Concorde simulation using Fenicsx

Centralesupelec
Université Paris-Saclay

29/3/2024

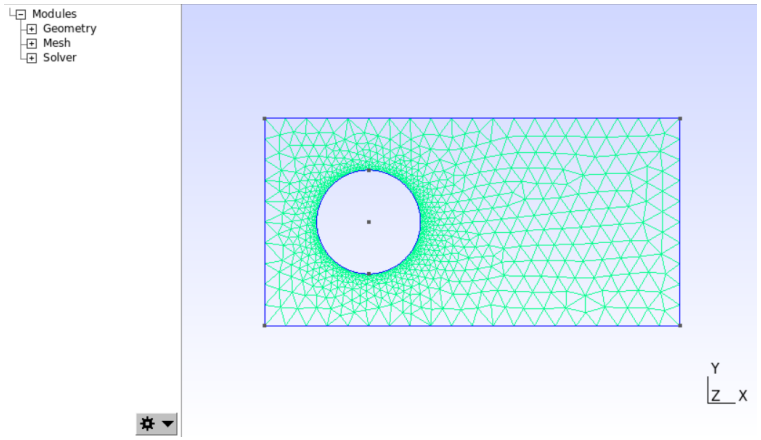
Author

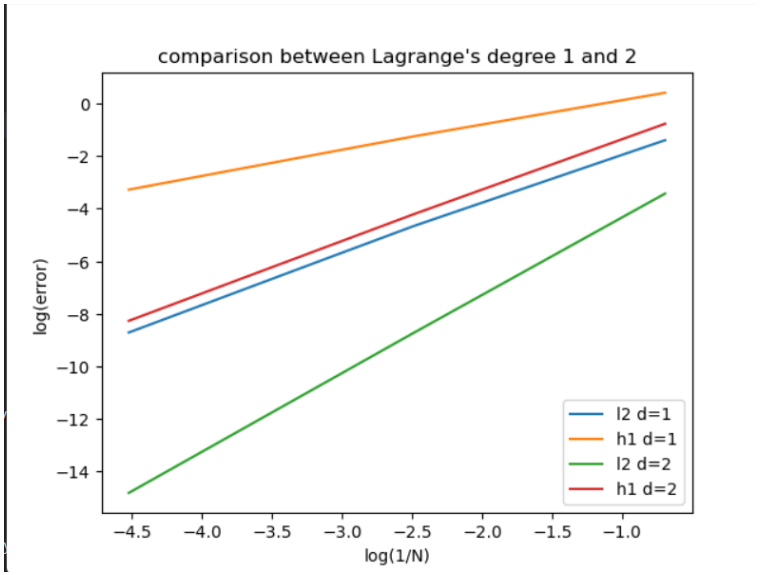
Guilherme Mertens de Andrade
Bowen Zhu
Giorgia Lanciotti
Arthur Thomas

Indices

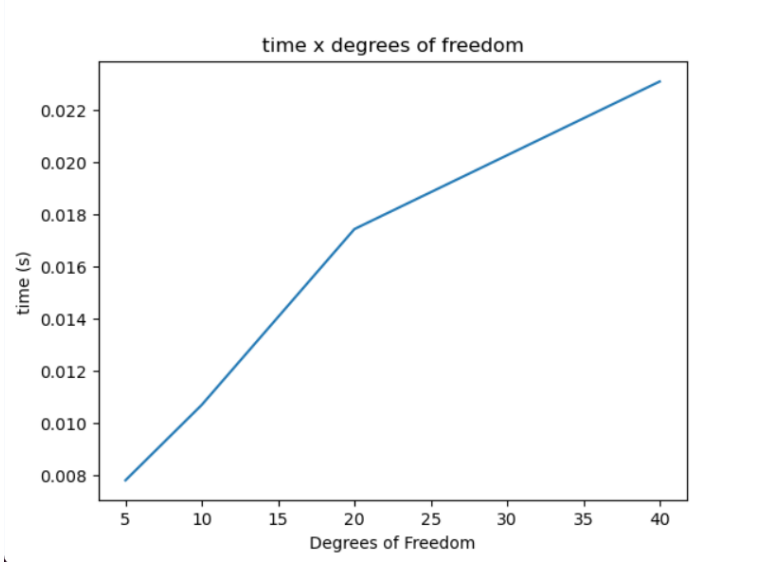
Overview of worksheet

Worksheet 1 : Results






```
L2-error: 5.38e-03
H01-error: 2.18e-01
```



Worksheet 2 : Results

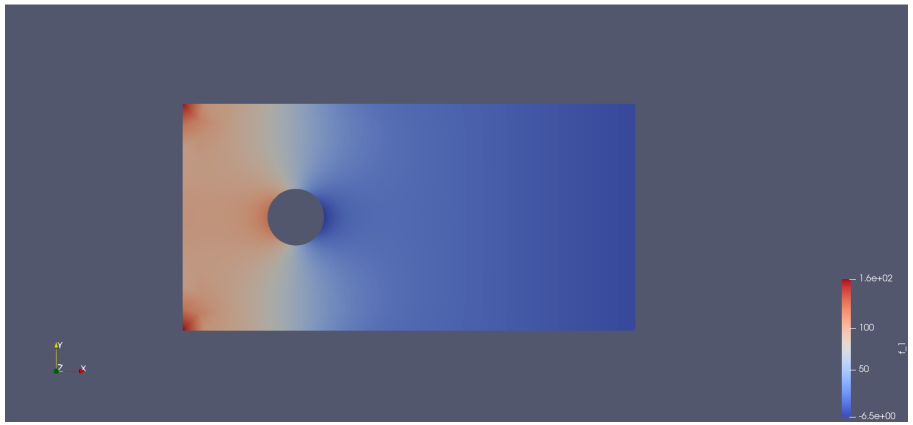


Figure – Pressure

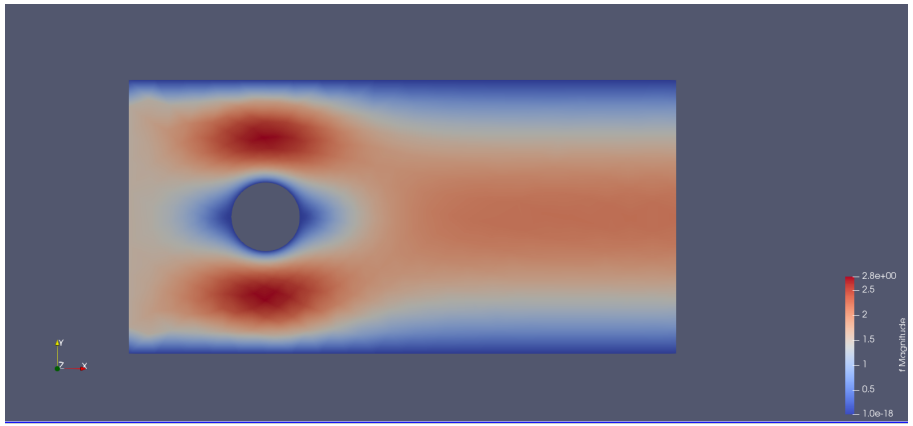
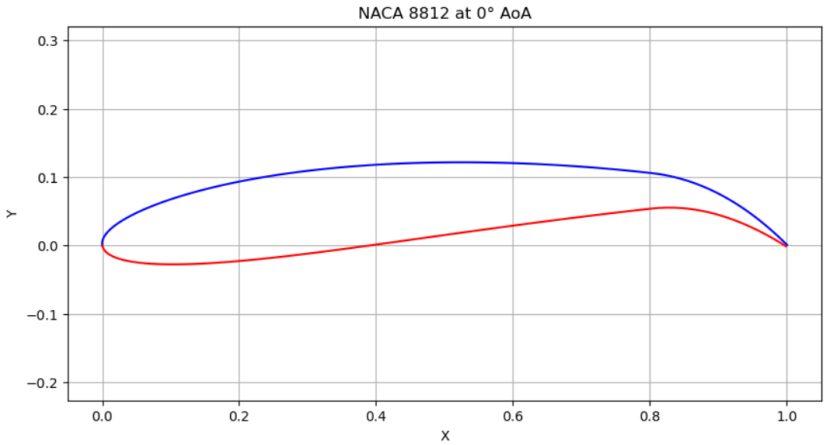
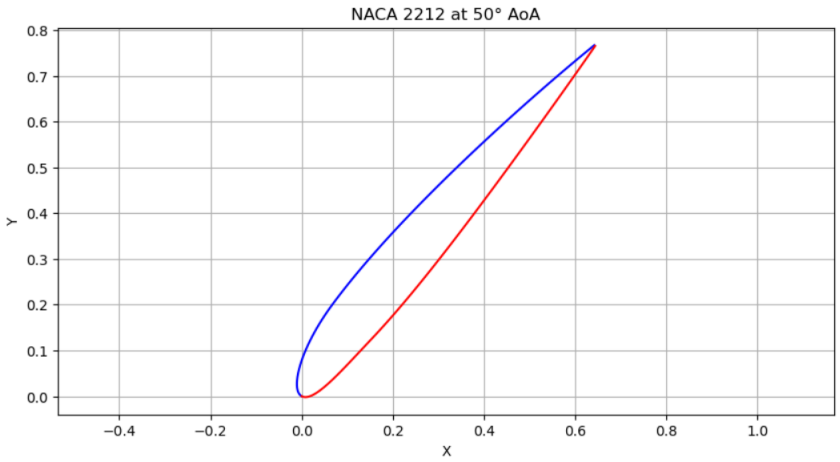


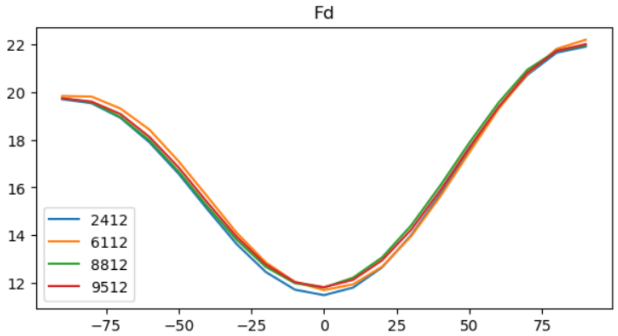
Figure – Velocity

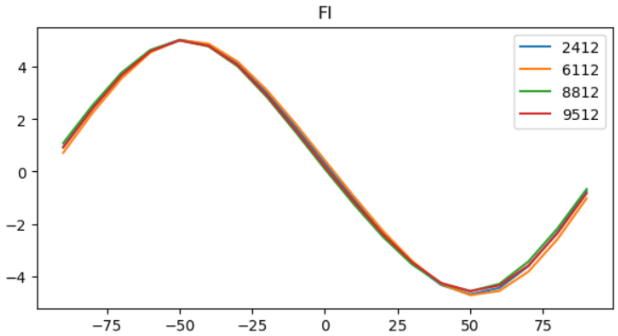
Worksheet 3 : Results

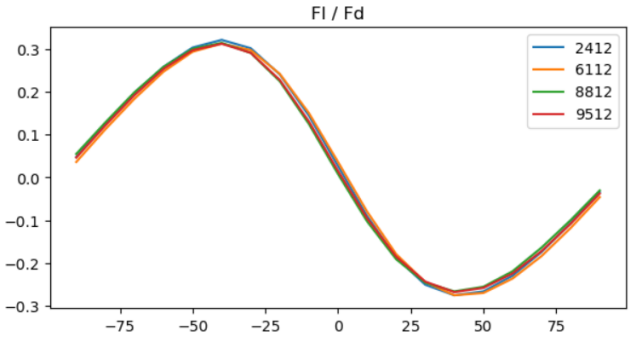


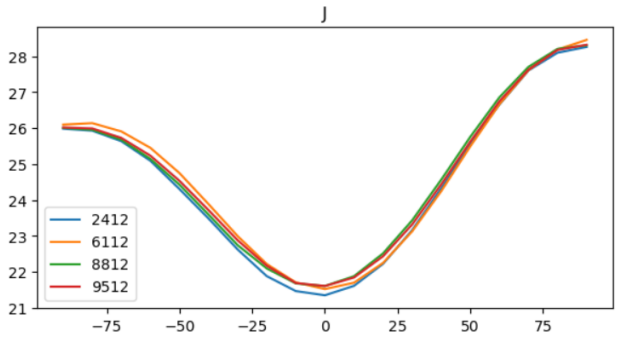


All cambers









Worksheet 3D : Results

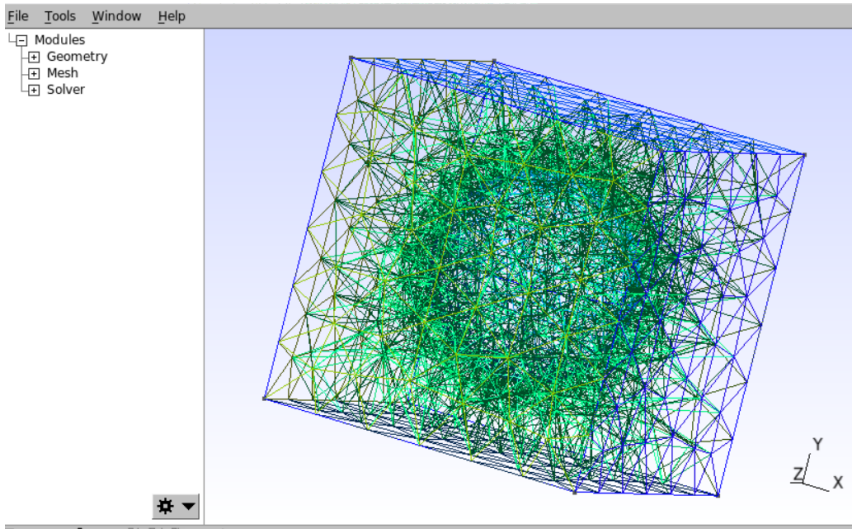


Figure – Mesh of a cube

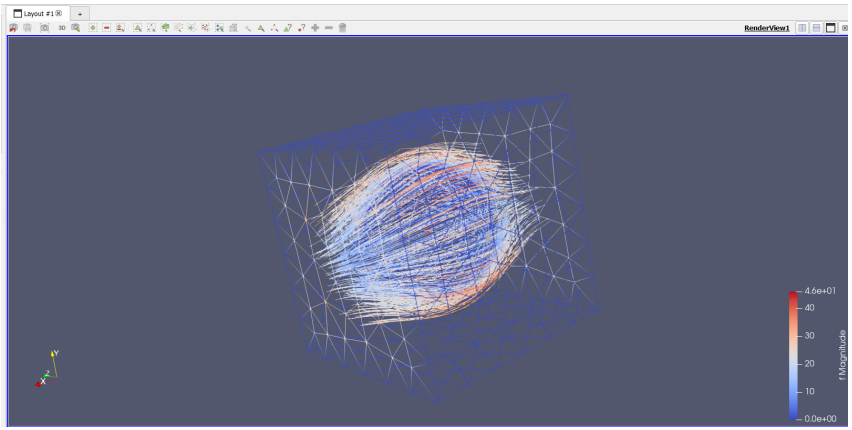
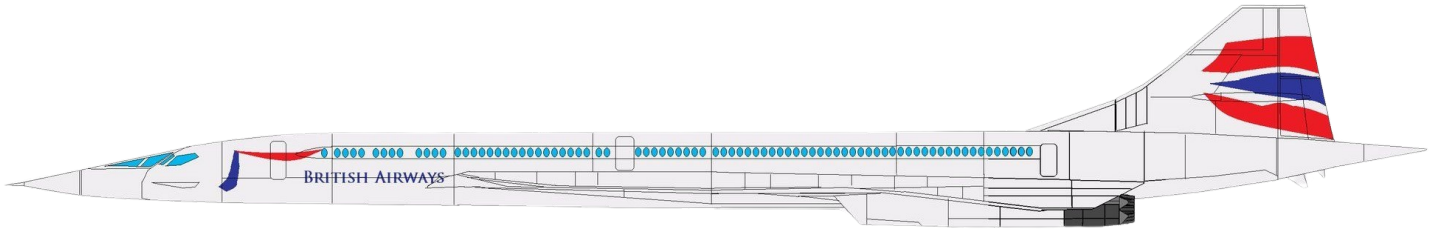


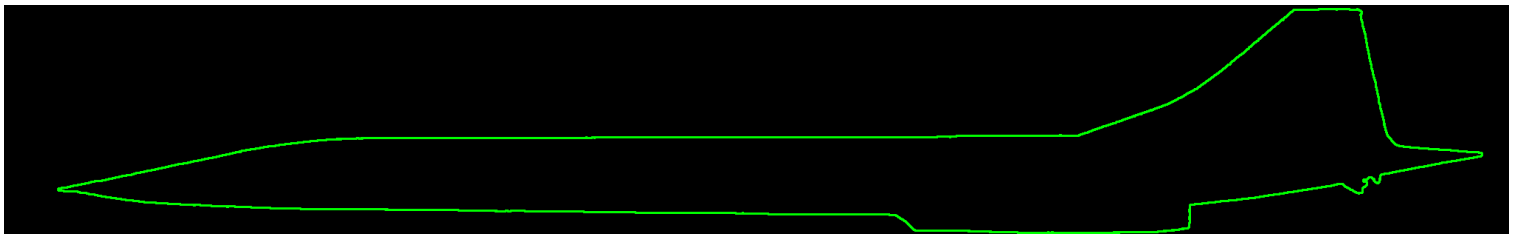
Figure – Velocity inside the cube

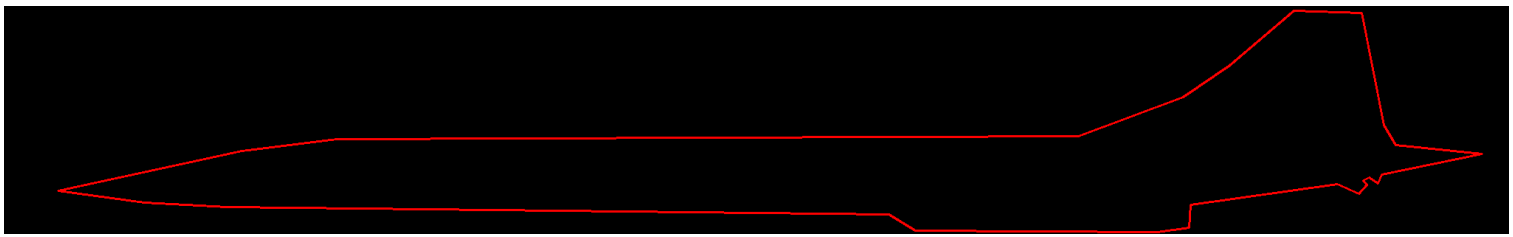
2D Simulations with Concorde

Generating the Mesh

It was created a script that uses computer vision to get the silhouette of any airplane, with smoothing and interpolation techniques if needed.







Visualization of the Results

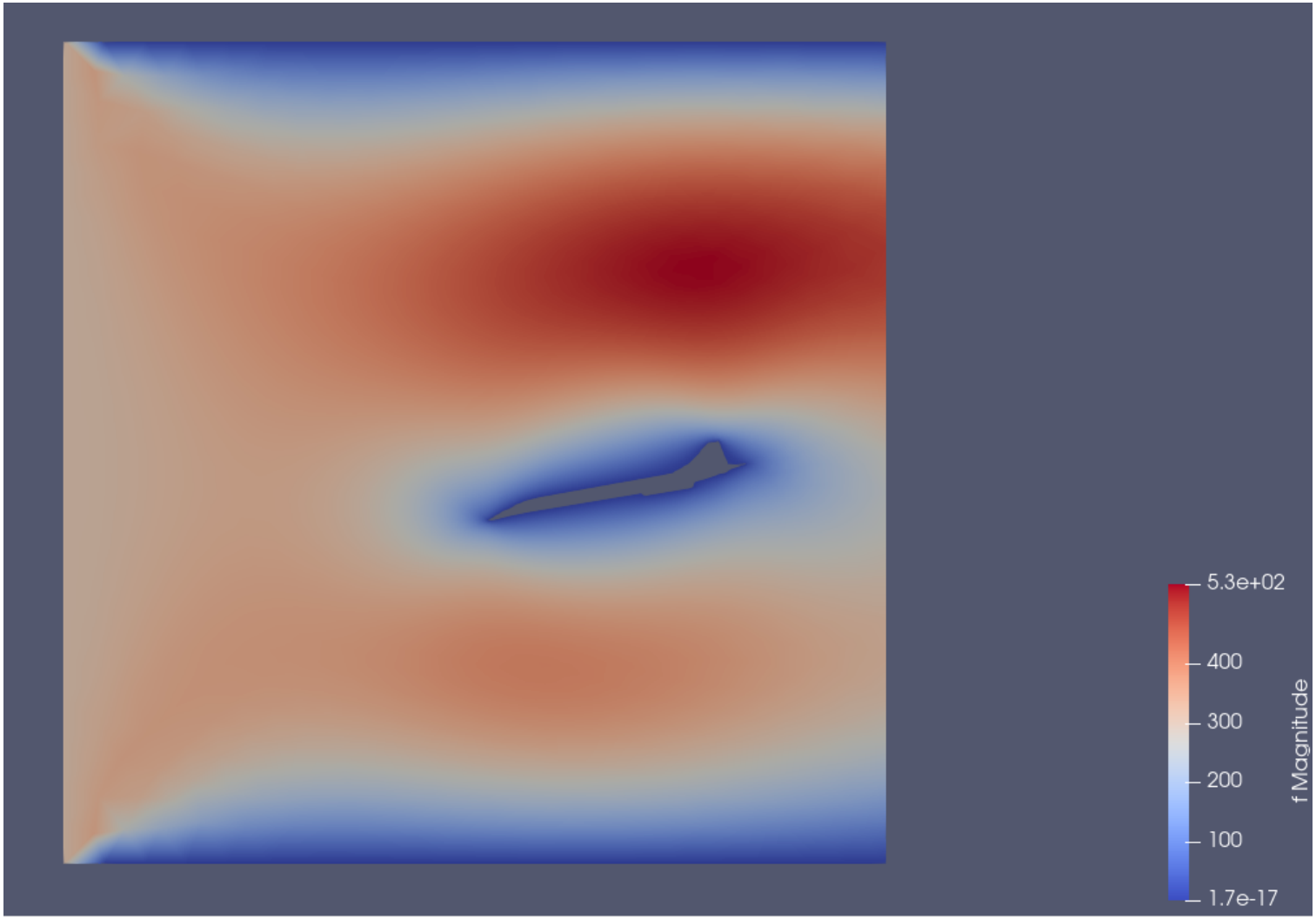


Figure – Visualization of Concorde with drop noose in ParaView.

Comparison between models

Main take-away : It was already expected a lower drag for the Concorde in normal operation, but it's possible to see that the engineers did a great job in projecting the broken nose, since it has almost the same drag during the take-off, which is when more fuel is used

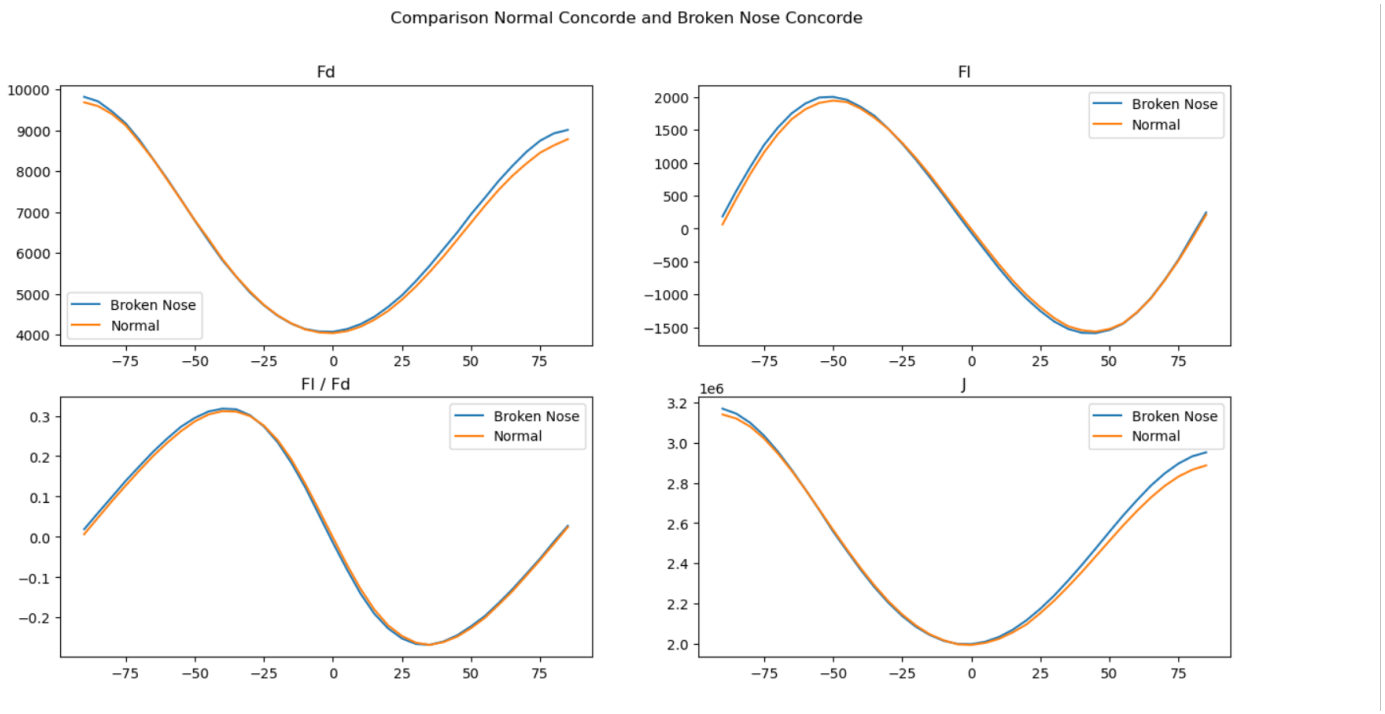


Figure – Comparison between Concorde with and without drop nose.

Meshing 3D

Surface Mesh Generation and Issues with STL Mesh

Problem of Original STL Mesh : STL is a common model file format in online libraries, optimized for sharing and 3D printing with minimal mesh elements. However, this often results in elongated elements and non-closing surfaces, which are not suitable for numerical simulation.

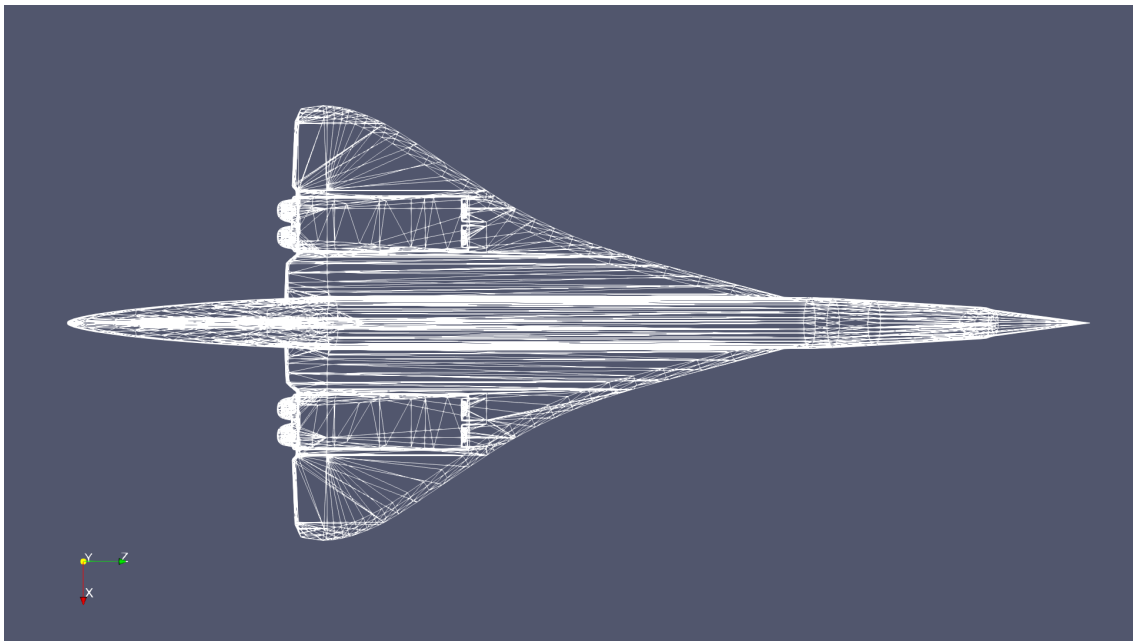


Figure – Original STL mesh showing elongated elements and gaps.

Solving Mesh Problems with Rhinoceros

ShrinkWrap with Rhinoceros : Utilizing Rhinoceros 8's ShrinkWrap function, we can generate a dense, closed mesh that encapsulates the original model. High smooth iteration settings are recommended for optimal mesh quality.

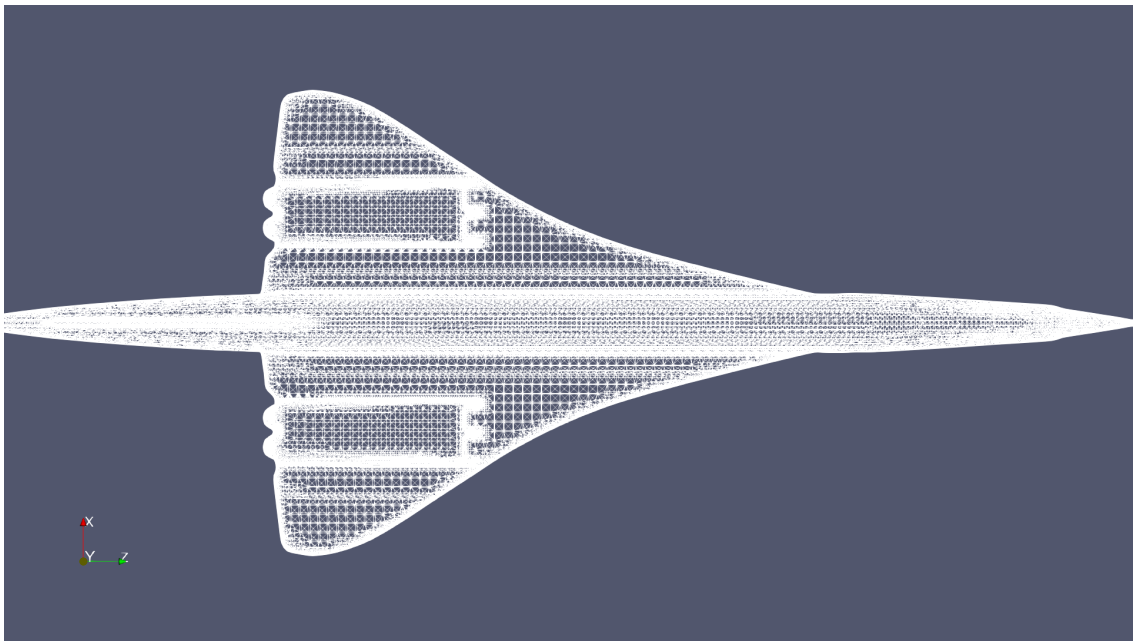


Figure – Mesh after applying ShrinkWrap, showing a denser and closed surface.

QuadRemesh for Optimized Mesh

QuadRemesh with Rhinoceros : The initial dense mesh is further processed using QuadRemesh to achieve an optimized mesh size for numerical simulation, allowing for various levels of refinement.

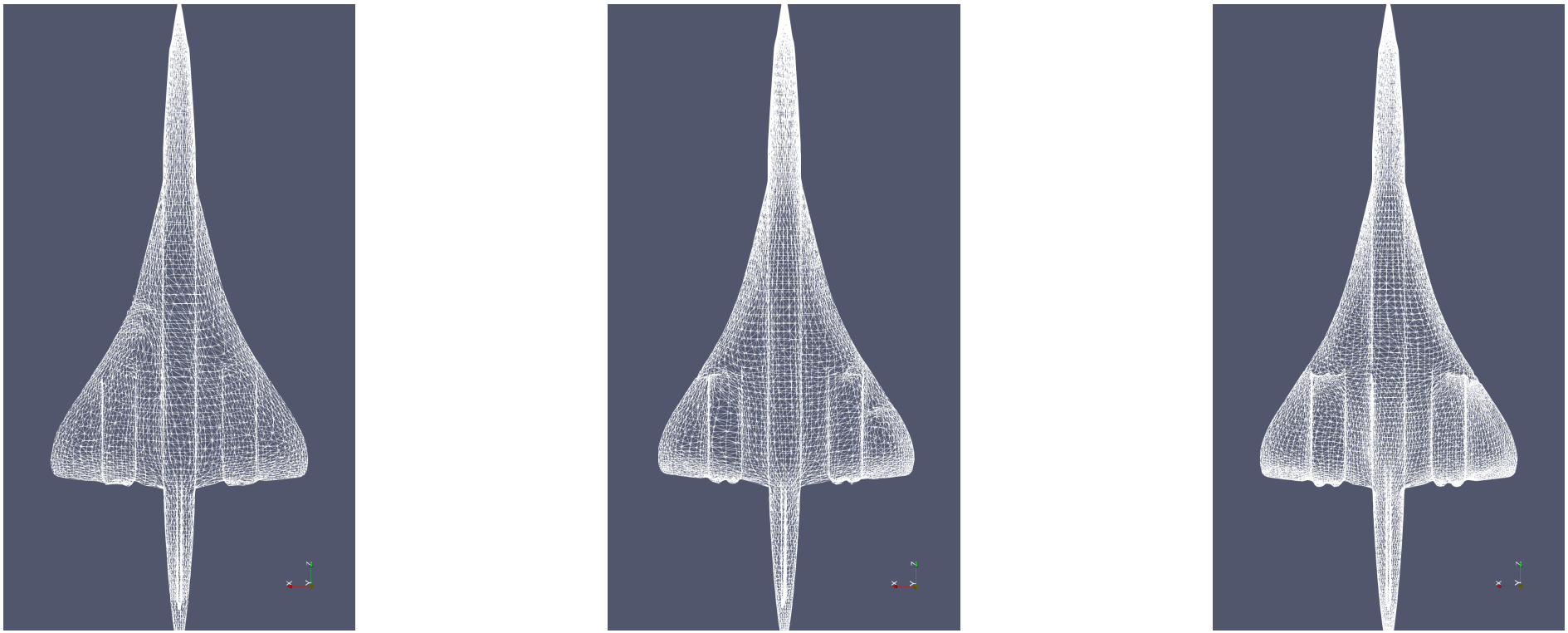


Figure – Mesh with 500 (left), 1000 (middle), and 2000 (right) surface elements, rotated for side-by-side comparison.

From STL to STEP : Ensuring Compatibility

Mesh Compatibility : STL formats, while popular, are not directly compatible with tools like Gmsh for 3D mesh generation. Conversion to STEP format via Rhinoceros' ToNURBS function facilitates the use of parametric surface panels for further processing.

Figure – 3D Mesh software.

3D Meshing with Gmsh

With a suitable surface mesh, the focus shifts to generating a 3D mesh in Gmsh. This step includes manipulating the mesh to simulate various angles of attack for performance testing.

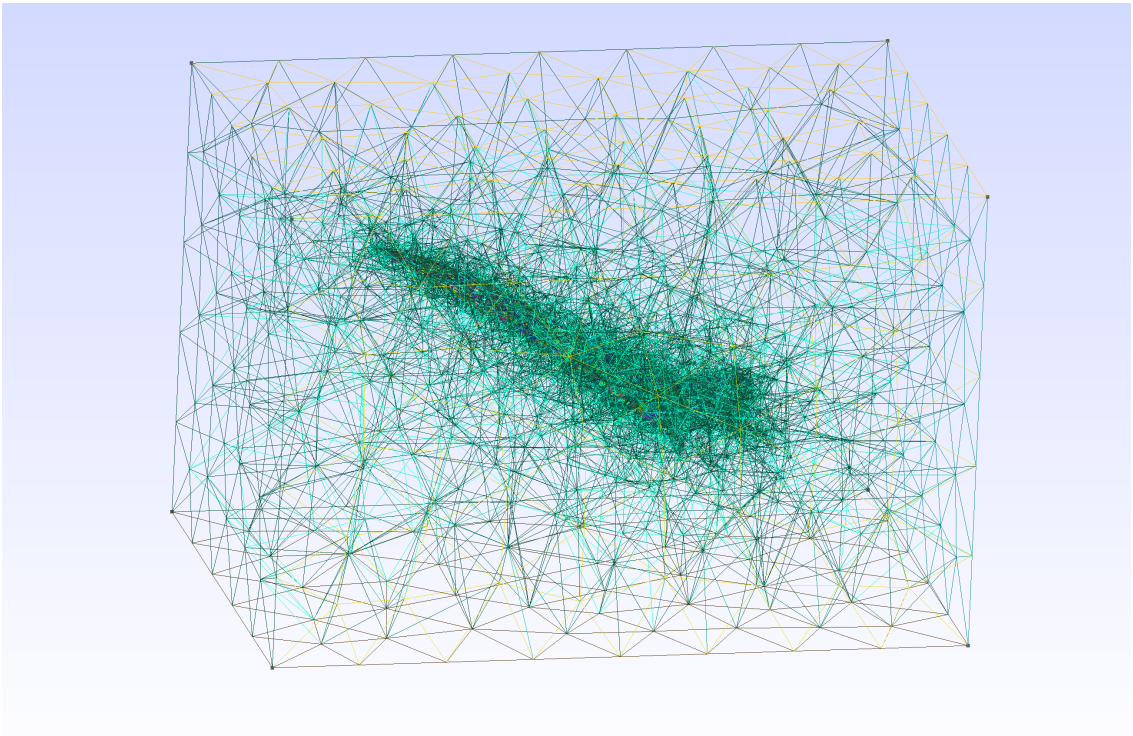


Figure – 3D Mesh generated in Gmsh for simulation purposes.

Gmsh Scripting for 3D Mesh

3D Meshing Script : The following Gmsh script outlines the process for 3D meshing, including operations like merging, rotating, and defining physical volumes and surfaces for simulation.

```
1 SetFactory("OpenCASCADE");
2 Merge "Concorde_skin_coarse1000.stp";
3 Box(2) = {-150, -150, -150, 300, 300, 300};
4 Rotate {{1,0,0}, {0, 0, 0}, -Pi/6} { Volume{1}; };
5 BooleanDifference(3) = { Volume{2};Delete;  }{ Volume{1};
      Delete;};
6 ...
7 Mesh 3;
```

This script demonstrates the procedure for setting up a 3D mesh in Gmsh, highlighting the importance of specific operations and definitions for simulation readiness.

Solving Stokes equation in 3D

Mathematical Formulation

Stokes' equations describe the motion of viscous fluids at low Reynolds numbers, where the effects of viscosity are dominant over those of inertia.

Mathematical Formulation of the 3D Stokes' equations :

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= 0 \quad \text{in } \Omega, \\ \nabla \cdot \mathbf{u} &= 0 \quad \text{in } \Omega, \end{aligned}$$

where $\mathbf{u}$ is the velocity field and p is the pressure in the three-dimensional domain Ω.

Boundary Conditions

Boundary Conditions : they are essential for defining the behavior of the fluid at the limits of the domain of interest and for ensuring the physical and mathematical correctness of the solutions to the 3D Stokes equations.

$$\mathbf{u} = (0,0)^T \tag{1}$$

$$\mathbf{u} = \left(-\frac{1}{24}(y-6)(y+6), 0\right)^T \tag{2}$$

$$-p\mathbf{n} + \nabla\mathbf{u} \cdot \mathbf{n} = 0 \tag{3}$$

Equation (1) enforces the no-slip condition at walls and obstacles, ensuring fluid velocity is zero, as per viscosity laws.
Equation (2) models a parabolic inflow velocity, characteristic of laminar flow in conduits.
Equation (3) ensures that the outgoing flow is not influenced by non-physical external forces, allowing the fluid to exit the domain freely.

Weak Formulation

Weak Formulation of the 3D Stokes' Equations : It is a powerful tool to analyze and solve viscous flow problems, transforming the physical laws governed by differential equations into a more manageable mathematical optimization problem.
The weak formulation is given by :

$$\int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v}\, dx - \int_\Omega p\, \mathrm{div}(\mathbf{v})\, dx + \int_\Omega \mathrm{div}(\mathbf{u})\, q\, dx = 0$$

where $\mathbf{u} \in \mathbf{V}$ is the velocity and $p \in Q$ is the pressure for all $(\mathbf{v}, q) \in \mathbf{V} \times Q$.

This formulation integrated into the boundaries conditions contributes to determining a unique and physically realistic solution for the velocity field $\mathbf{u}$ and the pressure p within the domain Ω.

Software Architecture : Computing different angles

It was created a python routine to read a geo file, alter the angle based on the input from the user, save a new geo file, export as a .msh file and read the mesh to be used. This automation significantly simplifies the testing procedure.

```python
class Airplane3D:

    def __init__(self, base_file) -> None:
        self.base_file = base_file
        self.new_angle_file = '.'.join([self.base_file.split(".")[0]+"newangle", "msh"])
        self.mesh_file = '.'.join([self.base_file.split(".")[0], "msh"])



    def change_attack_angle(self, file, attack_angle):
        with open(file, 'r') as f:
            linhas = f.readlines()

        # Localiza e substitui o valor "k"
        novas_linhas = []
        for linha in linhas:
            if 'angle' in linha:
                linha = linha.replace('angle', str(radians(attack_angle)))
            novas_linhas.append(linha)

        # Escreve as linhas modificadas para o novo arquivo
        with open(self.new_angle_file, 'w') as f:
            f.writelines(novas_linhas)


    def geo_to_msh(self, file):
        os.system(f'gmsh -3 -format msh2 {file} -o {self.mesh_file}')



    def save_mesh_file(self, attack_angle):
        self.change_attack_angle(self.base_file, attack_angle)
        self.geo_to_msh(self.new_angle_file)
        self.delete_auxiliar_files(remove_mesh=False)
```

Figure – Code to change the attack angle of 3D airplane.

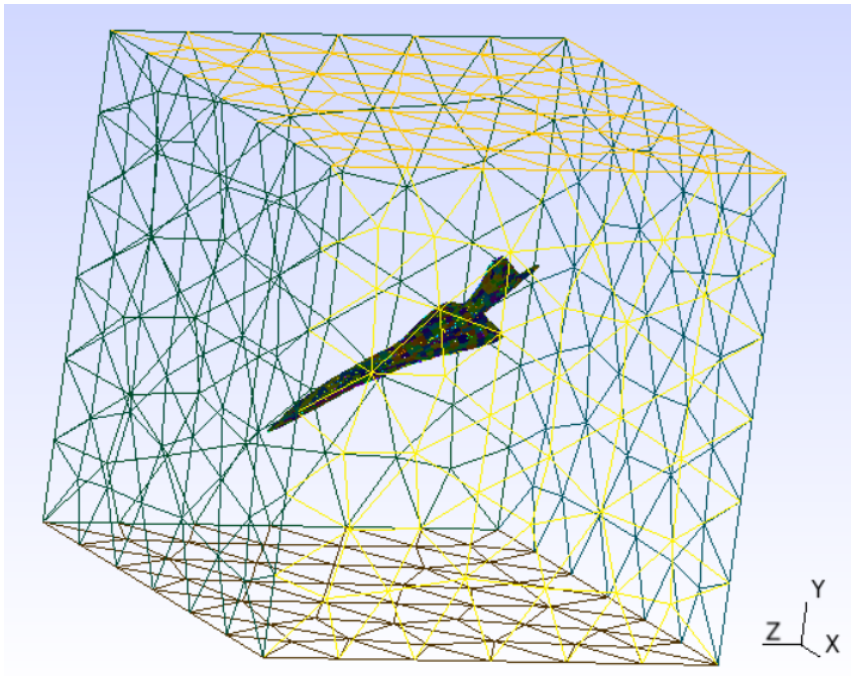


Figure – Concorde 3D with 30 °attack angle.

Software Architecture : Objects

```python
class ONERA:

    def solve_stokes(self, mesh, facets, inflow=None):
        if inflow:
            return self.stokes.solve(mesh, facets, inflow)
        else:
            return self.stokes.solve(mesh, facets)

    def solve_navier_stokes(self, mesh, facets):
        return self.navier_stokes.solve(mesh, facets)

    def solve_and_save_stokes(self, mesh, facets, path,
                              inflow: callable = lambda x: (np.stack((np.ones(x.shape[1]), np.zeros(x.shape[1]))))):
        j, fd, fl, u, p = self.stokes.solve(mesh, facets, inflow)
        self.stokes.save(mesh, u, p, path)



    def solve_and_save_stokes_in_range_angle(self, generare_airplane_function: callable, save: bool = False, path: str = None,
                                             angles: list = list(np.arange(-90, 90, 5)),
                                             inflow: callable = lambda x: (np.stack((np.ones(x.shape[1]), np.zeros(x.shape[1]))))
                                             ):
        j_values = []
        fd_values = []
        fl_values = []
        for angle in angles:
            print(angle)
            mesh, _, facets = generare_airplane_function(angle)
            j, fd, fl, u, p = self.solve_stokes(mesh, facets, inflow)
```

Figure – Part of the main code.

```python
if __name__=="__main__":
    onera = ONERA()
    inflow_speed = lambda x: (np.stack((np.ones(x.shape[1])*300, np.zeros(x.shape[1]))))
    mesh, _, facets = onera.generate_3d_concorde(10)
    onera.solve_and_save_stokes(mesh, facets, path="final/airplane_3d/normal_results/", inflow=inflow_s


    result = onera.solve_and_save_stokes_in_range_angle(onera.generate_2d_concorde_broken_nose, save=Tr
                                                        path="final/airplane_2d/result_concorde_broken
    onera.plot_stokes_results("final/airplane_2d/result_concorde_broken_nose.csv")
```

Figure – Implementation example.

- generate_2d_airplane.py
- generate_2d_foil.py
- generate_3d_airplane.py
- main.py
- onera.py
- solve_navier_stokes.py
- solve_stokes.py
- utils.py

Figure – Codes and Classes.

Overview

The Concorde, known for its supersonic capabilities, presents unique challenges for aerodynamic modeling. Our approach utilizes the Stokes equations, assuming non-viscous and incompressible flow conditions $\nabla \cdot \mathbf{u} = 0$. This simplification, while practical for many scenarios, limits the accuracy of high-speed, especially supersonic, simulations. Despite this, we've conducted 3D simulations to analyze the Concorde's aerodynamic performance across various angles of attack and speeds, employing MPI for computational efficiency (refer to the MPI implementation section).

Results :

The simulations yield detailed profiles of the Concorde's drag and lift characteristics. Notably, the lift-to-drag ratio remains consistent across varying speeds, including transonic regimes. This consistency, although theoretically interesting, highlights the limitations of our mathematical model in capturing the complexities of real-life supersonic flight dynamics.

3D Simulation Results

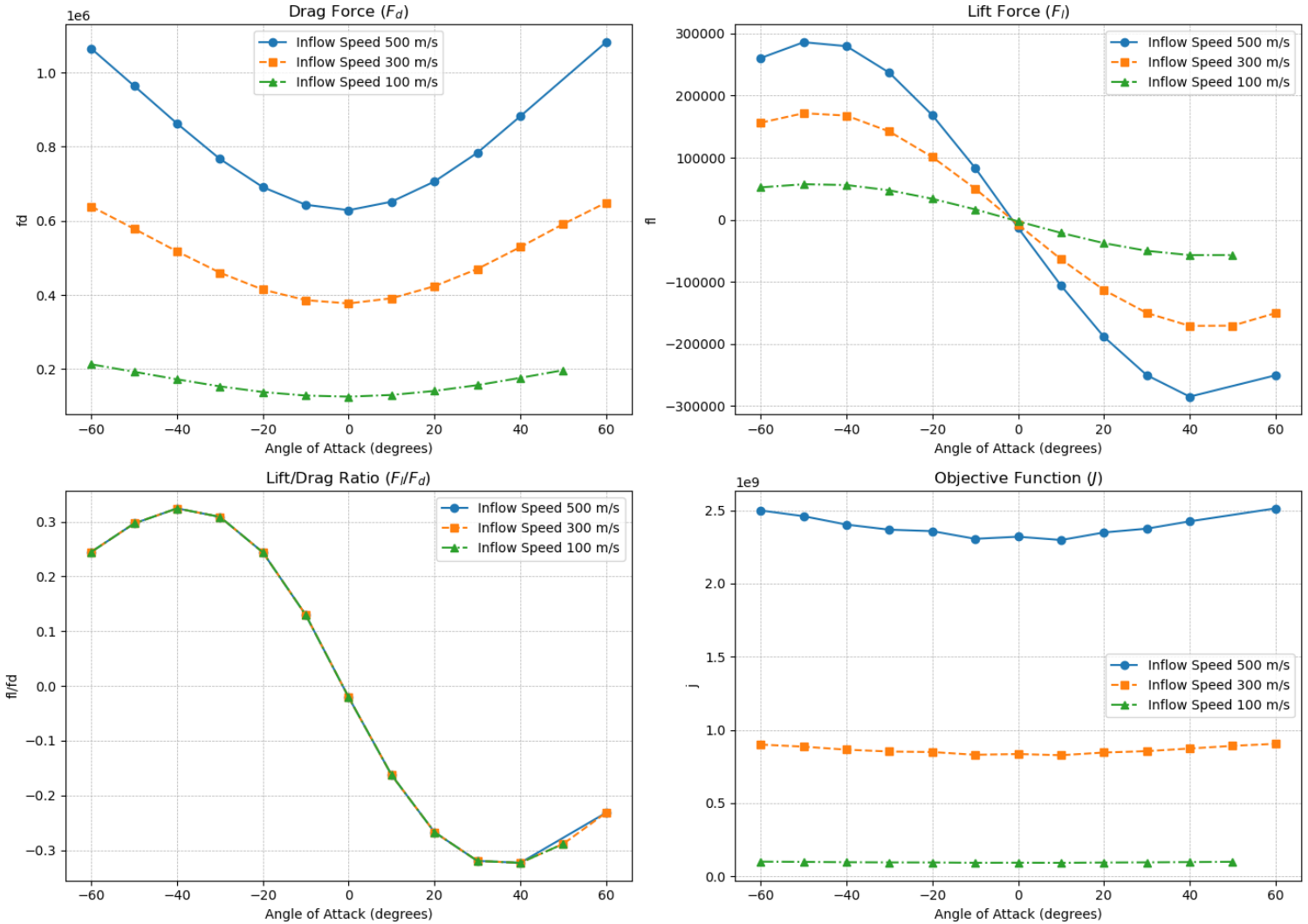


Figure – Drag, lift, and lift-to-drag ratio at speeds of 100 m/s, 300 m/s, and 500 m/s.

Streamline Visualization in ParaView

Streamline Visualization : Streamlines provide an intuitive way to visualize fluid flow patterns by tracing the paths that fluid elements follow. This technique is particularly useful for analyzing the flow inside and around complex geometries, such as aircraft surfaces.

Implementation in ParaView :

Wireframe Representation : To begin, adjust the visualization representation to "Wireframe." This enables a clear view of the internal flow paths by making the model's surface transparent.

Stream Tracer Application : Next, apply the "Stream Tracer" filter with the seed type configured as "Point Cloud." This step initializes the streamlines from specified seed locations within the flow field.

Tube Filter Enhancement : Finally, enhance the visibility of the streamlines by applying the "Tube" filter. This filter thickens the streamline paths, making them more distinguishable in the visualization.

Streamline Visualization in ParaView

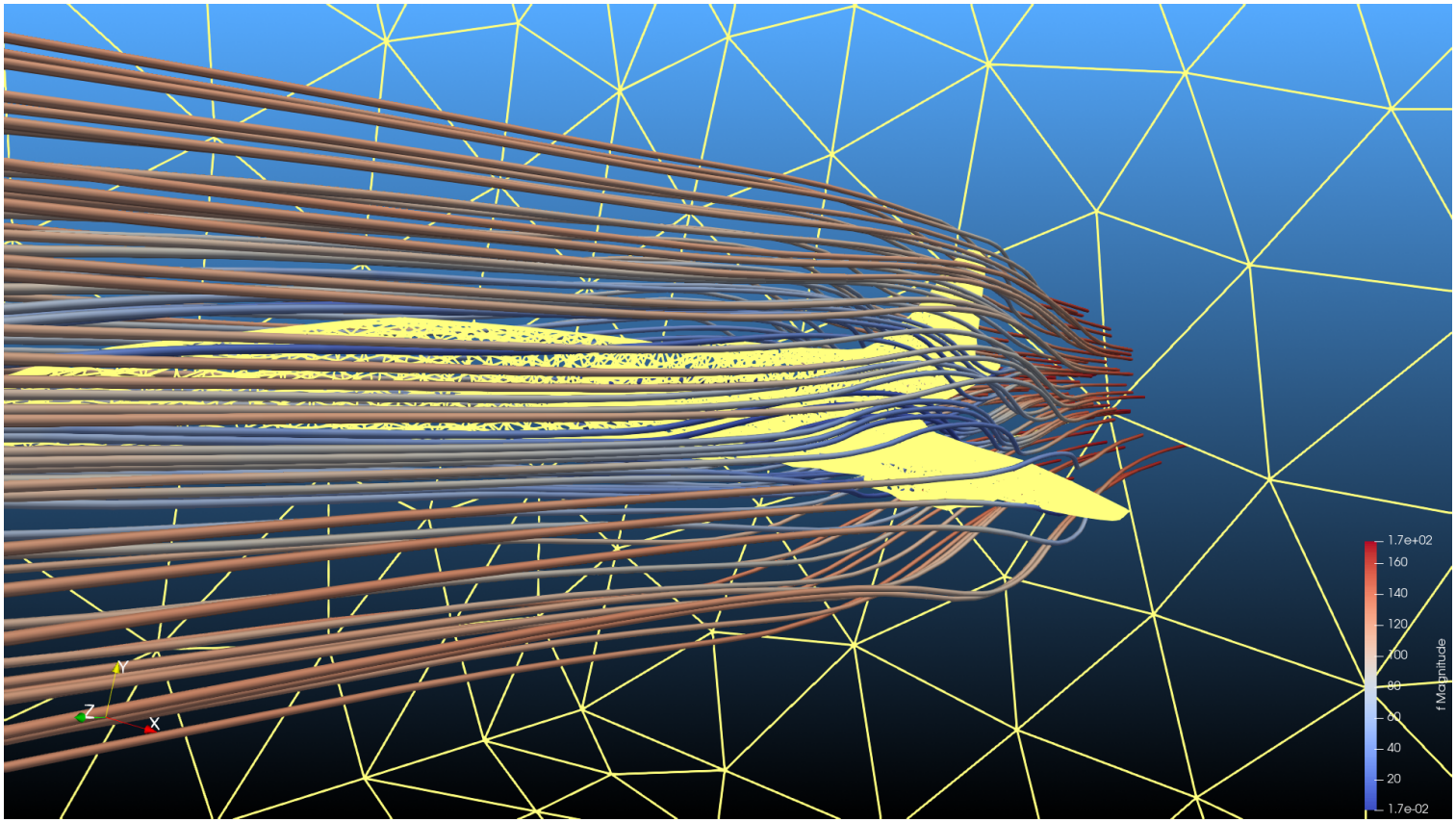


Figure – Streamline visualization of concorde using ParaView.

Streamline Visualization in ParaView

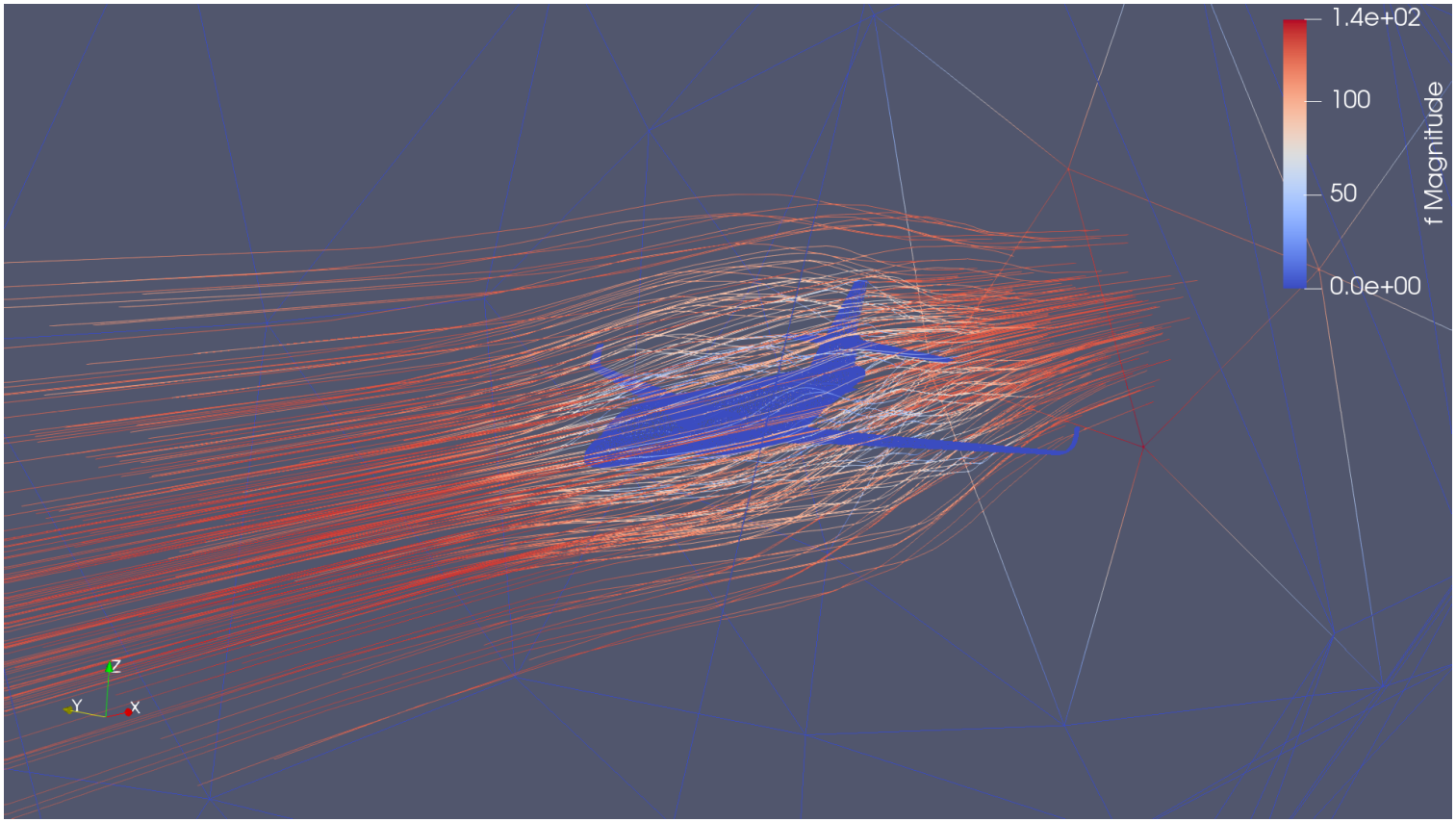


Figure – Streamline visualization of falcon using ParaView.

Solving N-S equation in 3D

Solving N-S equation in 3D

Physical sense of the N-S equation : FPD applied to a fluid in 3D, it forms a system of eqution for the velocity u and the pression p.
Mathematical expression :

$$\begin{aligned} \rho(\frac{\partial u}{\partial t} + u.\nabla u) &= \nabla.\sigma(u,p) + f \\ \nabla.u &= 0 \end{aligned} \tag{4}$$

Where $\sigma(u,p)$ takes into account the elasticity end the viscosity of the fluid.

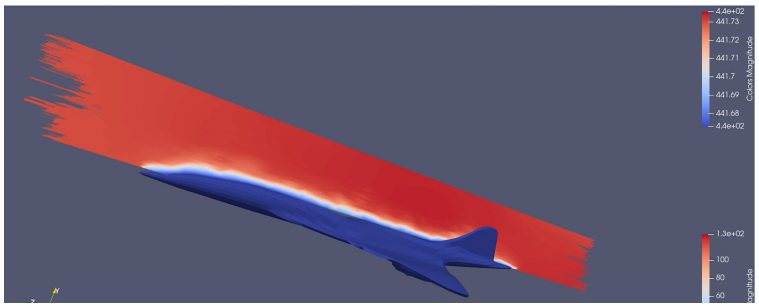
Variational Formulation

Using the following notations : $\langle v, w \rangle = \int_\Omega vw \, dx$, $\langle v, w \rangle_{\partial\Omega} = \int_{\partial\Omega} vw \, ds$, $u^{n+\frac{1}{2}} = \frac{u^n + u^{n+1}}{2}$ and μ the dynamic viscosity, we can write the $n+1$th step as find u^* such that :

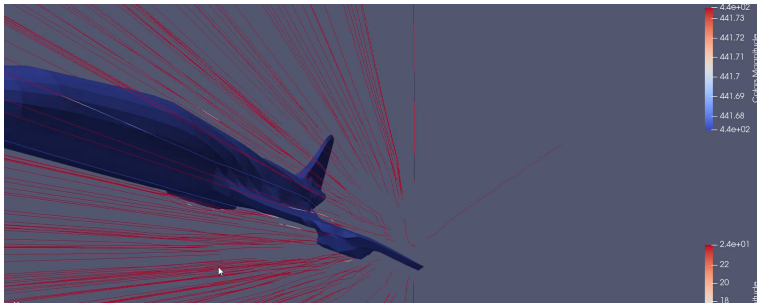
$$\langle \rho \frac{u^* - u^n}{\Delta t}, v \rangle + \langle \rho u^n . \nabla u^n, v \rangle + \langle \sigma(u^{n+\frac{1}{2}}, p^n, \epsilon(v) \rangle + \langle p^n n, v \rangle_{\partial\Omega} \\ - \langle \mu \nabla u^{n+\frac{1}{2}} . n, v \rangle_{\partial\Omega} = \langle f^{n+1}, v \rangle \tag{5}$$

Results in ParaView of Navier-Stolkes with Concorde

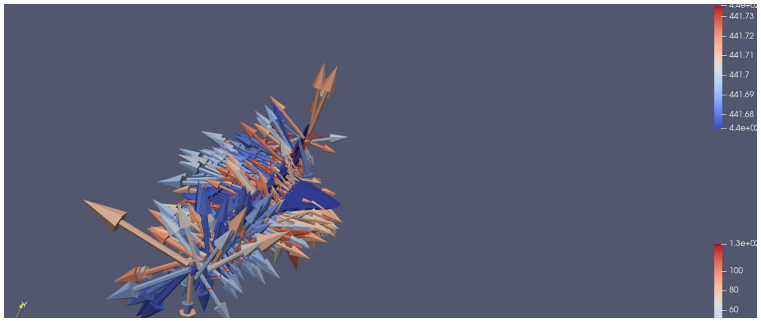
For the tests, we used $\mu = 1$ and $\rho = 1$, with inlet speed starting from $11m/s$ to $110m/s$. No slip condition was applied in the walls and in the airplane surface, and finally the outlet pressure was set to zero. We simulated with a duration of 10 seconds and 100 steps.



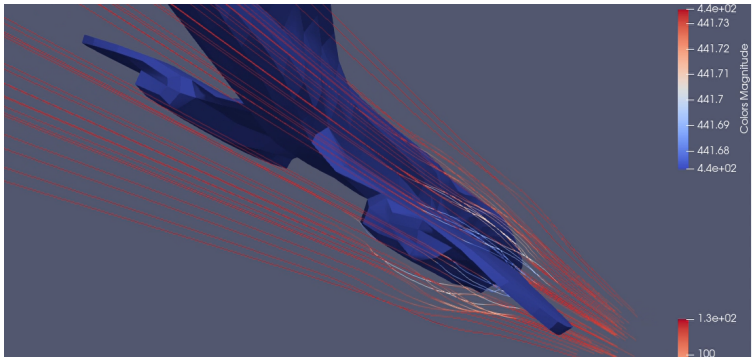
(a)



(b)



(c)



(d)

Comparisons of Navier-Stolkes in different velocities and angles

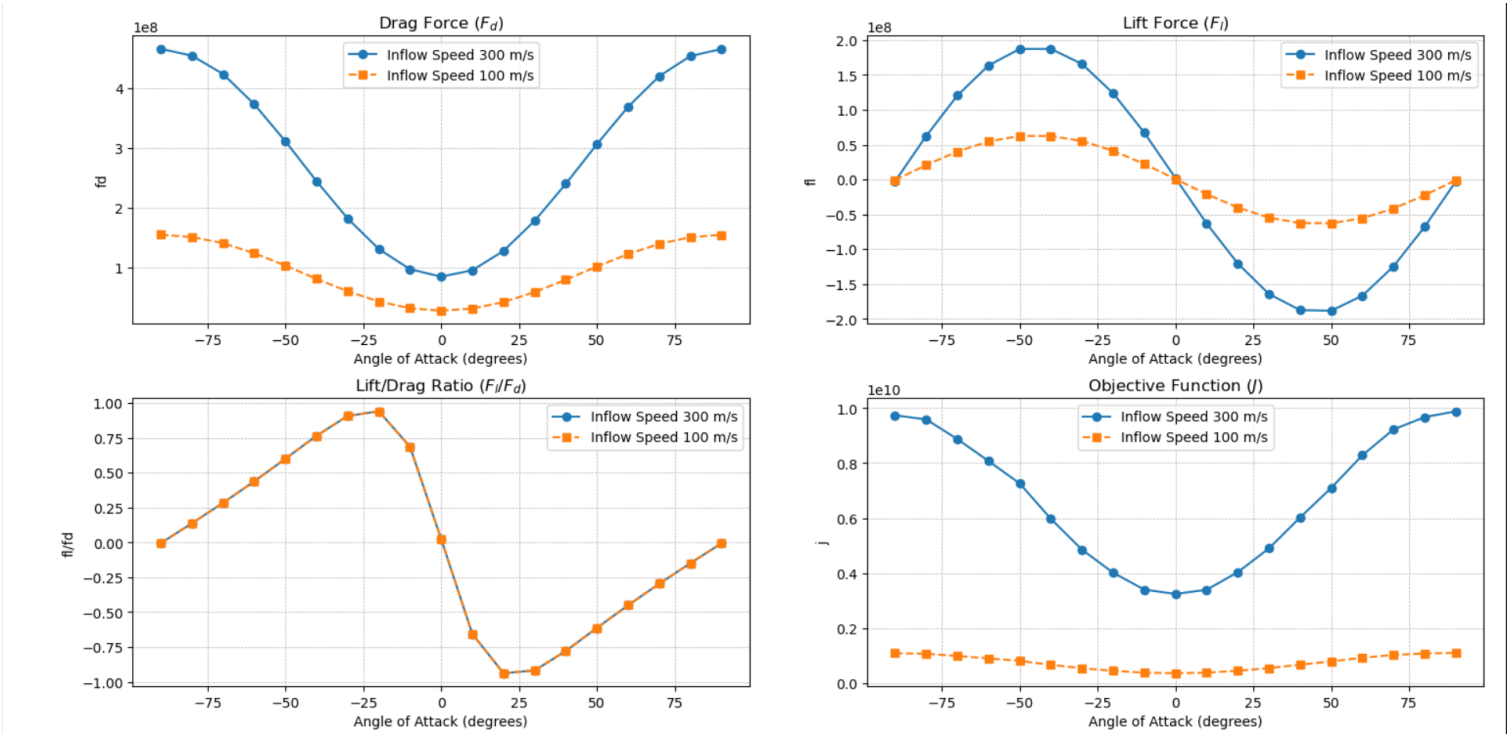


Figure – Drag, lift, and lift-to-drag ratio at speeds of 100 m/s, 300 m/s, and 500 m/s of NS equation

Comparisons of Navier-Stolkes in different velocities and angles

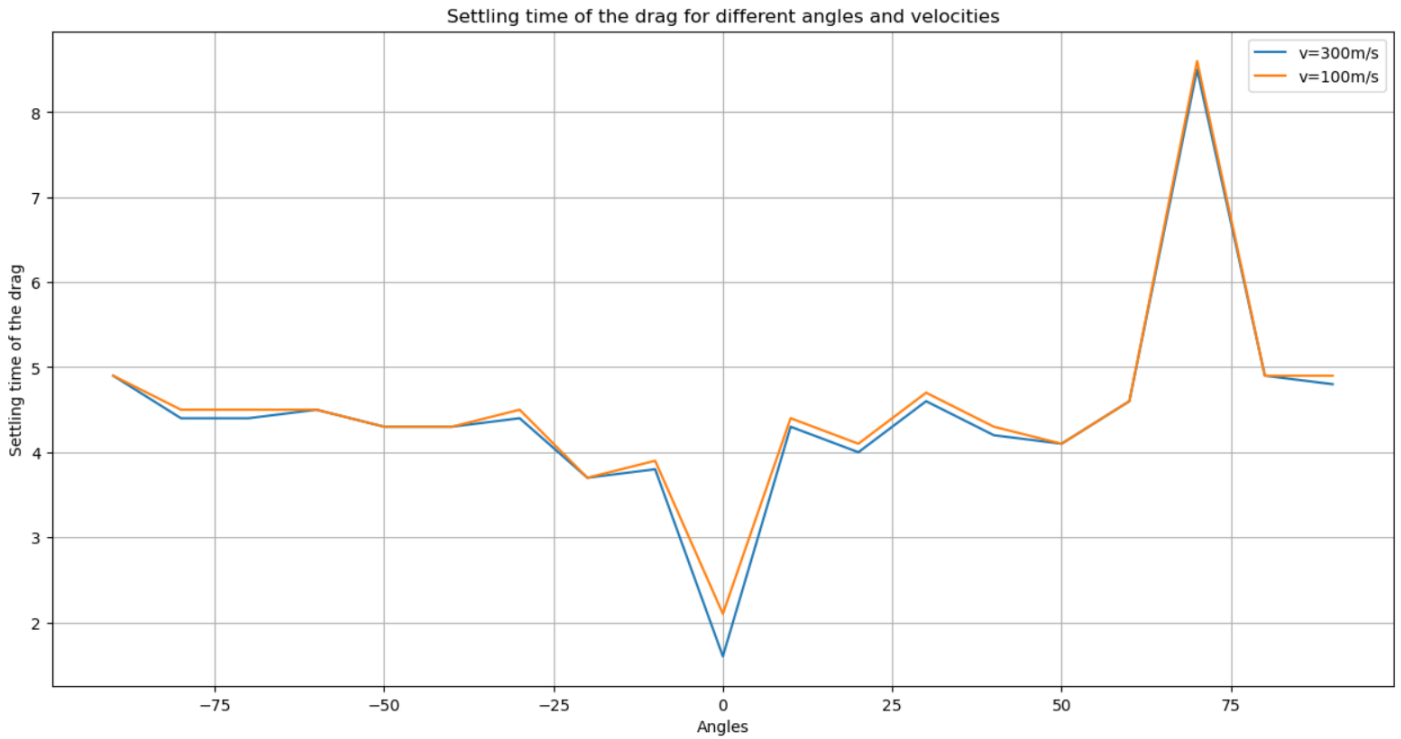


Figure – Settling time for drag at different AoA

Comparisons of Navier-Stolkes in different velocities and angles

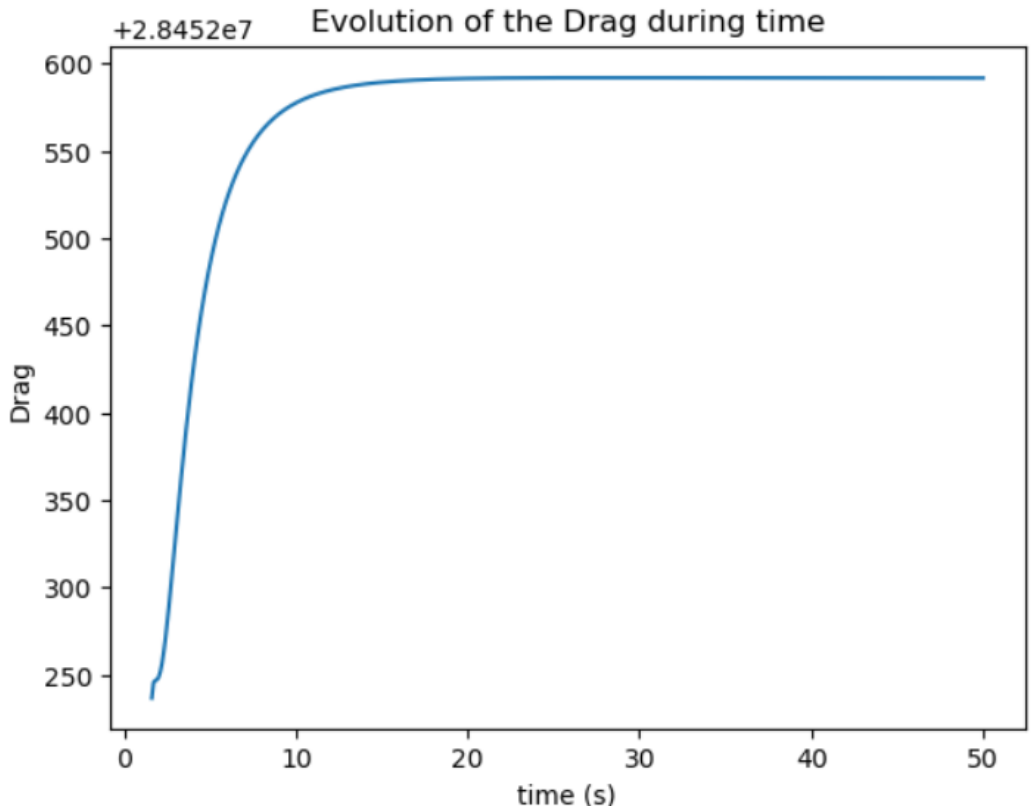


Figure – Settling time for drag at different AoA

MPI implementation

Mesh Partitioning with FEniCSx and MPI

Automatic Mesh Partitioning in FEniCSx :

```
msh = file.read_mesh(ghost_mode = dfx.cpp.mesh.GhostMode.
    shared_facet)
```

Default ghost_mode : shared_facet, suitable for general partitioning.
Inspecting Mesh Partition : View partition details per MPI rank.

```
comm = MPI.COMM_WORLD
def mpi_print(s):
    print(f"Rank {comm.rank}: {s}")
tdim = msh.topology.dim
msh.topology.create_connectivity(tdim - 1, tdim)
mpi_print(f"Ghost cells (global numbering): {msh.topology
    .index_map(tdim).ghosts}")
```

Assembling Global Values Across Ranks

Distributed Mesh Handling : Sum contributions from each rank for global values of J, F_{lift}, and F_{drag}.

```
j, fd, fl, u, p = self.solve_stokes(mesh, facets, inflow)
j_val = np.array(j, dtype='float64')
global_j = np.empty_like(j_val)
comm.Reduce([j_val, MPI.DOUBLE], [global_j, MPI.DOUBLE],
    op=MPI.SUM, root=0)
if rank == 0:
    j_values_ls.append(global_j.item())  # Convert to
        scalar
```

MPI Performance on DCE Cluster

Test Environment : Dual-socket system, 2 Intel® Xeon® Silver 4214R CPUs, 24 physical cores. We use a model with about 7000 nodes.
MPI Setup Example :

```
mpirun -np 8 --map-by numa --bind-to core python3 test_mpi
```

Execution Time Performance :

-np Value	Execution Time (seconds)
2	23.10
4	13.20
8	8.612
16	7.274

Table – MPI Execution Times