Project 2 Solution: Supersonic Engine Analysis

AE 523, Computational Fluid Dynamics, Fall 2020
Kalvin Monroe - kymonroe

1. Introduction

In this project, we implemented a 2D Finite Volume Method (FVM) solver to simulate supersonic flow through an engine inlet. The solver is first-order and utilizes an adaptive approach where the mesh was refined between successive iterations. The 2D compressible Euler equations (see below) were used to govern the flow.

$$\begin{aligned}
&\frac{\partial}{\partial t}(\rho) &&+ \frac{\partial}{\partial x}(\rho u) &&+ \frac{\partial}{\partial y}(\rho v) &&= 0 &&\text{(mass)}\\
&\frac{\partial}{\partial t}(\rho u) &&+ \frac{\partial}{\partial x}(\rho u^2 + p) &&+ \frac{\partial}{\partial y}(\rho u v) &&= 0 &&\text{(x-momentum)}\\
&\frac{\partial}{\partial t}(\rho v) &&+ \frac{\partial}{\partial x}(\rho v u) &&+ \frac{\partial}{\partial y}(\rho v^2 + p) &&= 0 &&\text{(y-momentum)}\\
&\frac{\partial}{\partial t}(\rho E) &&+ \frac{\partial}{\partial x}(\rho u H) &&+ \frac{\partial}{\partial y}(\rho v H) &&= 0 &&\text{(energy)}
\end{aligned}$$

Specifically, by using an adaptive approach, a goal of this project was to observe and resolve the complex shock structure within the engine.

2. Problem Setup and Geometry

Figure 1 shows the physical setup and computational boundary conditions of the problem. Notably, the domain is comprised of four different boundaries: a supersonic inflow, a supersonic outflow, and engine exit, and an engine wall (adiabatic).

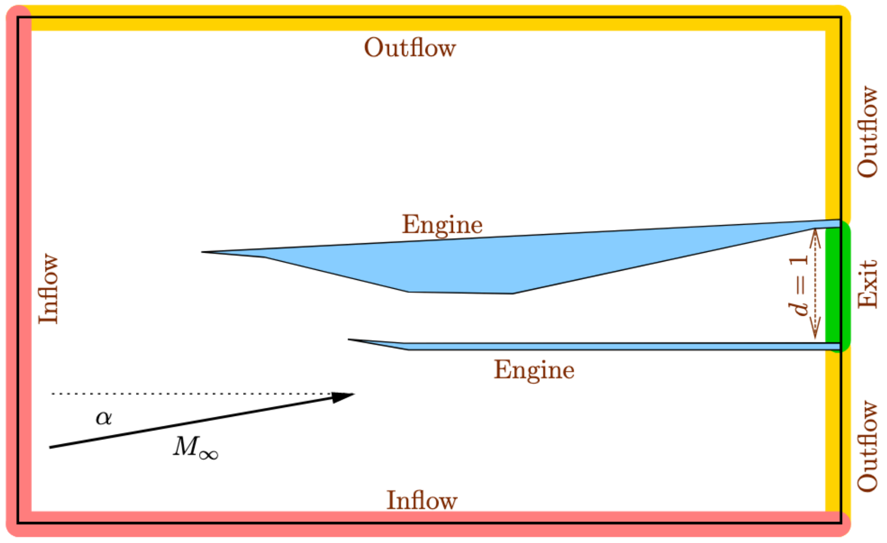


Figure 1 shows the geometry and potential flow solution of the domain

A freestream condition was given to us to us to serve as the inflow condition of our domain. In addition, we were asked to apply our solver against several angles of attack, seen below.

$$\mathbf{u}_\infty = [\rho, \rho u, \rho v, \rho E]^T = \left[1, M_\infty \cos\alpha, M_\infty \sin\alpha, \frac{1}{(\gamma-1)\gamma} + \frac{M_\infty^2}{2}\right]^T$$

$$M_\infty = 2.2$$
$$\gamma = 1.4$$
$$\alpha \in [0, 3°]$$

Additionally, we were initially provided with a baseline mesh of the solution. This mesh can be seen below in **Figure 2**. The purpose of this mesh was to serve as an initial guess to overlay onto further adaptive iterations.

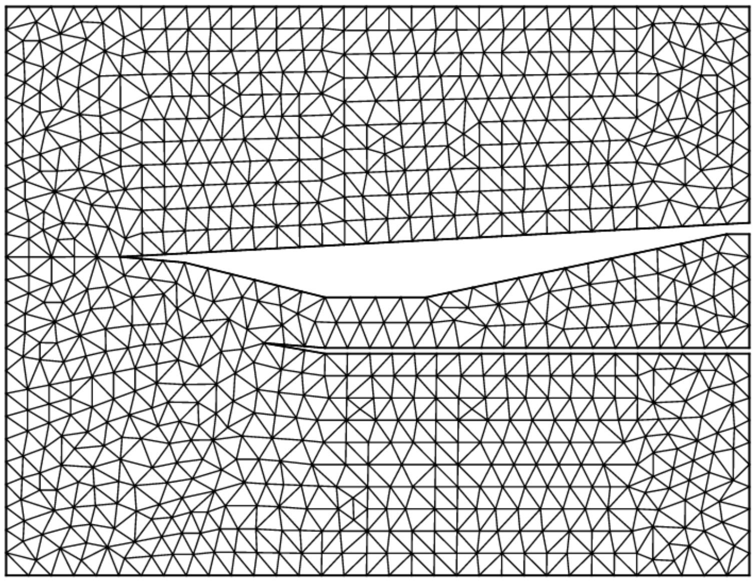


Figure 2 shows the baseline mesh and geometry

3. Numerical Method and Discretization

For this project, a First-Order FVM was implemented across the domain to solve for the flow throughout the engine. Specifically, a Forward Euler method with local time-stepping scheme is used to drive the solution to a steady state. The overall update equation for each cell in the domain can be seen below.

$$\mathbf{u}_i^{n+1} \quad = \quad \mathbf{u}_i^n - \frac{\Delta t_i^n}{A_i}\mathbf{R}_i(\mathbf{U}^n),$$

Here we see that the update is comprised of two components: A Flux Residual $\boldsymbol{R}$, and a scalar value of $\boldsymbol{\Delta t/A}$. For a triangular cell, as used in the discretized domain, the flux residual for a cell can be calculated as a sum of the state-fluxes across each side of the element (formula seen below). The specific formula of the state-flux will be described in a later section.

$$\mathbf{R}_i = \sum_{e=1}^{3} \hat{\mathbf{F}}(\mathbf{u}_i, \mathbf{u}_{N(i,e)}, \vec{n}_{i,e})\, \Delta l_{i,e}.$$

The scalar value of $\boldsymbol{\Delta t/A}$ can be found by relating it to the CFL number of for the cell. The formula for calculating it is seen below.

$$\frac{\Delta t_i}{A_i} = \frac{2\mathrm{CFL}_i}{\sum_{e=1}^{3} |s|_{i,e}\Delta l_{i,e}}.$$

Here, the denominator is again a sum across the three edges of a single element, with $\boldsymbol{s}$ corresponding to the maximum characteristic speed across an edge, and $\boldsymbol{\Delta l}$ corresponding to the side length of the edge. With this approach, the value of $\boldsymbol{\Delta t/A}$ will be different for each cell. Finally, we also define a new value to asses convergence: the L1 norm of the residual vector as defined below.

$$|\mathbf{R}|_{L_1} = \sum_{\text{cells } i} \sum_{\text{states } k} |R_{i,k}|$$

When this value has reached a value below 10^{-5}, we can deem the solution as converged.

4. Adaptation

As the given mesh is somewhat coarse, we are also tasked with implementing a mesh adaptation algorithm. Specifically, this algorithm will be required to flag edges that have a sufficiently high jump in Mach number for refinement. These flagged edges will then transform the original element into a new subset of multiple elements. This should help resolve certain features of the flow, such as shocks and expansion waves.

Specifically, the algorithm works by first determining the error across each edge according to the following formulas below, where M_k is the cell Mach number, and h_e is the edge length.

$$\text{interior: } \epsilon_e = |M_{k^+} - M_{k^-}| h_e.$$

$$\text{wall: } \epsilon_e = |M_k^{\perp}| h_e,$$

Once all errors have been calculated, the top 3% of the edges with the highest error are then flagged for refinement. Furthermore, to smooth out these refinements, any cells that have one edge flagged will automatically have *all* of their edges flagged. This results in the flagged elements falling into 3 categories, as seen below in Figure 3.

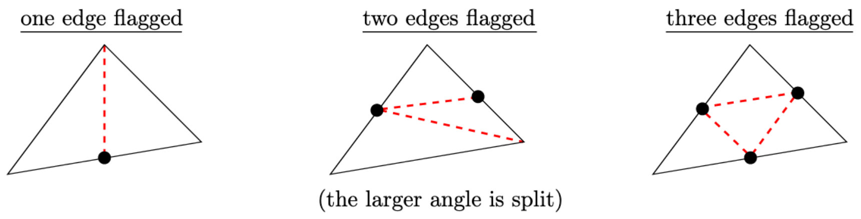


Figure 3 shows the 3 categories of refined elements from the adaptation algorithm

To help with convergence, the new elements within a cell will be initialized with the same condition as the original, larger element.

5. Questions and Tasks

5.1 Question 1 – Roe Flux

As stated in Section 3 above, the flux residual is a function of an implemented state-flux that is calculated across all sides of a cell. For this project, we implemented a Roe Flux. The implementation of this function can be seen in the MATLAB file "**roe_flux.m**". This function intakes the state on two bordering elements (left and right), as well as the normal vector pointing from the left element to the right. The overall output of this function is the "Roe Flux" over the edge of the elements. The formula for it can be seen below.

$$\hat{\mathbf{F}} = \frac{1}{2}(\mathbf{F}_L + \mathbf{F}_R) - \frac{1}{2}\begin{bmatrix} |\lambda|_3 \Delta\rho & +C_1 & \\ |\lambda|_3 \Delta(\rho\vec{v}) & +C_1\vec{v} & +C_2\vec{n} \\ |\lambda|_3 \Delta(\rho E) & +C_1 H & +C_2 u \end{bmatrix}$$

where

$$C_1 = \frac{G_1}{c^2}(s_1 - |\lambda|_3) + \frac{G_2}{c}s_2, \qquad C_2 = \frac{G_1}{c}s_2 + (s_1 - |\lambda|_3)G_2$$

$$G_1 = (\gamma - 1)\left(\frac{q^2}{2}\Delta\rho - \vec{v}\cdot\Delta(\rho\vec{v}) + \Delta(\rho E)\right), \qquad G_2 = -u\Delta\rho + \Delta(\rho\vec{v})\cdot\vec{n}$$

$$s_1 = \frac{1}{2}(|\lambda|_1 + |\lambda|_2), \qquad s_2 = \frac{1}{2}(|\lambda|_1 - |\lambda|_2)$$

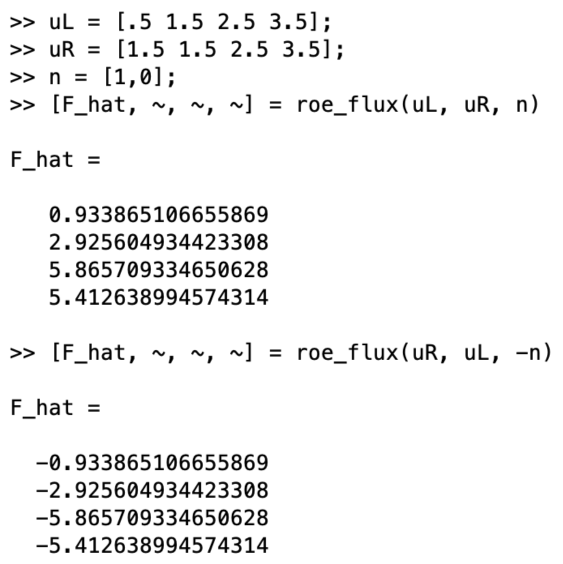
$$\Delta\mathbf{u} = \mathbf{u}_R - \mathbf{u}_L, \qquad q^2 = u^2 + v^2$$

The function works by first calculating the state-flux $\boldsymbol{F_{L,R}}$ of each state. Next it finds the "Roe-Averaged State" – a combination of the left and right states. Using the "Roe-Averaged State", characteristics of the state are backed out, such as the speed of sound, velocity, and enthalpy. Using these characteristics, the coefficients $\mathbf{C_1}, \mathbf{C_2}$ can be computed, and thus, so can the full expression of $\boldsymbol{\widehat{F}}$. As a side note, an entropy fix was implemented within the Roe Flux function to protect against two elements with too similar of speeds. The fix is described with the expression below.

$$\text{if } |\lambda|_i < \epsilon \text{ then } \lambda_i = \frac{\epsilon^2 + \lambda_i^2}{2\epsilon}, \quad \text{for all } i \in [1,4],$$

To check that the function was implemented correctly, test cases were passed into the Roe Flux function.

1. $\boldsymbol{F}(u_L, u_R, \vec{n}) = -\boldsymbol{F}(u_R, u_L, -\vec{n})$
   a. This check determines whether or not that flipping the direction results in the opposite flux value. To test this, each case was run, and the output of the Roe Flux function was compared. The output, as seen below, passes this test



```
>> uL = [.5 1.5 2.5 3.5];
>> uR = [1.5 1.5 2.5 3.5];
>> n = [1,0];
>> [F_hat, ~, ~, ~] = roe_flux(uL, uR, n)

F_hat =

   0.933865106655869
   2.925604934423308
   5.865709334650628
   5.412638994574314

>> [F_hat, ~, ~, ~] = roe_flux(uR, uL, -n)

F_hat =

  -0.933865106655869
  -2.925604934423308
  -5.865709334650628
  -5.412638994574314
```

2. $\boldsymbol{F}(u_L, u_L, \vec{n}) = \vec{\boldsymbol{F}}(u_L) \cdot \vec{n}$
   a. This test checks the consistency of the Roe Flux. To test this, a test state was passed into the flux function, and the expression above was evaluated. The output calculations, as seen below, passes the test. The variable FL_dotted is computed as $\vec{\boldsymbol{F}}(u_L) \cdot \vec{n}$

```
>> uL = [.5 1.5 2.5 3.5];
>> n = [1,0];
>> [F_hat, ~, FL_dotted, ~] = roe_flux(uL, uL, n);
>> F_hat

F_hat =

   1.500000000000000
   2.500000000000000
   7.500000000000000
   4.500000000000002

>> FL_dotted

FL_dotted =

   1.500000000000000
   2.500000000000000
   7.500000000000000
   4.500000000000002
```

3. States with a supersonic normal velocity should return the analytical flux $\vec{\boldsymbol{F}}(u)$ from the upwind state.
    a. This test ensures that within a supersonic flow, information does not travel upstream. To test this, supersonic inlet condition was passed as both an upwind and downwind state. As seen below, the final output flux is equal to the analytical flux of the upwind state, conforming that this condition is satisfied.

```
K>> uL = [1, 2.3.*cosd(1), 2.3.*sind(1), (1./(.4.*1.4)) + (2.3^2)./2]

uL =

   1.000000000000000   2.299649698859700   0.040140534805752   4.430714285714285

K>> uR = [1, 2.*cosd(2), 2.*sind(2), (1./(.4.*1.4)) + (2^2)./2]

uR =

   1.000000000000000   1.998781654038192   0.069798993405002   3.785714285714286

K>> n = [1,0];
K>> [F_hat, ~, FL_dotted, FR_dotted] = roe_flux(uL, uR, n);
K>> F_hat

F_hat =

   2.299649698859700
   6.002674451751222
   0.092309168778115
  11.831697700633153

K>> FL_dotted

FL_dotted =

   2.299649698859700
   6.002674451751222
   0.092309168778115
  11.831697700633153
```

5.2 Question 2 – Finite Volume Solver

To solve the flow over the given domain, the MATLAB script "**p2_FVM.m**" was created to apply a finite volume method. The output of the script is a final array of the state values of the mesh in each element. An overview of the steps the script follows can be found below

1. A mesh file (specified as either the given mesh or a user-created restart file) is read.
    a. If the mesh is the given mesh (mesh0.gri), freestream conditions are initialized.
    b. Else, the conditions from the previous run are used to initialize the mesh.
    c. Call "edgehash.m" to create the IE and BE arrays.
2. The boundary edges, BE, are looped over
    a. The MATLAB version of the mesh reader does not provide the boundary type in BE. For each boundary edge in BE, the corresponding edge nodes are found within mesh.B.
    b. A vector of size [Length of BE, 1] is created with each entry being a scaler number 1-4. Respectively, these numbers correspond to the boundary type: Engine, Exit, Outflow, Inflow.
    c. This vector is tacked on to BE. This allows for the BE array to contain information regarding the boundary type of each boundary edge.
3. Basic variables for the time-stepping section are initialized. These include the residuals, ATPR values, and residual tolerance.
4. TIMESTEPPING SECTION - While the residual norm is greater than the tolerance,
    a. Loop over the interior edges, IE
        i. Define the normal as pointing from elem1 to elem2 in IE
        ii. Calculate the Roe Flux across the edge and add it to that elements residual state vector
    b. Loop over the boundary edges, BE
        i. Define the normal as pointing out from elem1 in BE
        ii. Determine the boundary state (Engine, exit, outflow, inflow) using the added information in BE and calculate the corresponding Roe Flux.
    c. Update the state at each element and calculate the ATPR
5. Plot mesh and field plots
6. Save a restart file that contains the mesh struct, as well as the new state u.

To assist this script, the following functions were created: "**post_process.m**", "**p_t.m**", "**freestream.m**". Respectively, these serve the purpose of plotting the final fields of Mach and total pressure, determining the total pressure within an element, and returning the constant freestream condition, when called.

5.3 Question 3 Running the Finite Volume Solver

The script "**p2_FVM.m**", as described in the previous section, was run at an angle of attack of $\boldsymbol{\alpha = 1}°$. The initial condition that was specified for all elements was the freestream condition. At the end of the simulation, the outputs of "**p2_FVM.m**" included a convergence plot of the simulation's residuals, a converge plot of the calculated ATPR, and two fields plots of the Mach number and total pressure. These plots can be seen below.

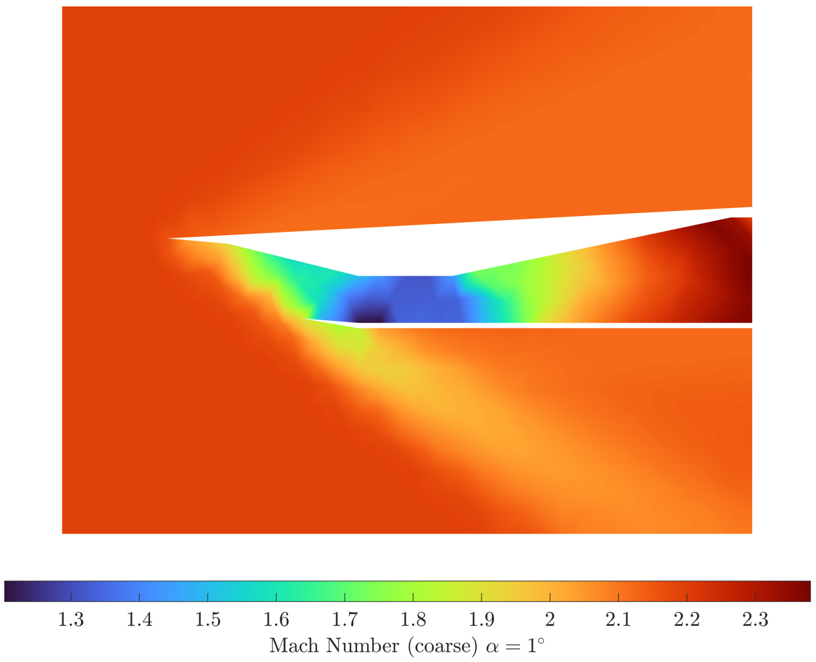


Figure 4 shows the Mach number field plot of the coarse mesh for $\alpha = 1°$.

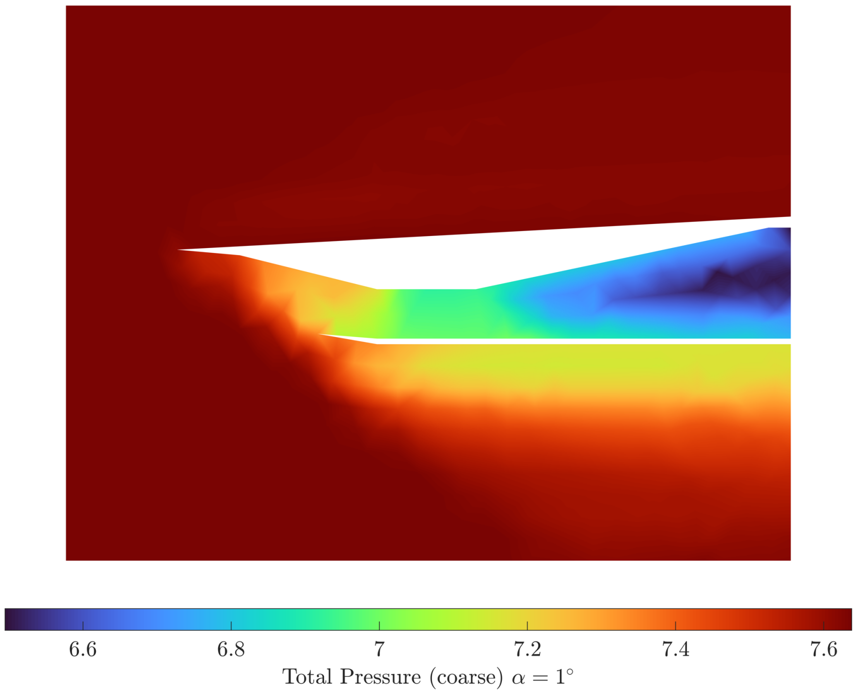


Figure 5 shows the Total Pressure field plot of the coarse mesh for $\alpha = 1°$.

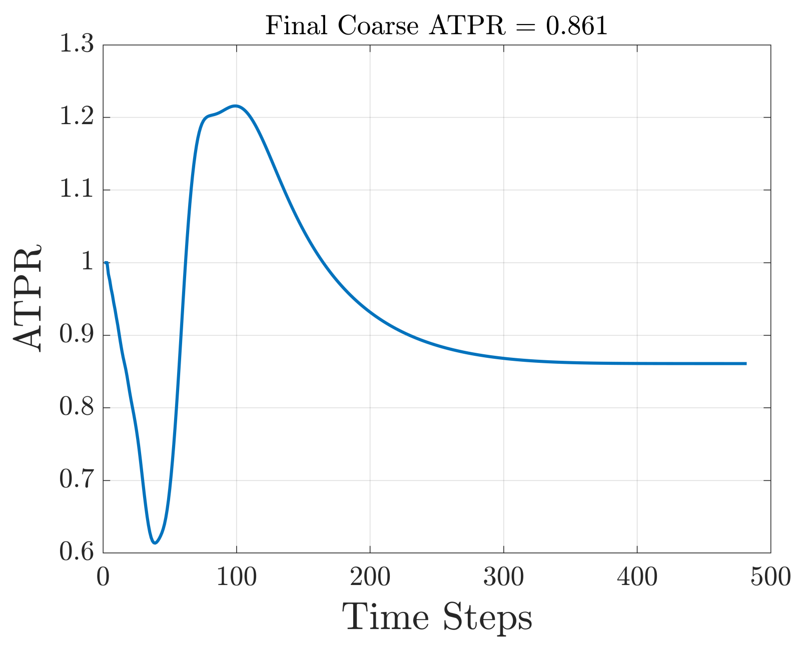


Figure 6 shows the ATPR convergence plot of the coarse mesh for $\alpha = 1°$.

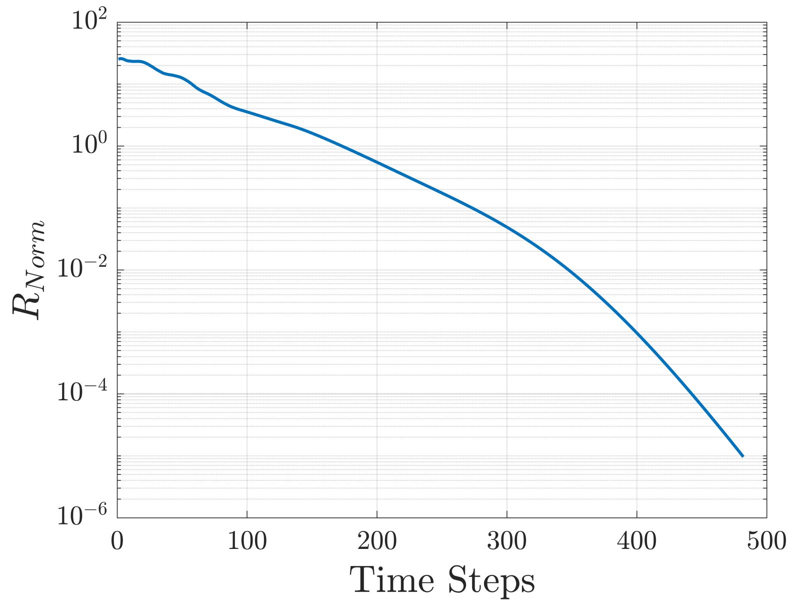


Figure 7 shows the residual convergence plot of the coarse mesh for $\alpha = 1°$.

As seen above, the final field plots have considerable tessellations due the coarseness of the mesh. This should disappear as the mesh is further refined.

5.4 Question 4

Next, a mesh adaptation algorithm was implemented in the script "**adapt.m**". The is script has the effect of refining certain elements within the mesh based off of the change in Mach number jumps, as described in Section 4. A brief summary of the script can be seen below.

1. Read in the restart file from any previous runs of "p2_FVM.m" calls. This will provide the adaption algorithm with a working mesh.
2. Create a variable "edge_map". This is a [Number of elements x 6] sized array. Each row corresponds to an element. The first three rows correspond to the edge numbers that are associated with that element. The next three rows are Booleans that state weather or not the edge in question is flagged for refinement.
3. Loop over the internal edges IE
    a. Assign the edge number to an element on "edge_map" and calculate the error across the edge.
4. Loop over the boundary edges BE
    a. Assign the edge number to an element on "edge_map" and calculate the error across the edge.
5. Knowing the error across each edge, flag the top 3% of the errors for refinement in accordance with Section 4.
6. Loop over the rows of edge_map (loop over the elements)
    a. For every edge on each element, record the number of edges that need to be refined.
        i. If one edge needs to be refined, this element will be split into two separate elements. Modify mesh.Elem to incorporate the new node triplets, and mesh.B.nodes to contain the new coordinates.
        ii. If two edges need to be refined, this element will be split into three separate elements. Modify mesh.Elem to incorporate the new node triplets, and mesh.B.nodes to contain the new coordinates.
        iii. If all three edges needs to be refined, this element will be split into four separate elements. Modify mesh.Elem to incorporate the new node triplets, and mesh.B.nodes to contain the new coordinates.

    b. Assign the old elemental state to the newly created elements
7. Overwrite the restart file for future use in "p2_FVM.m"

With the mesh adaptation algorithm in place, it was applied to the previous coarse simulation at $\boldsymbol{\alpha = 1°}$ five times. During this sequence, plots of the adapted meshes were created. These can be seen below in **Figures 8-13**.

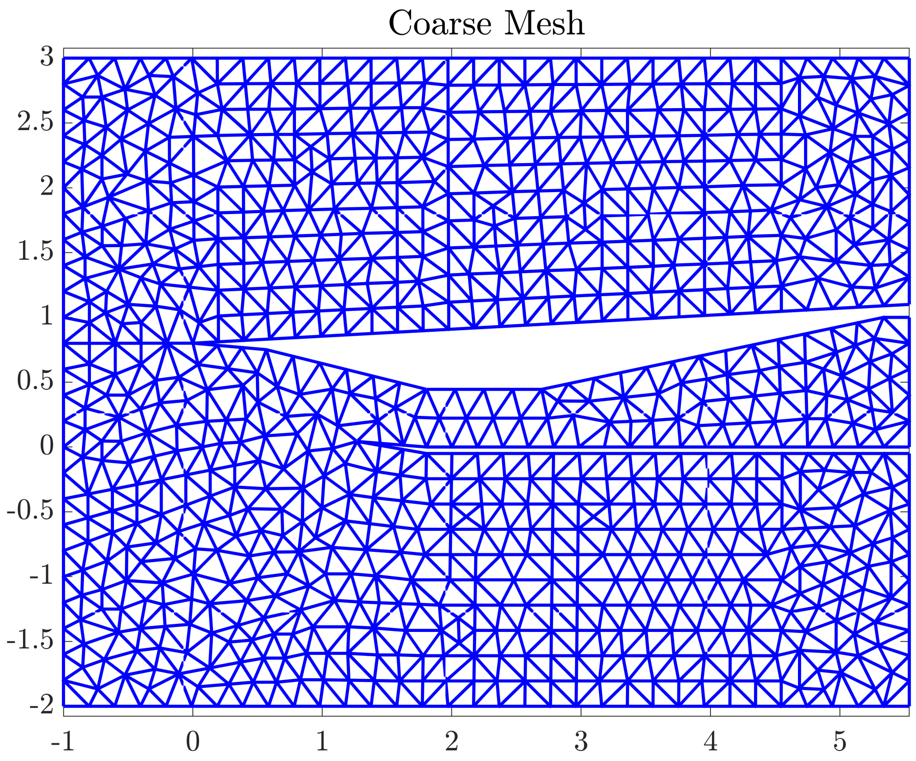


Figure 8 shows the original coarse mesh for $\alpha = 1°$.

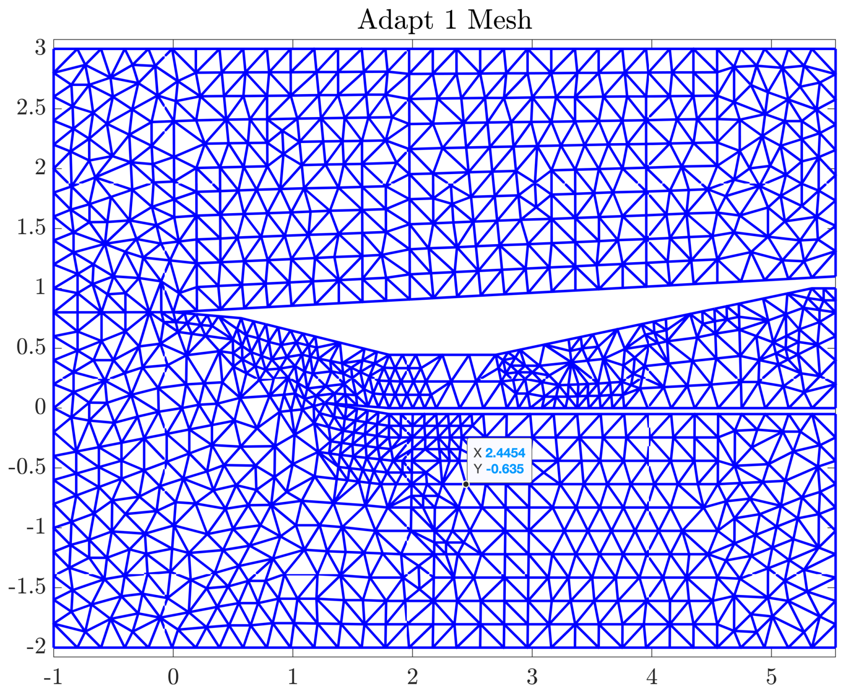


Figure 9 shows the first adaptive mesh for $\alpha = 1°$.

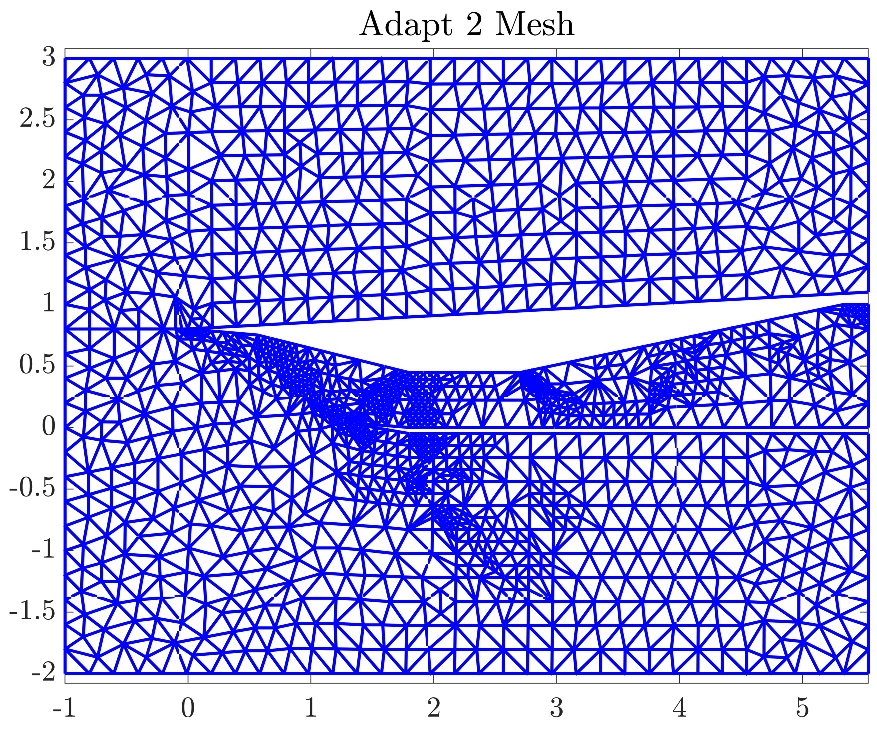


Figure 10 shows the second adaptive mesh for $\alpha = 1°$.

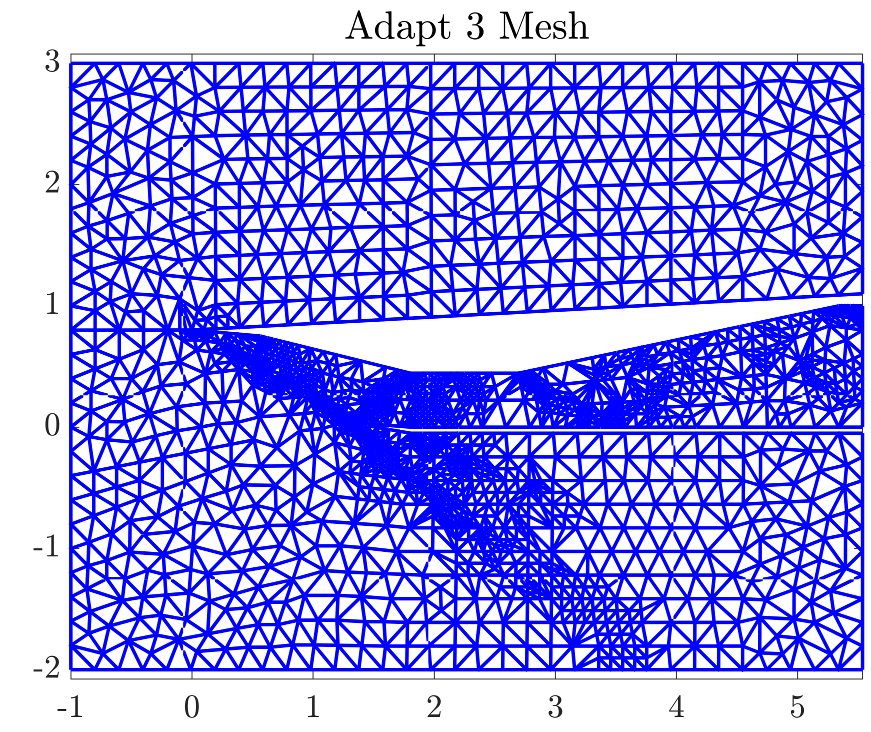


Figure 11 shows the third adaptive mesh for $\alpha = 1°$.

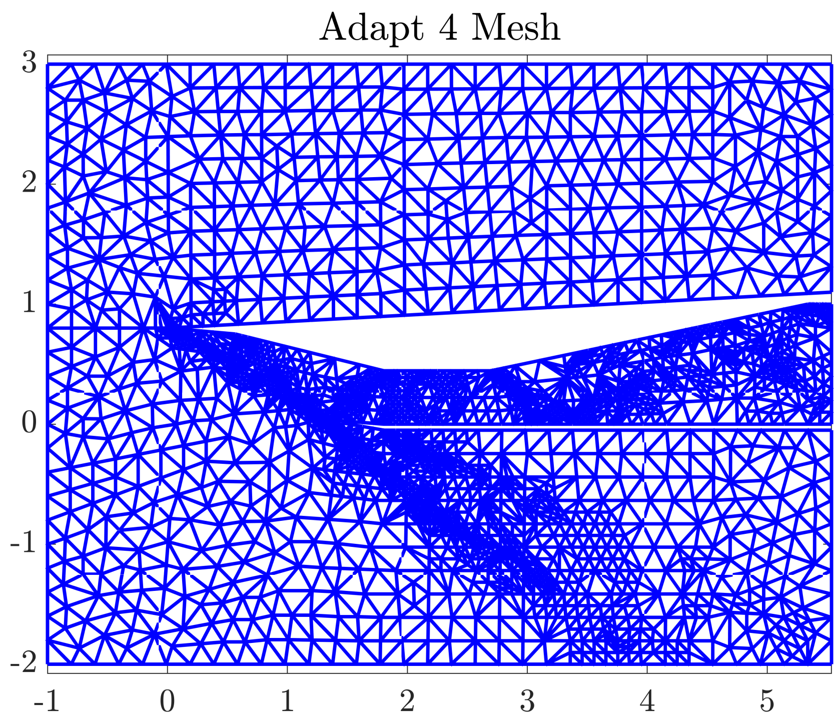


Figure 12 shows the fourth adaptive mesh for $\alpha = 1°$.

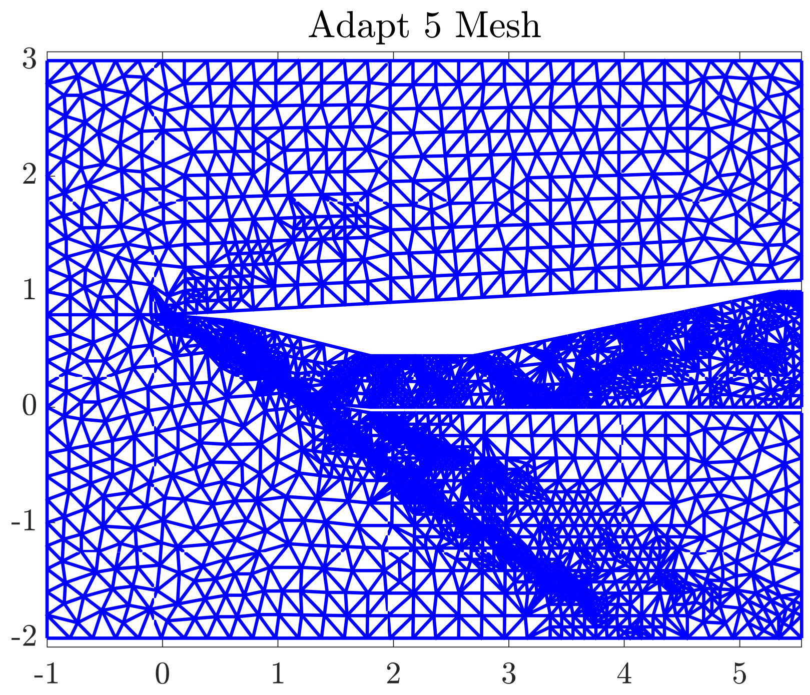


Figure 13 shows the fifth adaptive mesh for $\alpha = 1°$.

A plot of the convergence of the ATPR value was also calculated. This is seen below in **Figure 14**.

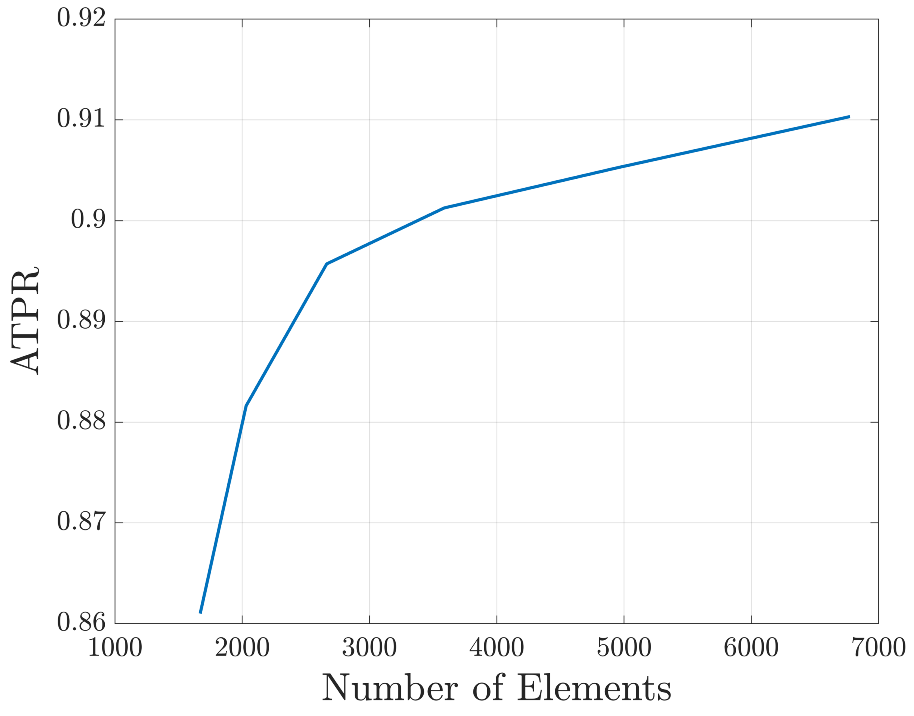


Figure 14 shows the convergence of ATPR against time step iterations for $\alpha = 1°$.

As seen above, as the number of elements increased through more adaptive iterations, the ATPR of the engine outlet converged upon a value nearing 0.91. Note that the axes of **Figure 14** above are not equal, therefore it looks as if the behavior of the ATPR against the number of elements is steeper than in reality. Additionally, field plots of Mach number, Total Pressure, and ATPR values were recorded. Plots of these values can be seen below in **Figures 15-16.**

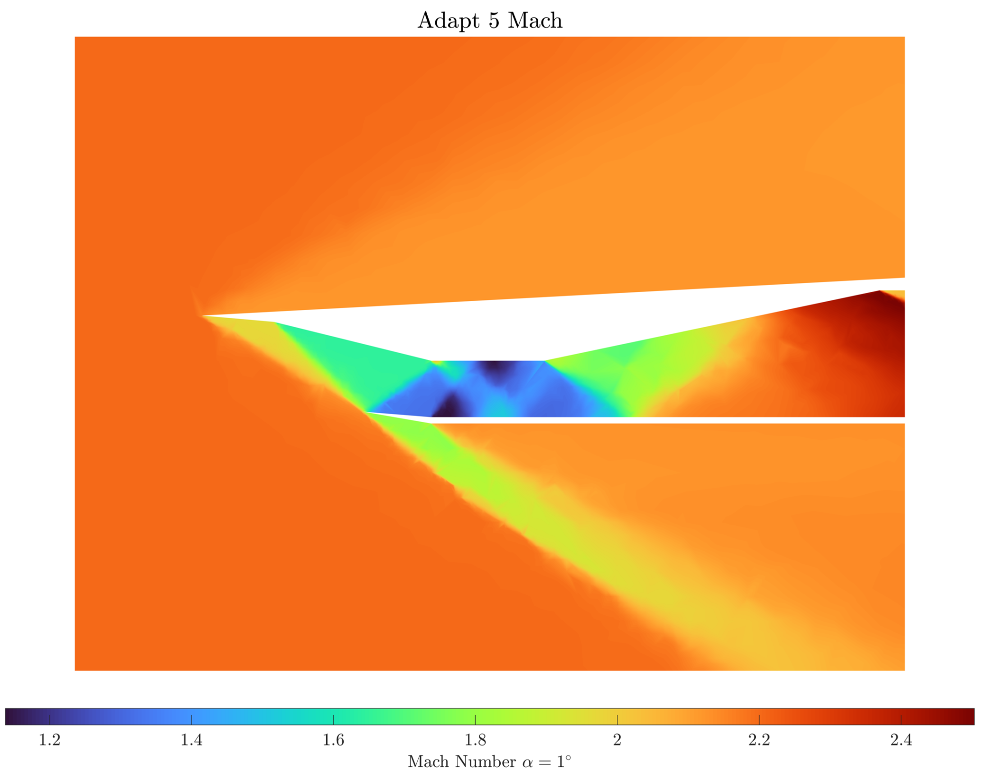


Figure 15 shows the finest Mach field plot for $\alpha = 1°$.

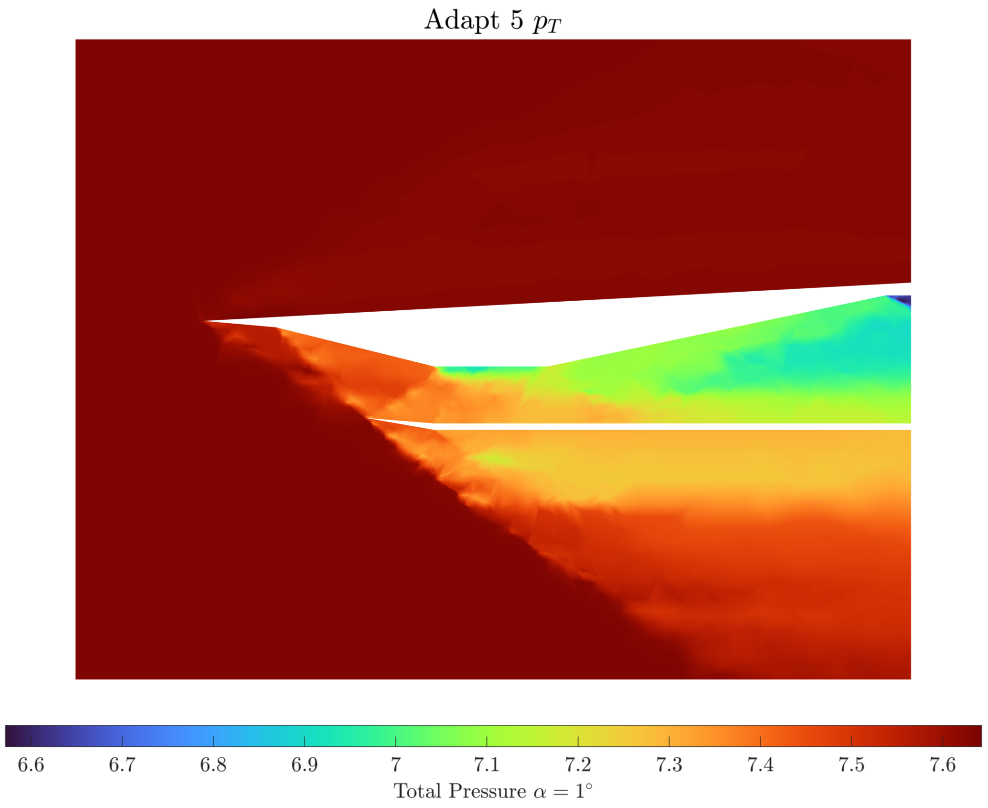


Figure 16 shows the finest Total Pressure field plot for $\alpha = 1°$.

One thing to note is the locations at which the mesh adaptations were taking place. As discussed in Section 4, these locations were determined by calculating the magnitude at which the Mach number changes. In comparing these locations in **Figures 8-13** to the Mach field plots, it's interesting to see that these locations correspond to within the engine inlet – specifically across locations of the shock train and expansion waves. Mostly, this occurs in locations where the geometry has a sharp angle. It makes sense that the value of ATPR is converging with more refined meshes, as the shocks, and their effects on the downstream air, are being better resolved.

5.5 Question 5

Finally, adaptive iterations were run over the mesh for a sweep of alphas, where $\alpha = [0.5, 1, 1.5, 2, 2.5, 3]$ degrees. For each value of alpha, it was chosen to run the adaptive algorithm **six times. Figures 17** shows the final converged value of ATPR for each alpha.

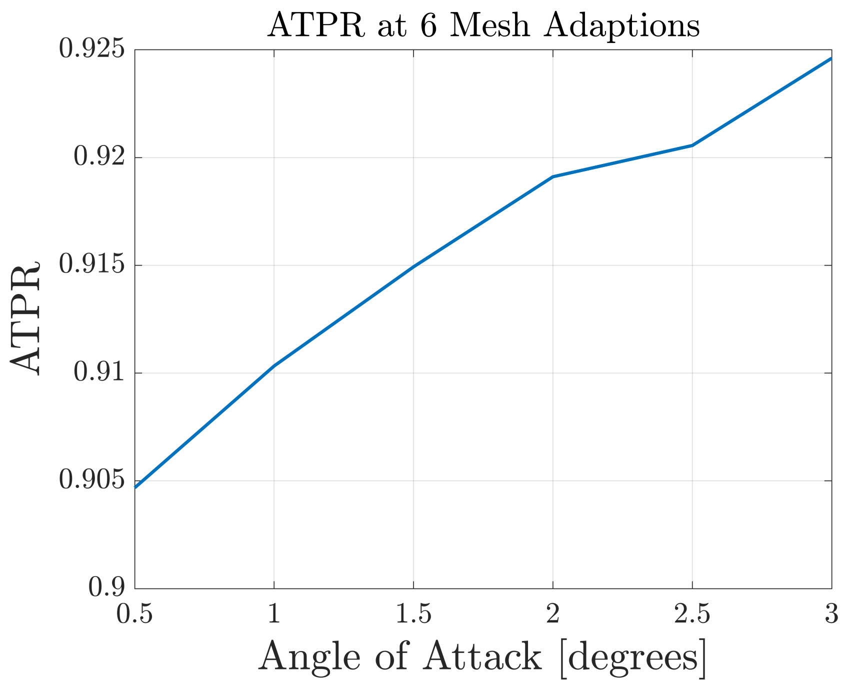


Figure 17 shows the converged ATPR after 6 adaptions for $\alpha = [0.5, 1, 1.5, 2, 2.5, 3°]$

First of all, it is important to note that the last evaluation at AoA = 3 resulted in a ATPR that was higher than expected. While I am not completely sure why this is the case, I do believe it has something do to with the resolution of the mesh – specifically, I belive that at a AoA of 3 degrees, 6 adaptive iterations may not have been enough to properly resolve the shocks and expansion waves. If I had more time, I would like to run each case to 8 adaptive iterations and re-make this plot.

Globally, however, in viewing this plot is that there is a global trend in how the ATPR is increasing with increasing angle of attack. While I'm not sure on the physical reasoning behind why this would increase, I know that a higher value of ATPR is desirable – therefore these results support that if this inlet were to be used within this range of alphas, it should be used at the peak: alphas between 2.5 and 3 degrees. These high AoA's will result in less of a loss of total pressure throughout the inlet.

Additionally, this conclusion is supported by the field plots. **Figures 18 and 19** compare the total pressure field of the inlet against the lowest, and highest, AoA. Both plots were created using a 6-times adapted mesh.

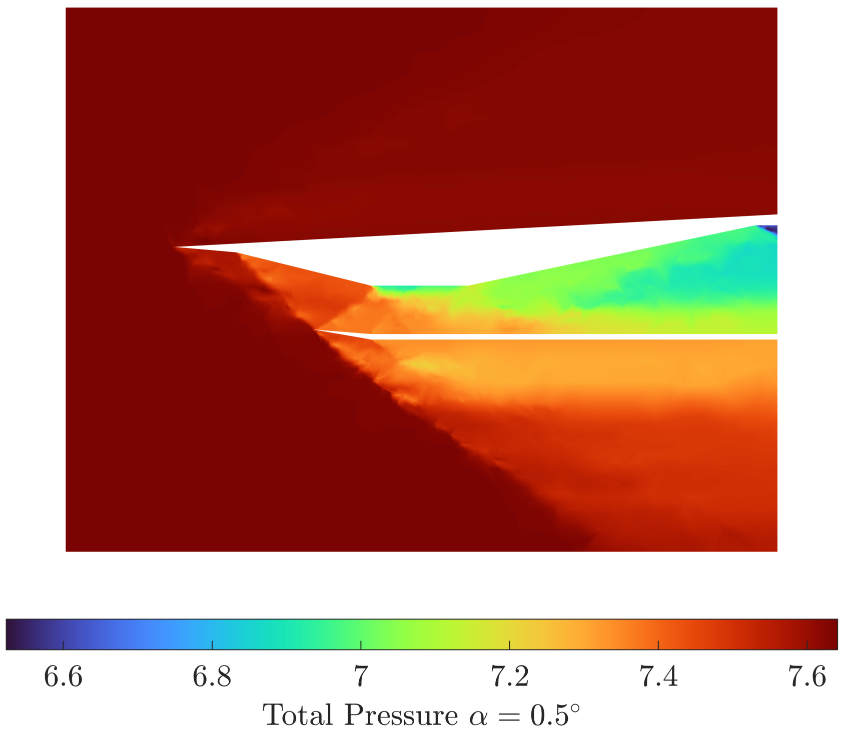


Figure 18 shows the total pressure after 6 adaptions for $\alpha = 0.5°$

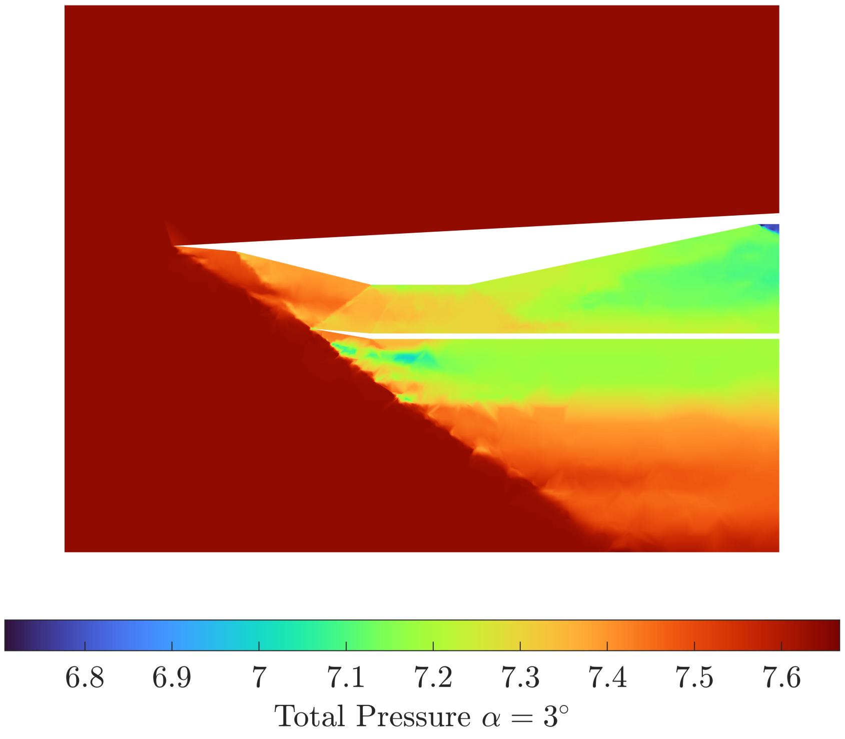


Figure 18 shows the total pressure after 6 adaptions for $\alpha = 3°$

Notably, if you compare these plots, you see that near the engine exit, the second case with AoA = 3 degrees has a much higher total pressure at the boundary compared to the AoA = 0.5 degree case. This supplements the previous claim that the higher AoA resulted in a higher ATPR, since the ATPR is a simple integral of the total pressure across the boundary. Again, although I am unsure of the direct effects, another aspect to consider is that the shocks present in the AoA = 3 degree case are much stronger, as the flow has to turn a greater angle due to the angle of attack.

4. Conclusion

In this project, we implemented a 2D Finite Volume Method (FVM) solver to simulate supersonic flow through an engine inlet. The solver is first-order and utilizes an adaptive approach where the mesh was refined between successive iterations. The 2D compressible Euler Equations served as the governing equations of our state. Specifically, it uses a first-order Euler time-stepping scheme and calculates a Roe Flux between its elements.

Additionally, a mesh adaptation algorithm was implemented that refined locations within the domain that contained large changes in Mach Number. By determining the error across each edge, this algorithm flags certain edges for further refinement – increasing the total number of elements within the mesh.

Implementing these two aspects allowed us to converge upon a reasonably fine mesh and resolve the geometry and locations of the shockwaves present within the body. We see that a shock train develops within, and downstream of the mouth of the engine inlet. Additionally, we see that the total pressure of the flow decreases as it crosses the numerous shocks. Finally, in varying our angle of attack, we found that the Average Total Pressure Recovery across the engine exit increases.