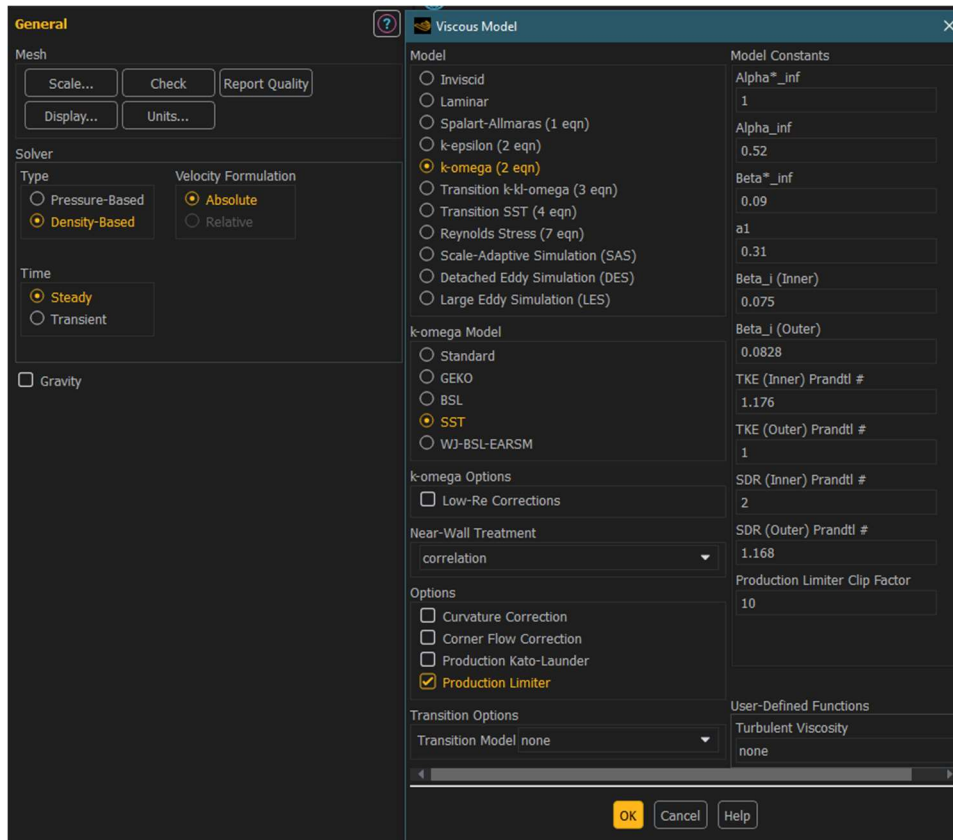


Figure 2.1 — Screen capture of Ansys Fluent solver and viscous model setup.

Understanding the effects of the assumptions helps determine the complexity of the CFD simulation. Table 2.1 below shows a few common assumptions and their effect on the mathematics of the CFD simulation.

Table 2.1 — Assumptions and their effects for the mathematics of CFD.

Assumption	Effect on the CFD Simulation
Steady state	No time dependence
2D flow	Two momentum equations
1D flow	One momentum equation
Incompressible flow	Density is constant
Inviscid flow	Viscosity is zero
Calorically perfect flow	Specific heat ratio is constant
No gravitational forces	Force of gravity is zero
No electromagnetic forces	Force of electromagnetism is zero
Constant area	Simplifies dot product
No heat addition	Change in external heat is zero
No work done	External work is zero

2.3 Mesh Generation

Mesheres are used to discretize the control volume and simplify complex geometries. Breaking curved surfaces into many flat surfaces simplifies the mathematics while ensuring the general shape of the control volume is preserved. The quality of the mesh dictates the accuracy of the computational fluid dynamics (CFD) solution up to a certain threshold. Crossing this threshold results in longer computational times and more resources consumed with a minimal improvement in solution accuracy. Therefore, performing a mesh convergence study is important to determine the mesh quality threshold. Figure 2.2 shows an example of a mesh convergence study performed using the von Mises stress [14]. In the case of CFD, the mesh convergence study can be performed using properties such as lift, drag, and moment.

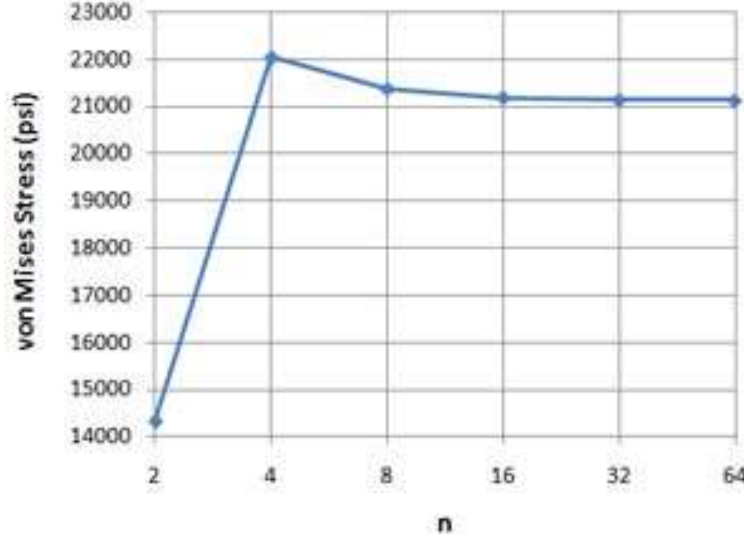


Figure 2.2 — Mesh convergence study example using von Mises stress [14].

2.4 Convergence

The computational fluid dynamics (CFD) simulation begins with an initial guess and gradually changes it until it arrives at a final solution. The iterations progress depending on the absolute error between previous iterations. The reduction of this error leads towards convergence for the CFD simulation. This continues until there is almost no noticeable change in the flow field, and the residuals are within a predetermined tolerance ($\epsilon \approx 10^{-6}$) [15].

2.5 Analytical System of Equations with Applied Assumptions

The largest assumption being made is the assumption of steady state conditions. This removes the time-dependent components in the Navier-Stokes equations listed above in Section 2.1 in Eqs. (2.1)-(2.5). The following equations show the analytical system of equations representing the fluid dynamics problem at hand in this project.

- Mass:

$$\frac{d}{dt} \iiint_{CV} \rho dV + \oint_{CS} \rho \vec{v} \cdot d\vec{A} = 0 \quad (2.7)$$

- Momentum in the x -, y -, and z -directions:

$$\oint_{CS} v_x (\rho \vec{v} \cdot d\vec{A}) = \overline{F_{p_x}} + \overline{F_{\tau_x}} + \overline{F_{g_x}} + \overline{F_{EM_x}} \quad (2.8)$$

$$\oint_{CS} v_y (\rho \vec{v} \cdot d\vec{A}) = \overline{F_{p_y}} + \overline{F_{\tau_y}} + \overline{F_{g_y}} + \overline{F_{EM_y}} \quad (2.9)$$

$$\oint_{CS} v_z (\rho \vec{v} \cdot d\vec{A}) = \overline{F_{p_z}} + \overline{F_{\tau_z}} + \overline{F_{g_z}} + \overline{F_{EM_z}} \quad (2.10)$$

- Energy:

$$\oint_{\text{CS}} e(\rho \vec{v} \cdot d\vec{A}) = \dot{Q}_{\text{ext}} + \dot{W}_{\text{ext}} \quad (2.11)$$

3. Baseline Rocket Design

The purpose of the baseline rocket design is to establish a control for the optimization of the rocket fins. It will also be used to understand which parameters are relevant for the optimization and which of them have the largest effect with the smallest adjustments.

3.1 3D Model

The 3D model is designed using Autodesk Fusion 360. The body of the rocket is designed with a length of 100 inches and a diameter of 5 inches. The nose is designed using a power series curve with a factor of 0.3 and a length of 15 inches. Figure 3.1 shows the sketch used to model the nose cone.



Figure 3.1 — Power series curve for nose cone design.

There are four fins at the base of the rocket and are initially designed with a root chord of 18 inches, tip chord of 8 inches, span of 5.5 inches, sweep angle of approximately 20 degrees, and thickness of 0.1875 inches. Figure 3.2 shows the sketch used to model each fin.

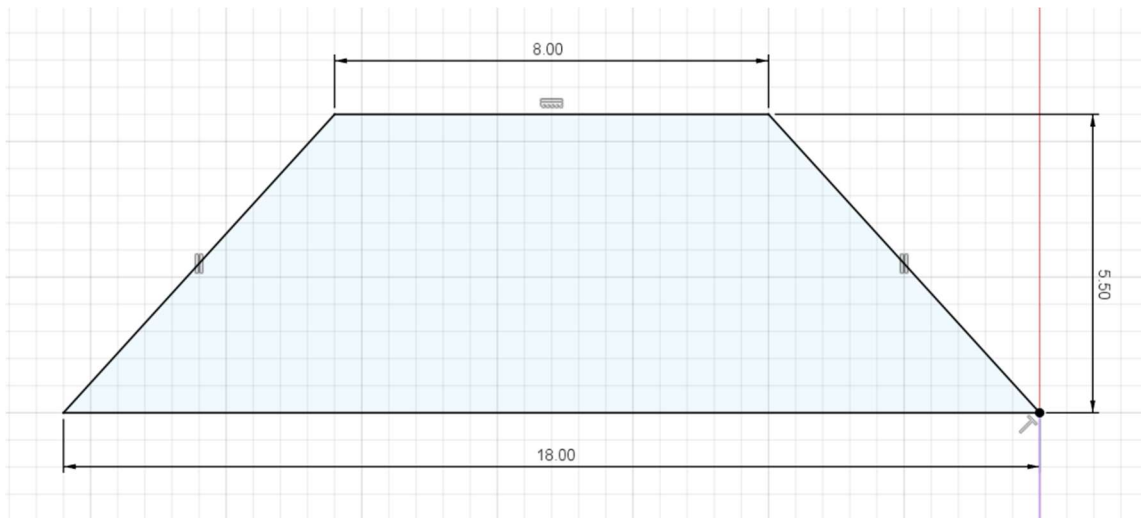


Figure 3.2 — Isometric trapezoidal fin design.

Figure 3.3 shows the complete 3D model for the baseline design.

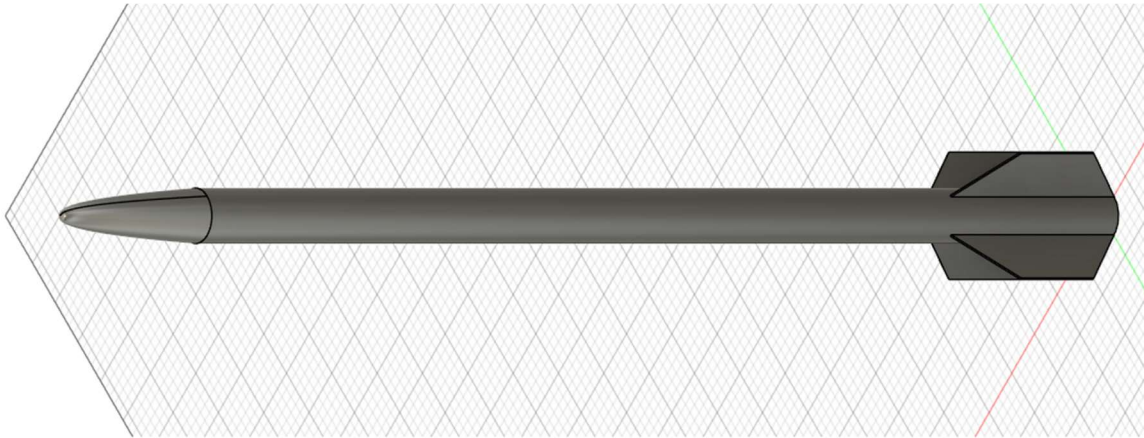


Figure 3.3 — Complete 3D model of the baseline design.

3.2 Control Volume

The control volume must be large enough to capture the propagation of the expected phenomena and how it affects the rest of the control volume. To do so, a cylindrical control volume with a radius of four times the length of the rocket is constructed. The cylinder extends two times the length from the nose of the rocket and five times the length from the end of the rocket. These domain scalars were chosen based on the work done in a study with the Grot rocket [1]. Figure 3.4 shows the control volume for the baseline design. The control volume used in the computational fluid dynamics (CFD) simulation will be a quarter of the full volume radially to reduce the size of the mesh and the number of nodes required. This can be done due to the rocket's radial symmetry when divided into four parts. The CFD simulation will fill in the rest of the volume with the assumption of periodic boundary conditions.

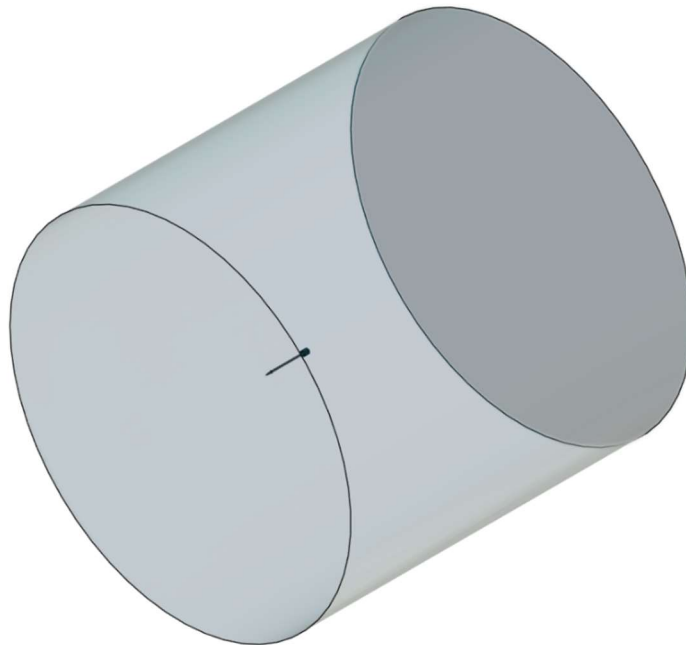


Figure 3.4 — Control volume of the baseline design.

4. Meshing Approach

4.1 Meshing in Computational Fluid Dynamics

As discussed briefly in a previous section, meshing refers to the process of discretizing the control volume into smaller elements in which the fluid properties are the same throughout. This allows for a reduction in complexity of the fundamental equations of fluid dynamics when applied in computational fluid dynamics (CFD). Each element can be treated as a subsidiary control volume in which the mass, momentum, and energy entering the element is the same when exiting the element. Additionally, fluid transfer is easier to track since fluid can only enter and exit each element through a face that it shares with an adjacent element.

Although meshing can simplify the mathematics involved in CFD, it is a form of approximation and introduces errors that propagate from element to element through the entire mesh. In general, the error is directly related to the quality of the mesh. The quality is dictated by the following properties of the mesh elements:

- The size of each element in the mesh.
- The proximity of each element to important geometry.
- The orthogonality of the elements in relation to each other.

These properties can be controlled to a certain degree in meshing software by providing rules such as specified element sizes in certain parts of the mesh.

Meshing is a complex component of CFD that has spawned a plethora of studies and careers. This reflects the importance of the meshing step in CFD and the impact it can have on the simulation results.

4.2 Meshing the Baseline Rocket Control Volume

The meshing for the baseline rocket control volume is completed using Ansys Mechanical after importing the baseline rocket control volume 3D model into the Ansys Workbench. The mesh quality is increased by applying “body sizing” rules to different aspects of the control volume 3D model. The entire mesh has a maximum element size limit of 5 inches, which is mainly applied at the outer edges of the control volume model away from the rocket body. The control dialogue for this rule is shown in Figure 4.1 below.

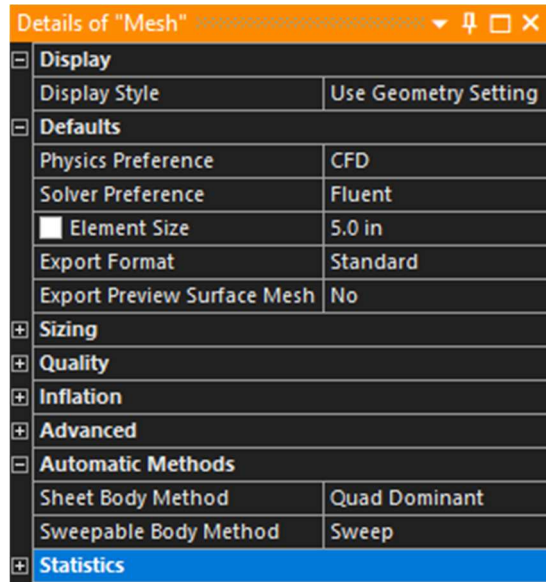


Figure 4.1 — Maximum element size specification.

The geometry of the model along the thickness of the fins became thin when the control volume was divided into four parts. Capturing the edges of the fins is essential, so another body sizing is applied specifically to the region around the fin assembly. This is accomplished through a “sphere of influence” centered at the middle of the fin assembly. The sphere is defined with a radius of 12 inches, and the elements within the sphere are specified to have a maximum element size of 0.5 inches. The control dialogue for this rule is shown in Figure 4.2 below.

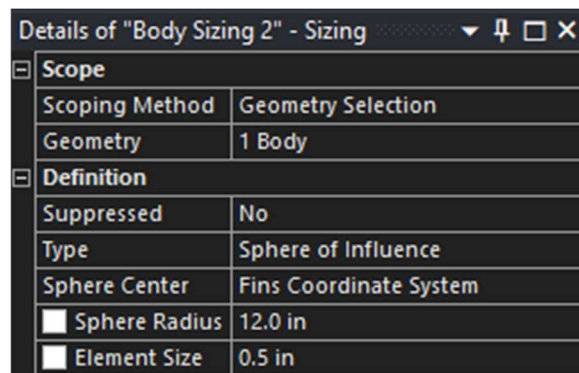


Figure 4.2 — Sphere of influence element size specification.

Finally, a “match control” rule is applied to enforce the periodic boundary conditions within the mesh itself. The rule ensures that the flat surfaces that define the quarter section of the control volume model are meshed in the same way. This allows the CFD solver to more easily connect the quarter section of the control volume model to the three other virtual quarter sections to complete the entire control volume around the rocket body. The control dialogue for this rule is shown in Figure 4.3 below.

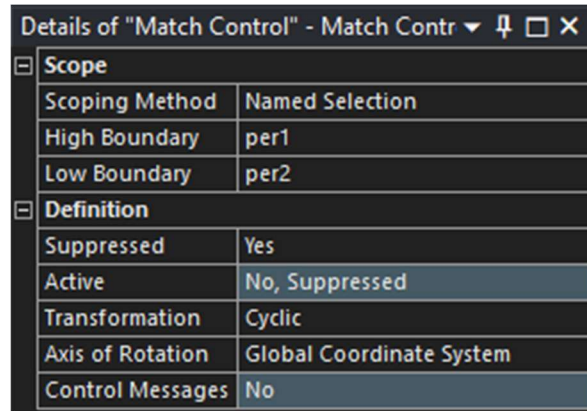


Figure 4.3 — Match control periodic boundary specification.

The resulting mesh is shown in Figure 4.4. The mesh is constructed with triangular elements in an unstructured mesh format. Although the reduction in orthogonality due to the unstructured mesh can decrease the accuracy of the CFD simulation results, it allows for easier meshing and captures the complexity of the rocket model geometry. The increased mesh quality around the fin assembly is represented by the clustering of nodes appearing to darken a section in the positive x-direction of the mesh. In addition, the mesh statistics are described in Table 4.1.

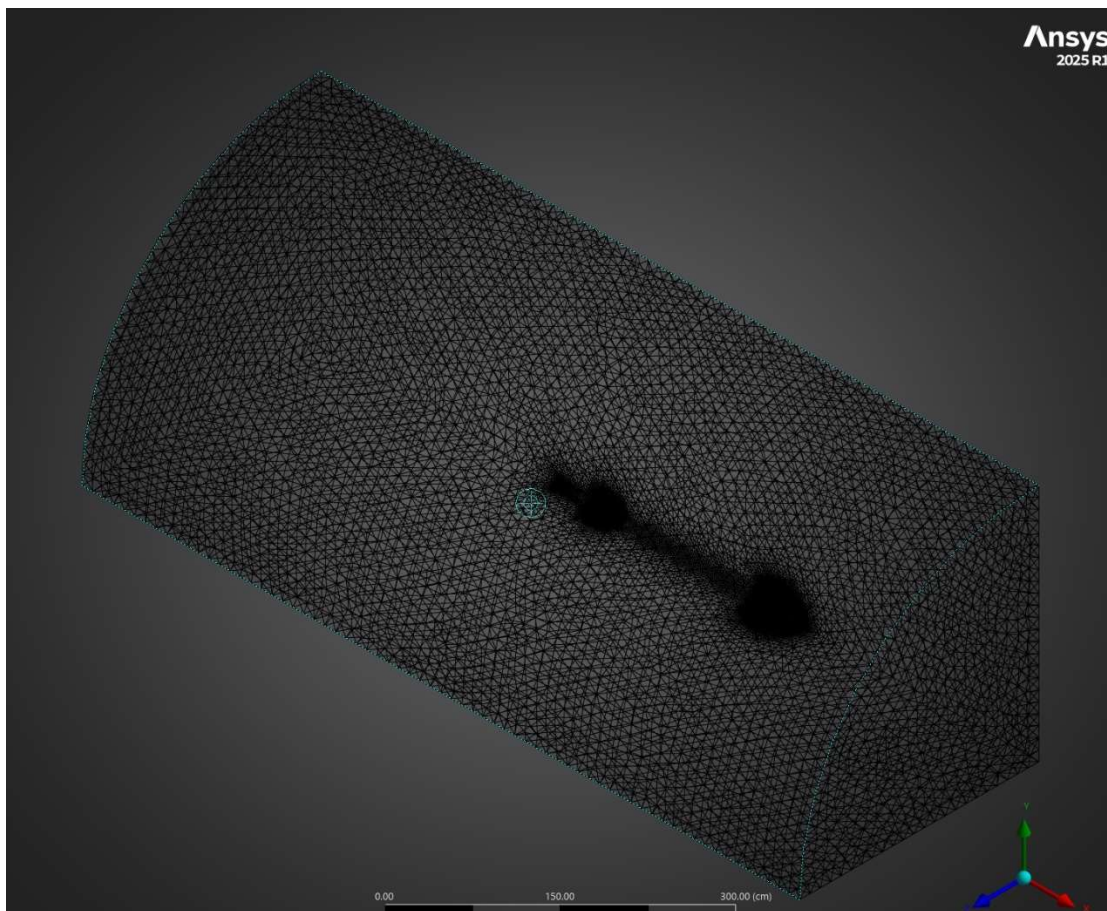


Figure 4.4 — Initial mesh of quarter section of the baseline rocket design control volume.

Table 4.1 — Baseline rocket control volume mesh statistics.

Node Count	420,425
Element Count	2,426,096
Max Aspect Ratio	47.664
Min Orthogonal Quality	0.036

5. Baseline Rocket Simulation and Results

5.1 Simulation

The computational fluid dynamics (CFD) simulation for the baseline rocket is conducted using Ansys Fluent after importing the mesh shown in the previous chapter. The flow is specified to be steady-state and turbulent, which is defined to be modeled with the $k-\varepsilon$ model. As discussed in previous chapters, the boundary conditions of the surfaces parallel with the flow direction are defined as periodic boundary conditions.

The inlet flow speed is varied at Mach 0.5, 1.0, and 1.5 assuming an altitude of 1000 meters. This allows for a study of three operating conditions of the baseline rocket:

- Subsonic ($M < 1$)
- Transonic ($M = 1$)
- Supersonic ($M > 1$)

5.2 Results

For each operating condition, the velocity and pressure contours along one of the periodic boundary surfaces are extracted using Ansys CFD Post. The velocity contour for the Mach 0.5 inlet flow speed is shown in Figure 5.1 below. The minimum recorded velocity is 0 meters per second along the outer wall, along the body of the rocket, and directly behind the rocket. This is expected along the surfaces because of the no-slip boundary condition, and the area behind directly behind the rocket indicates flow separation.

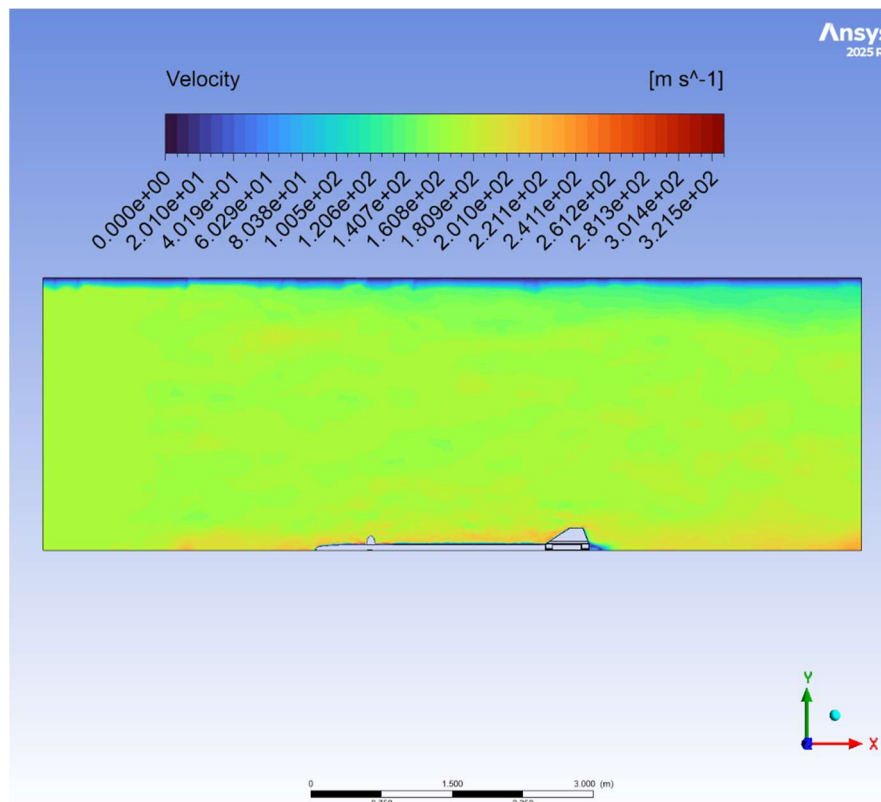


Figure 5.1 — Velocity contour for the baseline rocket at Mach 0.5.

The pressure contour for the Mach 0.5 inlet flow speed is shown in Figure 5.2 below. The maximum recorded pressure is 35.41 kilopascals at the tip of the nose cone of the rocket. This is expected due to the relatively rounded nose cone design constructed by the revolved power series curve.

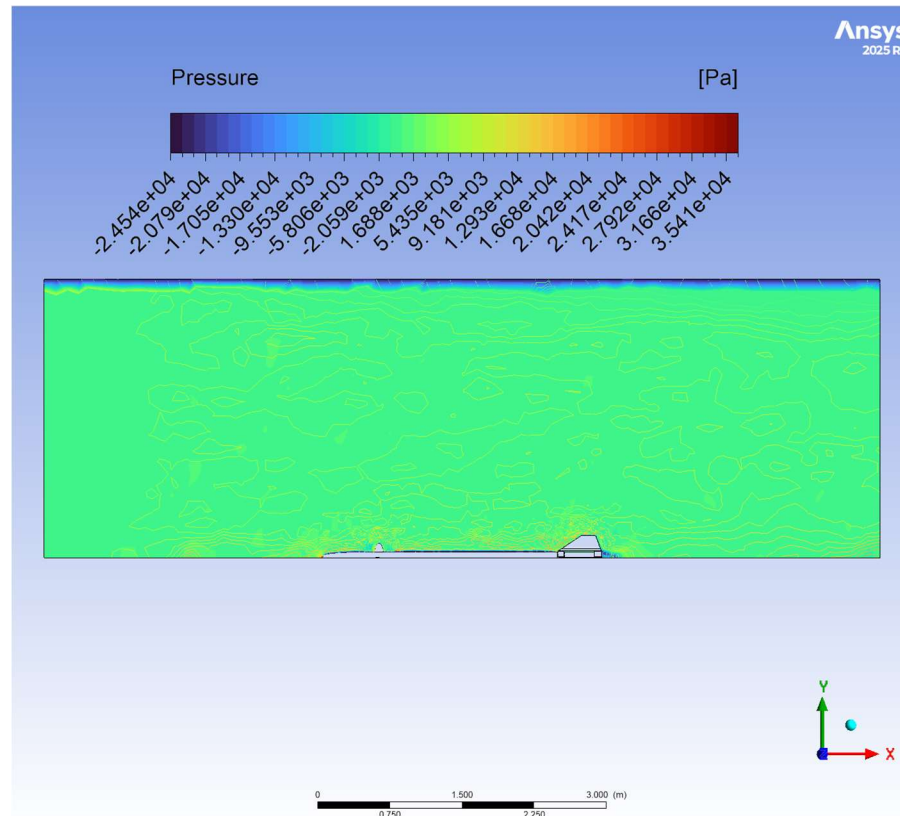


Figure 5.2 — Pressure contour for the baseline rocket at Mach 0.5.

The velocity contour for the Mach 1.0 inlet flow speed is shown in Figure 5.3 Figure 5.1below. Like the Mach 0.5 case, the minimum recorded velocity is 0 meters per second along the outer wall, along the body of the rocket, and directly behind the rocket. The flow separation appears to be slightly larger, and the relative speed of the flow is doubled.

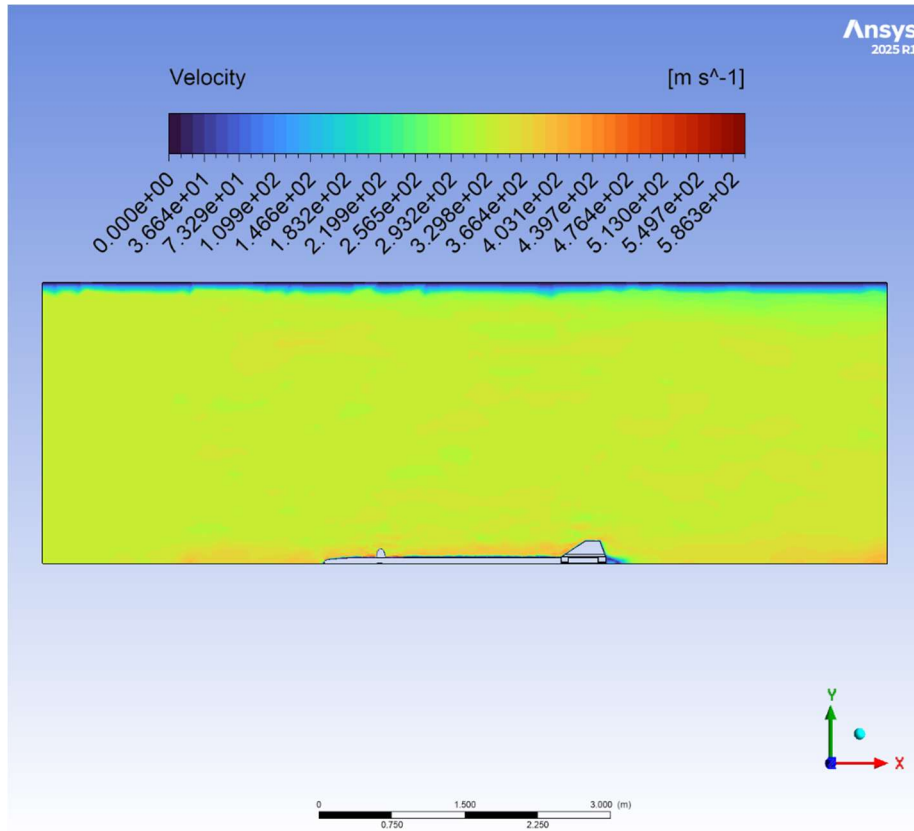


Figure 5.3 — Velocity contour for the baseline rocket at Mach 1.0.

The pressure contour for the Mach 1.0 inlet flow speed is shown in Figure 5.4 below. The maximum recorded pressure is 122.3 kilopascals at the tip of the nose cone of the rocket. This is expected for the same reason as in the Mach 0.5 case, and the increased maximum pressure is due to the increased speed.

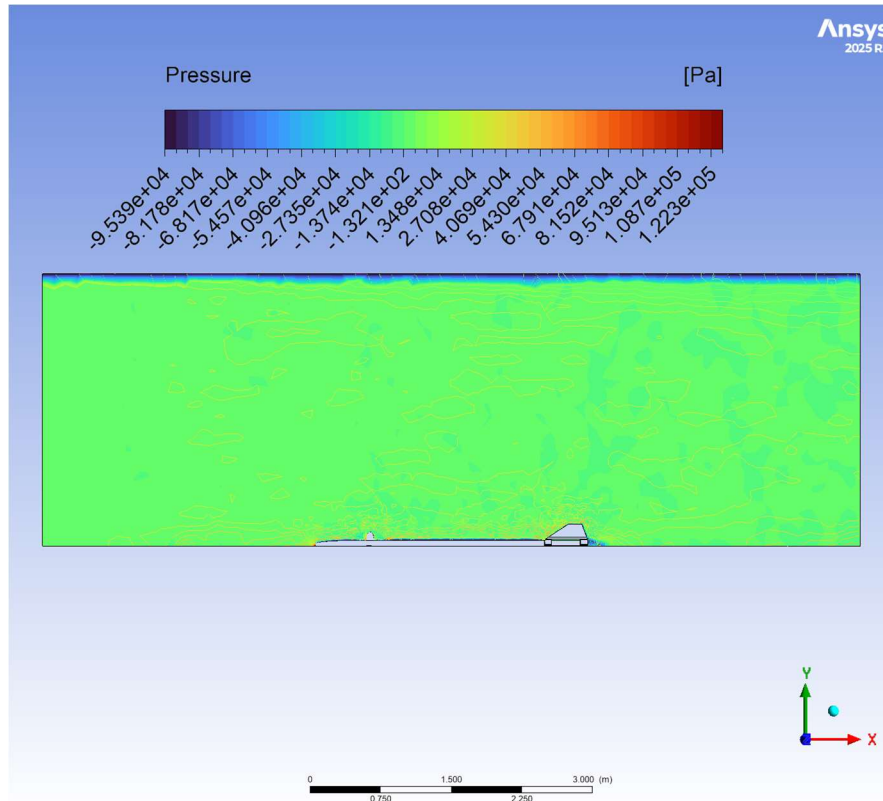


Figure 5.4 — Pressure contour for the baseline rocket at Mach 1.0.

The velocity contour for the Mach 1.5 inlet flow speed is shown in Figure 5.5 Figure 5.1below. Like the Mach 0.5 and Mach 1.0 cases, the minimum recorded velocity is 0 meters per second along the outer wall, along the body of the rocket, and directly behind the rocket. The flow separation appears to be much larger, and the relative speed of the flow is 50% more.

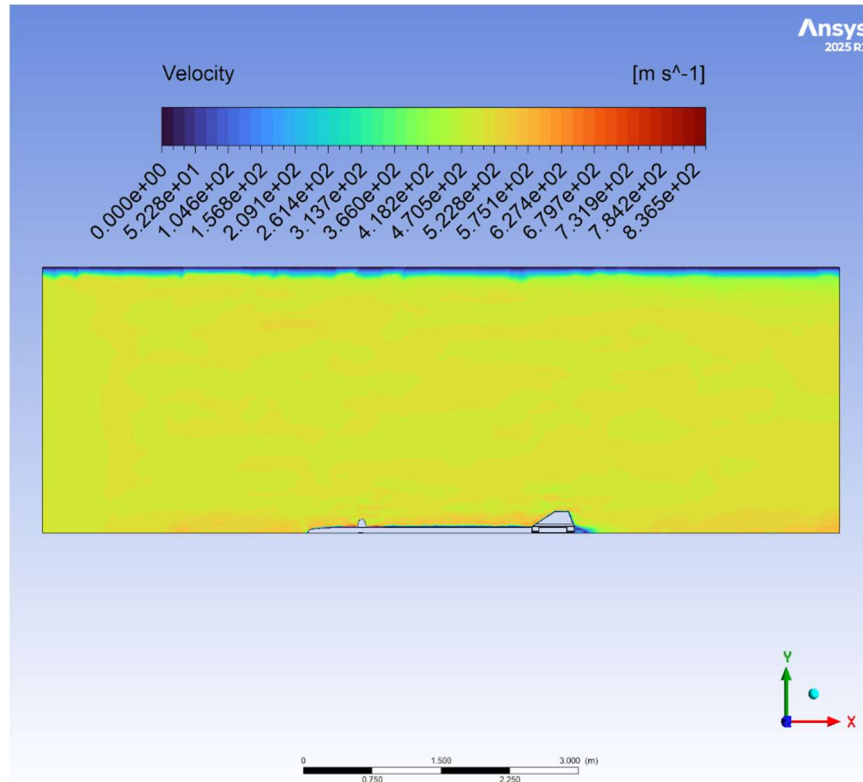


Figure 5.5 — Velocity contour for the baseline rocket at Mach 1.5.

The pressure contour for the Mach 1.5 inlet flow speed is shown in Figure 5.6 below. The maximum recorded pressure is 255.1 kilopascals at the tip of the nose cone of the rocket. This is expected for the same reason as in the Mach 0.5 case, and the increased maximum pressure is due to the increased speed.

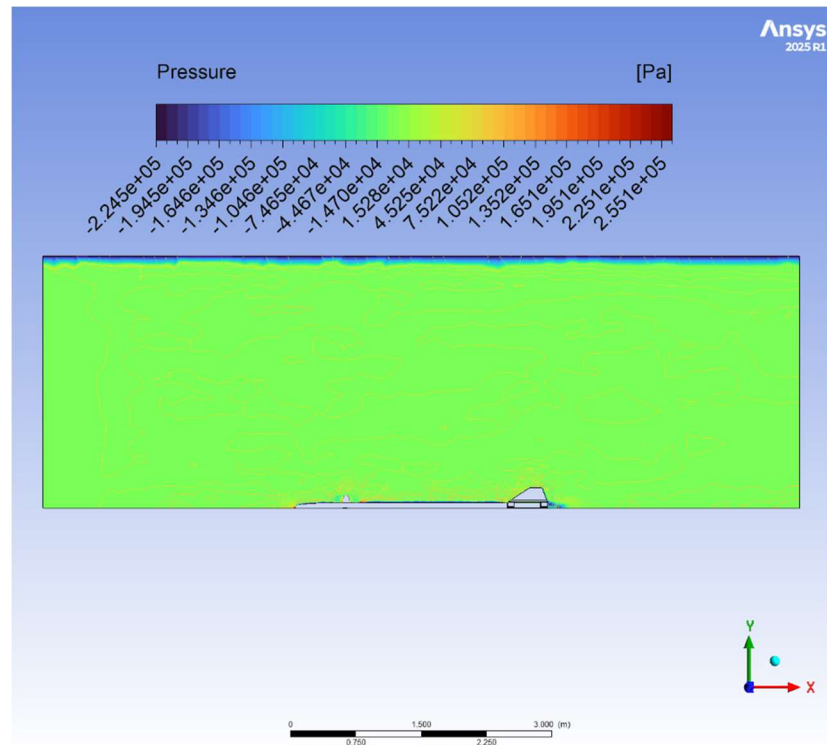


Figure 5.6 — Pressure contour for the baseline rocket at Mach 1.5.

6. CFD Software Automation

As mentioned previously, CFD simulations, especially 3D simulations, are computationally intensive and require long calculation times. These costs are compounded when the simulations require minor changes in between trials. If the changes or exact simulation parameters are known ahead of time, it is valuable to be able to automate the simulation process to reduce manual parameter adjustments and remeshing times.

6.1 Ansys PyAnsys

Ansys provides a Python library, PyAnsys, to allow users to interface with their software suite and reduce time spent manually within their software. For the purposes of this project, three packages were considered and/or utilized to simplify the simulation process:

- PyMechanical
 - Automates the meshing portion of the trial process
- PyFluent
 - Provides another way to mesh CAD models directly within Ansys Fluent
 - Automates the simulation parameter definition process
 - Conducts the CFD simulation as a whole
- PyFluent – Visualization
 - Allows for the visualization of CFD simulations
 - Provides a method of graphical user interface (GUI) integration

6.2 Meshing with PyAnsys

Meshing can be done in a plethora of ways using PyAnsys to access the multiple meshing-capable software in the Ansys software suite. For this project, PyMechanical and PyFluent were considered due to the existing familiarity with the software they interact with (Ansys Mechanical and Ansys Fluent). After a brief trade study exploring the benefits and drawbacks of PyMechanical and PyFluent, PyFluent was determined to be the optimal choice because of its compatibility with the CFD simulation aspect of Ansys Fluent.

The PyFluent package features two main documented methods of interacting with the Ansys Fluent software [3]:

- Text user interface (TUI)
 - Text-based commands that reflect how a user would interact with the built-in Ansys Fluent console
- Settings API
 - Structured and logical commands that emulate the experience of navigating through the Ansys Fluent option menus

The PyFluent package is still relatively new in its development, so documentation is sparse and difficult to utilize effectively, resulting in large amounts of time spent in trial and error to understand how the package is best used. This lack of documentation is particularly apparent when trying to use the TUI method to control the Ansys Fluent software or when building a relatively more complex test case. However, the TUI method is compatible with the “Python Journaling” setting available within the Ansys Fluent software that allows for the recording of actions performed by the user in the software as a series of TUI commands. The

recording can then be analyzed to extract the necessary TUI commands to construct the Python script to replicate the test case.

Following the TUI command logic detailed above, a Python script was formulated to perform the meshing for a test case for the rocket CFD simulation. The script begins by initiating a meshing session with double precision and two processors allocated to the task as a user would when opening the Ansys Fluent software. The CAD geometry STEP file is then imported and scaled to match the desired physical properties of the rocket body. Local mesh properties were defined to ensure an adequate representation of the finer curves and edges present along the rocket body. Using a test rocket nozzle CAD model, the Python script was able to generate a mesh with 22,500 cells and a minimum orthogonal quality of 0.34 as shown in Figure 6.1.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Create polyhedra ...

Merging Domains...
done.

-----
name      id cells (quality < 0.05)  minimum quality  cell count
-----
fluid     90                          0               0.33647483    22470
-----
name      id cells (quality < 0.05)  minimum quality  cell count
-----
Overall Summary  none                          0               0.33647483    22470
-----

Total Number of Cell Zones : 1

[Quality Measure : Orthogonal Quality]

----- 22470 cells were created in : 0.07 minutes
----- The mesh has a minimum Orthogonal Quality of: 0.34
----- The volume meshing of component2-solid is complete.

```

Figure 6.1 — PyFluent TUI method meshing terminal results.

6.3 PyFluent Core Package

The PyFluent Core Python package includes almost all the necessary classes and variables required to utilize the Settings API method to its full potential. The package is imported into a Jupyter notebook file to explore the package's capabilities in a more controlled manner. The Ansys Fluent software solver is launched using the `launch_fluent()` function. Like the meshing Python script, the function takes arguments to establish double precision and two processors, as well as an extra argument to determine the mode of the Ansys Fluent instance. The Jupyter notebook file then takes an exported mesh file in the form of a `msh.h5` file and reads it into the Ansys Fluent instance. The mesh is simultaneously checked for any meshing errors and general mesh quality. The settings dictating the type of simulation flow and model materials required are then defined using the following commands:

- `solver.setup.models.energy()` dictates whether energy is conserved in the simulation flow
- `solver.setup.models.viscous()` dictates the viscosity model used in the simulation flow
- `solver.setup.materials()` dictates the types of materials used for the body of the rocket and the simulation flow

- `solver.setup.boundary_conditions.velocity_inlet()` dictates the boundary conditions at the inlet of the simulation flow (specified as a velocity inlet)
 - The velocity property can be referenced using the `momentum.velocity` variable
 - The temperature property can be referenced using the `thermal.t` variable

Following the methodology of a regular Ansys Fluent user, the solver instance is then initialized using the hybrid initialization method unique to the Fluent software with the `solver.solution.initialization.hybrid_initialize()` function. Finally, after specifying the number of iterations desired for the simulation run time using the `solver.solution.run_calculation.iter_count` variable, the simulation is commanded to run using the `solver.solution.run_calculation.iterate()` function. Figure 6.2 shows an example of the PyAnsys PyFluent core package use case as described thus far in a basic rendition.

```
# Imports
import ansys.fluent.core as pyfluent

# Launch solver and read mesh
solver = pyfluent.launch_fluent(precision="double", processor_count=2, mode="solver")
solver.file.read_mesh(file_name="Meshes\test_nozzle.msh.h5")

# Checking settings
solver.setup.models.energy()
solver.setup.models.viscous()
solver.setup.materials()
solver.setup.boundary_conditions.velocity_inlet()

# Initializing and running solver
solver.solution.initialization.hybrid_initialize()
solver.solution.run_calculation.iter_count = 600
solver.solution.run_calculation.iterate()
```

Figure 6.2 — PyAnsys PyFluent core package Settings API method demonstration.

6.4 PyFluent Parametric Package

With single test case simulations running using the PyFluent core package, the code can be expanded to systematically process through a series of test cases with predefined parameters. The expansion to multiple test cases can be supported using the PyFluent parametric package, which utilizes the parametric studies feature built into the Ansys Fluent software. Following a similar process to the single test case Jupyter notebook file using the PyFluent core package, the parametric version begins with all the same functions and arguments. However, instead of reading in a mesh file, the parametric version reads in an established case file in the form of a `cas.h5` file using the `read_case()` function. Figure 6.3 shows an example of the PyAnsys PyFluent parametric package use case as described thus far in a basic rendition.

```

# Imports
import ansys.fluent.core as pyfluent
from ansys.fluent.parametric import ParametricProject, ParametricStudy
import pandas as pd

# Launch solver and read case file
solver = pyfluent.launch_fluent(precision="double", processor_count=2, mode="solver")
solver.file.read_case(file_name="Cases\\test_case.cas.h5")

# Setting Up Parametric Study
solver.solution.run_calculation.iter_count = 650
solver.setup.named_expressions()
solver.solution.report_definitions.surface()
solver.parameters.output_parameters.report_definitions()

# Initializing Parametric Study
study_1 = ParametricStudy(solver.parametric_studies)

# Adding Design Points and Running Solver
design_point_1 = study_1.add_design_point()
study_1.update_all_design_points()

# Exporting Design Point Table and Reading Data
study_1.export_design_table("design_points_study1.csv")
design_point_dataframe = pd.read_csv("design_points_study1.csv")

```

Figure 6.3 — PyAnsys PyFluent parametric package Settings API method demonstration.

This endeavor increases in complexity with the use of different CAD geometry models and meshes because the case file would need to be changed between the use of each new mesh.

6.5 Data Handling and Display

With the predefined set of output variables collected from the sequence of parametric test cases, the Jupyter notebook file can extract the design point table as a comma-separated values (CSV) file using the `export_design_table()` function. Using other standard data processing Python libraries such as NumPy, Pandas, and Matplotlib, the output data can be analyzed, manipulated, and displayed in various forms. The PyAnsys PyFluent visualization package can also be used to display graphics and plot data like how they would appear within the Ansys Fluent software.

6.6 Drawbacks and Limitations

Although the PyAnsys library appears to be comprehensive and fully capable of performing the task set forth by this project, there are a few drawbacks and limitations that hold it back from solving the optimization problem proposed above. These issues can be summarized as follows:

- Meshing is still an extremely complex problem with many important and impactful components that can influence the final outcome significantly. This is especially true when working with 3D objects and control volumes.
- The periodic boundary conditions feature in Ansys Fluent in combination with the PyAnsys PyFluent package implementation produce interesting results, resulting in varying degrees of success when utilized. Again, this is especially true with 3D objects and control volumes.
- The PyAnsys library packages are still very early in development, with version numbers still in the 0.x domain (not a full-release 1.x version). This indicates that the packages are still missing complete functionality.
- Documentation is also incomplete, and most of the parts that are present are updated regularly, making it a challenge to stay up to date with updates to the codebase.

7. Python Executable Usage and Demonstration

The Python executable developed to perform the parametric Ansys Fluent computational fluid dynamics (CFD) tests is a vital component of the project. It reduces the overall time spent setting up mesh models, opening and closing Ansys software in the Ansys Workbench tool, waiting for results from the CFD simulations, and processing multiple sets of results into presentable data.

7.1 Requirements

Running the Python executable requires a set of software and installations to be performed and ready for use. The required software (some are recommended, as they improve the user experience) are as follows:

- The Python language itself (version 3.10+)
- Microsoft Visual Studio Code and the corresponding extensions (recommended, but not needed if experienced with Python and Jupyter notebooks)
 - Python extension
 - Helps visualize components of the Python code
 - Jupyter notebook extension
 - Allows notebook hosting within the VS Code IDE and simplifies the user experience
- Ansys software suite (version 2024+)
- PyAnsys

After installing the Python language, the necessary Python packages must be installed, as well. These packages are as follows:

- PyAnsys packages
 - PyFluent Core (`ansys-fluent-core`)
 - PyFluent Parametric (`ansys-fluent-parametric`)
 - PyFluent Visualization (`ansys-fluent-visualization`)
- NumPy (`numpy`)
- Pandas (`pandas`)
- Matplotlib (`matplotlib`)
- Jupyter Notebook (`jupyter-notebook`)
- Other dependencies that should be automatically installed when installing the above packages

The Python packages must be installed using the `pip` command in the command prompt.

7.2 Operation

Before running the Python executable in the command prompt, there are input files that are required to be named in a particular way to be accessible to the Python executable. These files must be placed in the same directory as the Python executable and are detailed below:

- Meshes folder
 - Meshes to be used in the simulation following the naming scheme –
`test_geometry#.msh.h5`
- Cases folder

- Cases to be used in the parametric simulation setup following the naming scheme –
`test_case#.cas.h5`

With all the required software and Python packages installed and files in the correct directories, the Python executable can be called from the command prompt with the following command:

- `python ansys_autosim.py`

After running the Python executable without error, it will create a folder called `Results` in which the data from the simulations can be found.

8. Optimization Trials

8.1 Fin Geometry Variations

The Python executable file was tested with the rocket fin optimization problem using multiple fin designs. Four fin designs were chosen to reduce complexity, as shown in Figure 8.1. These fins were derived from the test completed by the team at the University of Warsaw [1].

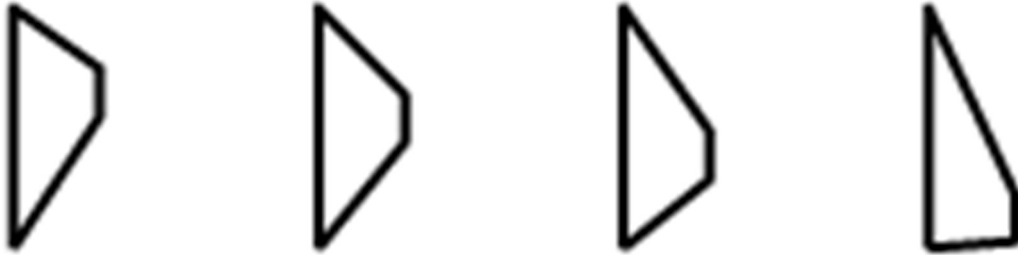


Figure 8.1 — Four fin designs tested using the Python executable.

Table 8.1 dictates the design parameters of the four fins.

Table 8.1 — Fin design parameters.

Fin Number	Leading Edge Sweep Angle [deg]	Taper Ratio	Aspect Ratio
1	35	0.2	0.6
2	45	0.2	0.6
3	55	0.2	0.6
4	65	0.2	0.6

The fins were modeled using Autodesk Fusion 360, like the baseline rocket design from Section 3.1.

8.2 Simulation Performance and Optimization Results

Running the simulations individually and manually would have taken a large amount of time as evidenced by the baseline simulation attempt from Chapter 5. With the use of the Python executable, although the simulation time might have been the same, the amount of time required by the user was significantly reduced because the only necessary action from the user was the initiation of the Python executable. The primary data points extracted by the Python executable were the drag and lift coefficients for each fin at each Mach number as shown in Table 8.2, Table 8.3, and Table 8.4 below.

Table 8.2 — CFD results for each fin tested at Mach 0.5

Fin Number	Drag Coefficient	Lift Coefficient
1	0.3101	0.1191
2	0.311	0.1346
3	0.3113	0.1327
4	0.3132	0.1321

Table 8.3 — CFD results for each fin tested at Mach 1

Fin Number	Drag Coefficient	Lift Coefficient
1	0.3898	0.1601
2	0.3821	0.1665
3	0.3902	0.1734
4	0.4289	0.1673

Table 8.4 — CFD results for each fin tested at Mach 3

Fin Number	Drag Coefficient	Lift Coefficient
1	0.2756	0.1302
2	0.2739	0.1323
3	0.2687	0.1298
4	0.2601	0.1254

8.3 Analysis

The tests performed by the Python executable yielded results that are relatively accurate to the expected values for the drag and lift coefficients. When compared to the results provided by the team at the University of Warsaw, the drag coefficients tended to be higher with an error range of 4-7% for the tests performed at Mach 0.5 and Mach 1. However, the tests performed at Mach 3 yielded larger errors (tending to the higher side) of approximately 9-11%. This is likely due to the lack of accountability of the supersonic flow when using the PyAnsys PyFluent library. The command used to set the velocity of the flow does not apply the same changes as when done manually in the Ansys Fluent software. The lift coefficient results from the Python executable tests were similarly close in value with an error range of 6-9%. This was consistent across all three flow velocities [1].

With respect to the simulation performance, the results are more complicated. From start to finish, the time required to complete an individual simulation using the Python executable is longer by 5-10 minutes. This is most likely caused by the increased overhead performance cost added by the use of the PyAnsys packages as they are still early in development. Instead, if the series of simulations are the focus of the measurement of performance, the Python executable is able to complete four simulations an estimated 20 minutes faster than running them individually and manually. The lack of needing to close and open the programs continuously drastically reduced the total time required. Furthermore, the Python executable is able to easily combine the results for comparison automatically, without requiring the user to manually extract each simulation's data and compile it in a presentable manner.

9. Conclusion

The project attempted to decrease the time and effort required by a user attempting to conduct a series of computational fluid dynamics simulations for optimization using a Python executable software. It was found that the use of parametric simulations is a relatively accurate method for optimizing rocket fin geometry. The simulations yielded drag and lift coefficient results that aligned closely with existing data with errors in the range of 4-9% on average. The drag coefficient results at supersonic flow speeds yielded larger errors, likely due to the infancy of the PyAnsys PyFluent libraries.

It was also found that the Python executable is ineffective at reducing runtime when used with a single simulation. However, multiple simulations run consecutively are completed more quickly using the Python executable. This is due to the efficacy of the balance between the overhead workload on the system introduced by the Python executable and the time taken to open, initiate, and close the CFD software. Future steps include the incorporation of the new SimAI artificial intelligence tool introduced by Ansys to further decrease simulation time and aid in the modeling process, as well. With further development of the PyAnsys PyFluent libraries and the incorporation of the SimAI tool, CFD optimization problems can be solved at a faster rate with high levels of accuracy [16]. These next steps will speed up the optimization while introducing potential improvements that the user may not have considered to increase the quality of the final result.

References

- [1] Żurawka, P., Sahbon, N., Pytlak, D., Sochacki, M., Puchalski, A., and Murpani, S., “Multi-Objective Optimization of a Fin Shape for a Passive Supersonic Rocket Stage,” Proceedings, 2023 IEEE Aerospace Conference, Yellowstone Conference Center, Big Sky, MT, United States, 4-11 March 2023. <https://doi.org/10.1109/AERO55745.2023.10115859>
- [2] Cao, R., and Zhang, X., “An Aerodynamic Shape Optimum Study for Long-Range Guided Rockets to Maximize the Range and Increase the Hit Accuracy,” Proceedings, 30th International Symposium on Ballistics, Long Beach, CA, United States, 11-15 September 2017, <https://doi.org/10.12783/ballistics2017/16816>
- [3] “PyFluent Documentation 0.35.0 — PyFluent,” PyAnsys, URL: < <https://fluent.docs.pyansys.com/> >, retrieved 3 October 2025.
- [4] Zhou, Z., and Huang, J., “Position Calculation for Front Fin of Rocket Forebody Using Variable Step Scheme,” *Aerospace*, Vol. 11, No. 8, 2024, p. 617. <https://doi.org/10.3390/aerospace11080617>
- [5] Chen, Y.-C., Gao, X.-B., Gao, M., and Lv, H.-M., “Aerodynamic Characteristic of a Canard Guided Rocket,” *International Journal of Modeling, Simulation, and Scientific Computing*, Vol. 8, No. 1, 2017. <https://doi.org/10.1142/S1793962317500015>
- [6] Barbosa, A. N., and Guimarães, L. N. F., “Multidisciplinary Design Optimization of Sounding Rocket Fins Shape Using a Tool Called MDO-SONDA,” *Journal of Aerospace Technology and Management*, Vol. 4, No. 4, 2012, pp. 431-442. <https://doi.org/10.5028/Jatm.2012.04044412>
- [7] Priyadarshi, P., Alam, M., and Saroha, K., “Multi-Disciplinary Multi-Objective Design Optimization of Sounding Rocket Fins,” *International Journal of Advances in Engineering Sciences and Applied Mathematics*, Vol. 6, No. 3, 2014, pp. 166–182. <https://doi.org/10.1007/s12572-015-0121-6>
- [8] Tola, C., and Nikbay, M., “Investigation of the Effect of Thickness, Taper Ratio and Aspect Ratio on Fin Flutter Velocity of a Model Rocket Using Response Surface Method,” Proceedings, 7th International Conference on Recent Advances in Space Technologies (RAST), Istanbul, Turkey, 16-19 June 2015. <https://doi.org/10.1109/RAST.2015.7208310>
- [9] Pektaş, A., Hacıabdullahoğlu, Ü., Ejder, N., Demircan, Z., and Tola, C., “Effects of Different Fin Shapes on Apogee and Stability of Model Rockets,” Proceedings, 9th International Conference on Recent Advances in Space Technologies (RAST), Istanbul, Turkey, 11-14 June 2019. <https://doi.org/10.1109/RAST.2019.8767439>
- [10] Kim, S., Tullu, A., and Jung, S., “Linearized State-Space Model-Based Attitude Control for Rocket With Four Controllable Fins—Part 1-1: Basic Modeling and Identification,” *IEEE Access*, Vol. 11, 2023, pp. 146014–146029. <https://doi.org/10.1109/ACCESS.2023.3338445>
- [11] Nguyen, D. P., Nguyen, M. P., Phan, B. T., Vo, V. B., and Nguyen, T. H., “Study on the Influence of Some Structural Features of the Stabilizer Fins on the Stability of Unguided Rockets,” Proceedings, 2023 International Conference on Military Technologies (ICMT), Congress Centre, Brno, Czech Republic, 23-26 May 2023. <https://doi.org/10.1109/ICMT58149.2023.10171280>
- [12] Rohman, T., and Riyadl, A., “Aerodynamic Study of Trapezoidal and Rectangular Fins Using Semi Empirical and Numerical Methods,” presented at the 2024 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES), 2024. <https://doi.org/10.1109/ICARES64249.2024.10767919>
- [13] Nielsen, J.N., *Missile Aerodynamics*, McGraw-Hill Series in Missile and Space Technology, 1960
- [14] “How to Perform a Mesh Convergence Study,” Autodesk, URL: < <https://www.autodesk.com/support/technical/article/caas/sfdcarticles/sfdcarticles/How-to-Perform-a-Mesh-Convergence-Study.html> >, retrieved March 28, 2025.
- [15] “What Is CFD | What Is Computational Fluid Dynamics?” SimScale, URL: < <https://www.simscale.com/docs/simwiki/cfd-computational-fluid-dynamics/what-is-cfd-computational-fluid-dynamics/> >, retrieved 28 March 2025.

- [16] El Hout, M., and Procario, J., “From Chips to Ships: Optimize Design With the Ansys SimAI Platform,” Ansys, May 29 2024, URL: < <https://www.ansys.com/blog/chips-ships-optimize-design-ansys-simai-platform> >, retrieved Feb. 16, 2025.

Appendix

Appendix A — Python Executable

```
# Imports necessary for the EXE
### General Purpose:
import numpy as np
import pandas as pd

### Ansys PyAnsys:
import ansys.fluent.core as pyfluent
from ansys.fluent.parametric import ParametricProject
from ansys.fluent.parametric import ParametricStudy

# EXE begins by opening Ansys Fluent. The "meshing" keyword
# opens the software in meshing mode. The "show_gui" variable
# controls whether or not the EXE opens the visual interface
# of the software.
fluent_meshing = pyfluent.launch_fluent(precision="double", mode="meshing",
                                         processor_count=2, show_gui=False)

# Create and define the workflow instance to import geometry
WF = fluent_meshing.workflow
WF.InitializeWorkFlow(WorkflowType="Watertight Geometry")
workflow_dict1 = {"FileName" : "rocket_geometry.pmdb",
                  "ImportCadPreferences" : {"MaxFacetLength" : 0},
                  "LengthUnit" : "in"}
WF.TaskObject["Import Geometry"].Arguments.set_state(workflow_dict1)
WF.TaskObject["Import Geometry"].Execute()

# Generate surface mesh
workflow_dict2 = {"AddChild" : "yes",
                  "BOICellsPerGap" : 1,
                  "BOIControlName" : "curvature_1",
                  "BOICurvatureNormalAngle" : 18,
                  "BOIExecution" : "Curvature",
                  "BOIFaceLabellList" : ["wall"],
                  "BOIGrowthRate" : 1.2,
```

```

        "BOIZoneorLabel" : "label"}
WF.TaskObject['Add Local Sizing'].Arguments.set_state(workflow_dict2)
WF.TaskObject['Add Local Sizing'].AddChildAndUpdate(DeferUpdate=False)
WF.TaskObject['Generate the Surface Mesh'].Execute()

# Describe and update/customize mesh
WF.TaskObject['Describe Geometry'].UpdateChildTasks(SetupTypeChanged=False)
workflow_str1 = "The geometry consists of only fluid regions with no voids"
workflow_dict3 = {"NonConformal" : "No",
                  "SetupType" : workflow_str1}
WF.TaskObject['Describe Geometry'].Arguments.set_state(workflow_dict3)
WF.TaskObject['Describe Geometry'].UpdateChildTasks(SetupTypeChanged=True)
WF.TaskObject['Describe Geometry'].Execute()
WF.TaskObject['Update Boundaries'].Execute()
WF.TaskObject['Update Regions'].Execute()
workflow_dict4 = {"LocalPrismPreferences" : {"Continuous" : "Continuous"}}
WF.TaskObject['Add Boundary Layers'].Arguments.set_state(workflow_dict4)
WF.TaskObject['Add Boundary Layers'].AddChildAndUpdate(DeferUpdate=False)

# Generate volume mesh
workflow_dict5 = {"VolumeFill" : "polyhedra"}
WF.TaskObject['Generate the Volume Mesh'].Arguments.set_state(workflow_dict5)
WF.TaskObject['Generate the Volume Mesh'].Execute()

# After completing the mesh, the meshing instance needs to be
# switched into a solver instance to proceed towards a CFD solution.
fluent_solver = fluent_meshing.switch_to_solver()

# Define solver parameters
fluent_solver.setup.models.energy.enabled = True
fluent_solver.setup.models.viscous.model = "k-omega"
fluent_solver.setup.cell_zone_conditions.fluid.material = "air"
fluent_solver.setup.boundary_conditions.velocity_inlet["inlet"].velocity = 160.8

# Initiate and run the solver using the defined parameters
fluent_solver.solution.initialize.hybrid_initialize()
fluent_solver.solution.run_calculation.iter_count = 600
fluent_solver.solution.run_calculation.iterate()

```

```

# Set up parametric study
fluent_solver.solution.run_calculation.iter_count = 650
parasolver_dict1 = {"input_parameter" : True,
                    "output_parameter" : False,
                    "unit" : "velocity",
                    "parametername" : "inlet_velocity",
                    "parameterid" : "real-1",
                    "description" : "",
                    "definition" : "160 [m/s]",
                    "name" : "inlet_velocity"}

fluent_solver.setup.named_expressions["inlet_velocity"] = parasolver_dict1
parasolver_dict2 = {"report_type" : "surface-areaavg",
                    "field" : "velocity-magnitude",
                    "surface_names" : ["plane-2"]}

fluent_solver.solution.report_definitions.surface["outlet-ve"] = parasolver_dict2
parasolver_dict3 = {"report_type" : "surface-areaavg",
                    "field" : "pressure",
                    "surface_names" : ["plane-2"]}

fluent_solver.solution.report_definitions.surface["outlet-pe"] = parasolver_dict3

# Initialize the parametric study and add design points with
# varying inlet velocities to simulate different flight velocities.
para_study = ParametricStudy(fluent_solver.parametric_studies)

para_dp_1 = para_study.add_design_point()
para_dp_1_input_params = para_study.design_points["DP_1"].input_parameters
para_dp_1_input_params["inlet_velocity"] = 160.8
para_study.design_points["DP_1"].input_parameters = para_dp_1_input_params

para_dp_2 = para_study.add_design_point()
para_dp_2_input_params = para_study.design_points["DP_2"].input_parameters
para_dp_2_input_params["inlet_velocity"] = 321.6
para_study.design_points["DP_2"].input_parameters = para_dp_2_input_params

para_dp_3 = para_study.add_design_point()
para_dp_3_input_params = para_study.design_points["DP_3"].input_parameters
para_dp_3_input_params["inlet_velocity"] = 482.4
para_study.design_points["DP_3"].input_parameters = para_dp_3_input_params

```

```

para_study.update_all_design_points()

# Export design points into a table using NumPy and Pandas
# to present resulting data.
para_study.export_design_table("DP_para_study.csv")
para_study_dataframe = pd.read_csv("DP_para_study.csv")

columns_dict = {"inlet_velocity" : "Inlet Velocity [m/s]",
                "outlet-ve" : "Outlet Velocity [m/s]",
                "outlet-pe" : "Outlet Pressure [Pa]"}
para_study_dataframe.rename(columns_dict, inplace=True)
para_study_dataframe.drop(0, inplace=True)
para_study_dataframe.drop(columns=["Unnamed; 13"], inplace=True)
para_study_dataframe.reset_index(drop=True, inplace=True)

```