# AE 623 Project 4: FVM Adjoint Based Mesh Adaptation

January 5, 2024

**First-order methods such as FVM are heavily dependent on the quality of the underlying computational mesh. Adjoint-based mesh adaptation is a powerful technique for improving the accuracy and efficiency of numerical simulations. By transferring solution states between coarse and fine spaces, problematic regions within the domain are identified. Deciding which cells are to be refined depends on the targeting strategy. The lift, drag, moment and entropy adjoints were used to adapt the mesh and the targeting strategy chosen was to refine 25% of the total error. The rate of convergence varies based on which adjoint is used to drive refinement as well as the specific targeting strategy. If finely tuned, using a statistical quantity such as the mean to target cells would be the most effective. The quality of the solution has improved compared to primitive adaptation methods and uniform refinement.**

## 1 Introduction

Mesh adaptation is the modification of an existing mesh in order to capture flow features. Deciding which areas of the mesh are refined could be done using the gradient of the state, however a more effective method is using indicators computed using an adjoint. The discrete adjoint is a vector of sensitivities revealing how a perturbation in the residual vector affects an output. The adjoint may be solved using an exact fine-space solution or a higher-order approximation. Once the error indicator is computed, there are diverse targeting strategies that may be employed to ensure that just the right amount of elements are refined. In order to increase the efficiency of repeated adaptive iterations, the existing explicit method may be modified to incorporate implicit time stepping or implement shared-memory multiprocessing through OpenMP.

## 2 Methods

### 2.1 Mesh Adaptation Framework

The Mesh Adaption Framework is shown in **Figure 1**.

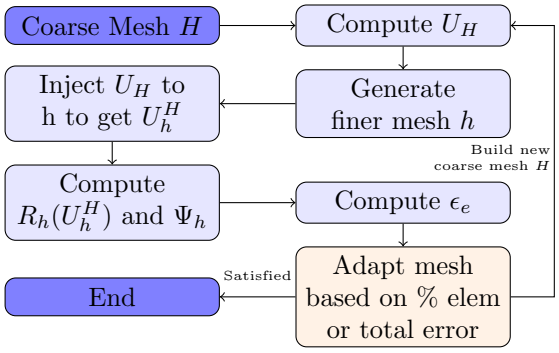


**Figure 1: Mesh Adaptation Framework**

The framework entails a series of steps that are described in detail below:

**1. Coarse State Solution**: The framework begins by computing the converged solution $\mathbf{U}_H$ on the coarse mesh $H$.

**2. Finer Mesh Generation**: A finer mesh $h$ is generated by uniformly refining the coarse space.

**3. State Injection**: The objective of this step is to obtain a fine-space solution $\mathbf{U}_h^H$. Two approaches can be used for this purpose, either using the Gradient theorem to approximate the state for the fine-space solution based on the coarse-space solution, or calculating the exact fine-space solution, which is more accurate but has a higher implementation cost.

**4. Residual Calculation and Adjoint Approximation**: The fine-space residual $\mathbf{R}_h(\mathbf{U}_h^H)$ is computed and then fine-space adjoint, $\Psi_h$, is approximated.

**5. Error Indicator Calculation**: The fine-space residual and fine-space adjoint are utilized to determine the error indicator on each element, which affects. The targeting strategy then identifies the elements that require refinement. If the final solution is unsatisfactory, the whole process is repeated.

### 2.2 Jacobian Matrices

#### 2.2.1 Residual

The goal of constructing the residual Jacobian matrix is to use it to solve the adjoint equation as seen in **Equa-**

**tion 1**.

$$\left(\frac{\partial \mathbf{R}}{\partial \mathbf{U}}\right)^T \Psi + \left(\frac{\partial J}{\partial \mathbf{U}}\right)^T = \mathbf{0} \tag{1}$$

To obtain the adjoint, the residual Jacobian matrix must be constructed for a first-order finite volume method: HLLE flux was used in this case. The construction of this matrix involves several steps. First, the derivative of the HLLE numerical flux with respect to the left and right states must be derived using the chain rule of differentiation. Additionally, the derivative of the flux vectors with respect to the states must be derived. Second, the derivative of the flux must be evaluated for each cell edge by looping through all the edges, multiplying the result by the edge length, and obtaining the residual Jacobian for each element and edge. Finally, the residual Jacobian matrix must be constructed by applying the same method used to calculate the residual, where the residual is added to the cell itself and subtracted from the neighboring cell. The resulting local Jacobian matrices at each cell interface must be assembled into a global matrix to obtain the residual Jacobian matrix. It should be noted that the Residual Jacobian matrix is typically sparse. By following these steps, the residual Jacobian matrix can be constructed and utilized to obtain the adjoint for the first-order finite volume method.

#### 2.2.2 Output

In order to obtain the output Jacobian for Lift and Drag, it is necessary to consider **Equation** 13, which makes use of the derivative of the boundary pressure with respect to the state. The boundary pressure can be expressed as:

$$p^b = (\gamma - 1)\left[\rho E^+ - \frac{1}{2}\rho^+ |\vec{v}^b|^2\right] \tag{2}$$

The boundary pressure can be differentiated with respect to the state as seen in **Equation 3**.

$$\frac{\partial p^b}{\partial \mathbf{u}} = \begin{bmatrix} (\gamma-1)\left[-u_1 \dfrac{\partial |\vec{v}^b|}{\partial u_1}|\vec{v}^b| - \dfrac{1}{2}|\vec{v}^b|^2\right] \\ -u_1(\gamma-1)\left[\dfrac{\partial |\vec{v}^b|}{\partial u_2}|\vec{v}^b|\right] \\ -u_1(\gamma-1)\left[\dfrac{\partial |\vec{v}^b|}{\partial u_3}|\vec{v}^b|\right] \\ (\gamma-1) \end{bmatrix} \tag{3}$$

where the boundary velocity is tangential and can be derived as:

$$\vec{v}^b = \vec{v}^+ - (\vec{v}^+ \cdot \vec{n})\vec{n} \tag{4}$$

The boundary velocity in the $x$ and $y$ directions can be expressed as:

$$v_x^b = \frac{u_2}{u_1} - \left(\frac{u_2}{u_1}n_x + \frac{u_3}{u_1}n_y\right)n_x \tag{5}$$

$$v_y^b = \frac{u_3}{u_1} - \left(\frac{u_2}{u_1}n_x + \frac{u_3}{u_1}n_y\right)n_y \tag{6}$$

Finally, the derivative of boundary pressure with respect to the state can be derived, as seen in **Equations 7** and **8**.

$$\begin{bmatrix} \dfrac{\partial v_x^b}{\partial u_1} \\ \dfrac{\partial v_x^b}{\partial u_2} \\ \dfrac{\partial v_x^b}{\partial u_3} \end{bmatrix} = \begin{bmatrix} \dfrac{(n_x^2-1)u_2 + n_x n_y u_3}{u_1^2} \\ \dfrac{1-n_x^2}{u_1} \\ \dfrac{-n_x n_y}{u_1} \end{bmatrix} \tag{7}$$

$$\begin{bmatrix} \dfrac{\partial v_y^b}{\partial u_1} \\ \dfrac{\partial v_y^b}{\partial u_2} \\ \dfrac{\partial v_y^b}{\partial u_3} \end{bmatrix} = \begin{bmatrix} \dfrac{(n_y^2-1)u_3 + n_x n_y u_2}{u_1^2} \\ \dfrac{-n_x n_y}{u_1} \\ \dfrac{1-n_y^2}{u_1} \end{bmatrix} \tag{8}$$

For calculating the output Jacobian for moment, **Equation 16** is used, which is similar to the equations used for determining lift and drag. The difference lies in determining the distance between the force and the aerodynamic center of the airfoil. In this case, the midpoint of each boundary edge has been used to calculate the distance.

To determine the output Jacobian for entropy, it is necessary to compute the derivative of the entropy as seen in **Equation 17** with respect to the states. The entropy function can be simplified to **Equation 9**.

$$\mathbf{U} = \frac{-u_1\left(\ln\left(\left(u_4 - \dfrac{u_2^2+u_3^2}{2u_1}\right)(\gamma-1)\right) - \gamma\ln(u_1)\right)}{(\gamma-1)} \tag{9}$$

**Equation 9** can be applied to find the derivative of the entropy to construct the output Jacobian for entropy.

### 2.3 Targeting Strategies

There are four main strategies:

**S1. Percentage of Elements:** Strategy 1 involves sorting the error within all cells in descending order and selecting a percentage of the cells with the largest error contribution.

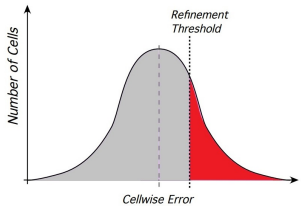


**Figure 2: Error Distribution Featuring Constant Threshold [1]**

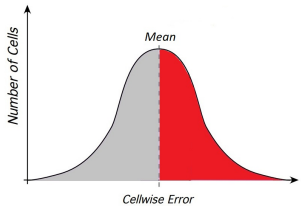


**Figure 3: Error Distribution Featuring Statistical Quantity [1]**

**S2. Percentage of the Error:** Strategy 2 is similar to Strategy 1 but now selects a percentage of the total error. This process is visualized in **Figure 2**.
**S3. Based on a Statistical Quantity:** Strategy 3 utilizes the mean error as a benchmark to determine when an element needs refinement. If an element's error surpasses the mean error, it is flagged for refinement, as seen in **Figure 3**.
**S4. Percentage of the Error with a Changing Threshold:** Strategy 4 is a variation of Strategy 2, but the selection threshold varies, with each iteration; see **Figure 4**.

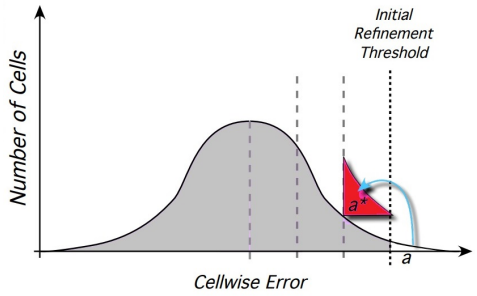


**Figure 4: Error Distribution Featuring Decreasing Threshold [1]**

## 2.4 Adjoint Solution: Fine Space Computation vs. Gradient Theorem

There exist two distinct methods for realizing state approximation from a coarse space to a fine space. The first entails acquiring the adjoint by solving the fine space solution, a process which results in an increased cost of achieving convergence in the fine space. The second approach involves the utilization of the gradient theorem, whose aim is to effectuate the injection of the coarse state solution into the fine space. This approach can avoid solving the state for fine space which is expensive.

$$\int_A \nabla u dA = \int_{\partial A} u \vec{n} dl \tag{10}$$

From **Equation 10** the gradient for each element can be obtained as follows:

$$\vec{L}_0 = \sum_{k=1}^{3} \frac{u_0 + u_k}{2} \vec{n}_{0k} \Delta l_{0k} \tag{11}$$

Next, the states can be interpolated to the centroid of the fine space adjacent element by using the following equation:

$$u(\vec{x}) = u_0 + (\vec{x} - \vec{x}_0) \cdot \vec{L}_0 \tag{12}$$

## 2.5 Indicators

The most common engineering quantities for the simulation of the airfoil are: the coefficient of lift ($c_l$), and the coefficient of drag ($c_d$). Additionally, entropy, moment, and residual were considered for use as indicators for mesh adaptation.
**1. Coefficient of Lift ($c_l$)**: The coefficient of lift can be calculated by computing the force coefficients and the pressure distribution on the airfoil surface. The two-dimensional force on the airfoil is:

$$\vec{F} = \int_{\text{airfoil}} p^b \vec{n} dl \tag{13}$$

The corresponding coefficients can be obtained by non-dimensionalizing the force components, which is given by **Equation 14**.

$$c_l = \frac{F_x^{'}}{0.5 \rho_\infty \left|\vec{v}_\infty\right|^2 c} \tag{14}$$

The output Jacobian for $c_l$ can be computed using **Equation 7**.
**2. Coefficient of Drag ($c_d$)**: Similarly, the coefficient of drag can be written as **Equation 15**, and the output Jacobian for $c_d$ can be computed using **Equation 8**.

$$c_d = \frac{F_y^{'}}{0.5 \rho_\infty \left|\vec{v}_\infty\right|^2 c} \tag{15}$$

**3. Coefficient of Moment ($c_M$)**: The moment of the airfoil can then be calculated by multiplying the force by the distance between the force and the aerodynamic center of the airfoil, $x_c, y_c$. The equation for $c_M$ can be derived as follows:

$$c_M = \frac{F_x^{'} y_c - F_y^{'} x_c}{0.5 \rho_\infty \left|\vec{v}_\infty\right|^2 c^2} \tag{16}$$

**4. Entropy**: The entropy function can be derived as:

$$U = -\rho \frac{(\ln(p) - \gamma \ln(\rho))}{(\gamma - 1)} \tag{17}$$

Differentiating with respect to the state results in the entropy Jacobian vector given by **Equation 9**, which is used to compute the output Jacobian.

**5. Residual**: The residual indicator is not based upon any engineering quantity, and is solely based upon the residual of the coarse-solution injected into the fine-space. Hence, the adjoint vector is just a vector of ones. Therefore, error equation is given by **Equation 18**.

$$\epsilon_e \equiv \left|\mathbf{R}_{he}(\mathbf{U}_h^H)\right| \tag{18}$$

# 3 Results

For all convergence studies, a `c4.gri` solution is used as the *exact* solution; see **Figure 5**. Roe flux is used consistently throughout the report with the exception of HLLE flux being used to confirm the analytically solved discrete adjoint.

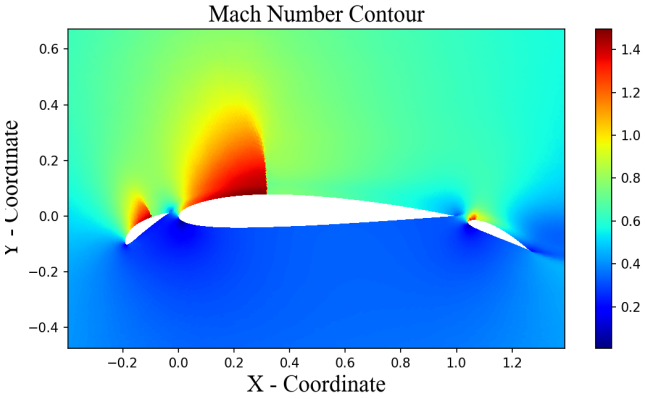


**Figure 5: *Exact* Solution Containing ≈ 500,000 elements**

## 3.1 Adjoint

The analytical solution of the discrete adjoint computed using the HLLE flux function with the converged coarse mesh state is presented in **Figure 6**.

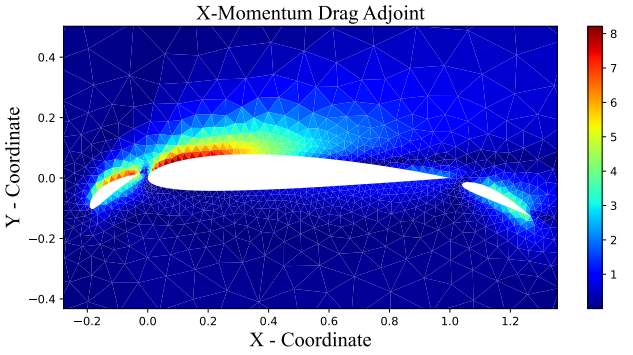


**Figure 6: Analytical HLLE Adjoint Solution**

By comparison, the discrete adjoint computed using the automatic differentiation package (ADOL-C) is shown in **Figure 7**.

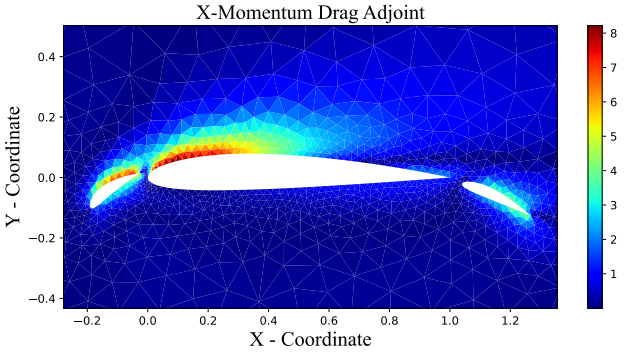


**Figure 7: Automatic Differentiation HLLE Adjoint Solution**

Although the HLLE analytical solution entries differ slightly to the numerical one, by inspection one can observe that they are in agreement. By virtue of this, the Roe flux was implemented using automatic differentiation and used throughout the remainder of the project; see **Figure 8**.

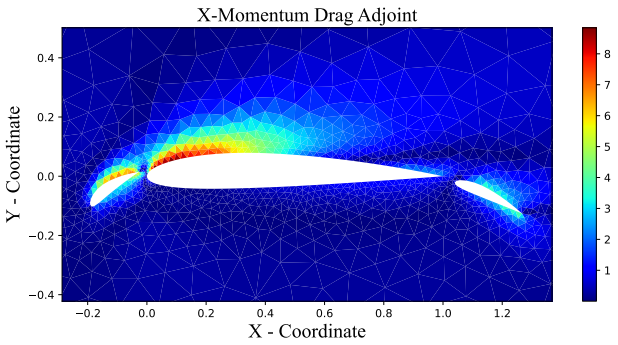


**Figure 8: Automatic Differentiation Roe Adjoint Solution**

## 3.2 Fine-Space Adjoint: Approximated vs. Exact

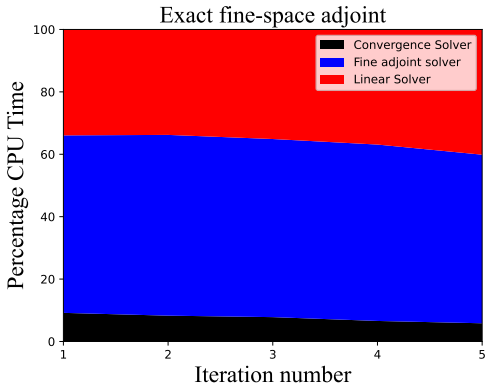


**Figure 9: Exact Fine-Space Adjoint**

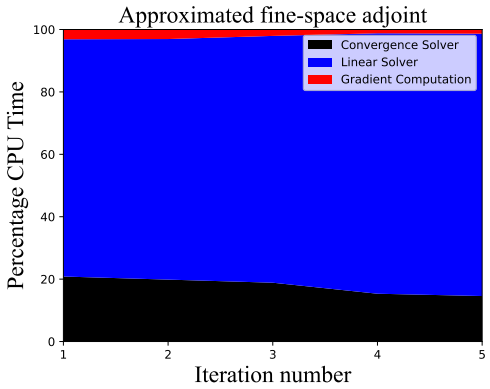


**Figure 10: Approximated Fine-Space Adjoint**

**Figures 9** and **10** compare the use of exact fine-space with an approximated fine-space to derive the Adjoint. **Figure 9** indicates that the majority of the CPU load is allocated to the adjoint solver, with the remaining 35-40 percent of the load devoted to the linear solvers. In contrast, the approximated fine-space approach that employs the Gradient Theorem exhibits significant differences when compared to the exact fine-space approach. The majority of CPU load, approximately 80%, is devoted to the Linear solver since the cost of using fine adjoint solver has been improved by utilizing the Gradient computation. Notably, while the exact fine-space approach requires nearly half of the CPU load to solve the adjoint, the approximated approach allocates less than 5% percent of its CPU load for this purpose.

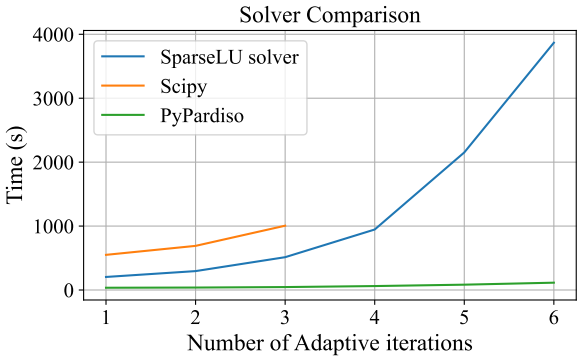


**Figure 11: Solver Comparison**

Based on the results of the three cases comparing the performance of different linear solvers, it is clear that the PyPardiso solver outperforms the other two solvers in terms of CPU time and memory usage. The Scipy solver had the highest CPU time and ran into memory usage issues, which limited the number of adaptations that could be run. In contrast, the PyPardiso solver was able to handle large sparse linear systems of equations with significantly improved CPU time, allowing for more adaptations to be run. It is worth noting that Eigen's SparseLU, which was implemented in C++ in this project, had a relatively large CPU time compared to PyPardiso. However, it was still a viable option for solving the linear system of equations. The reason SparseLU was chosen for this project could be due to its simplicity and ease of use in C++. Also, the reason that the convergence study with respect to CPU time has not been included in this project is due to the significant computational resources required to complete such analysis. Overall, the discovery of the PyPardiso solver at the end of the project is a significant finding, and it will be beneficial for future work. This solver could potentially improve the performance of similar projects and allow for larger and more complex systems to be solved efficiently.

### 3.3 Targeting Strategies

The four different targeting strategies are investigated by adapting w.r.t. the lift coefficient adjoint and seeing which has the most reliable convergence. For Strategy **S1)** 8% of elements are chosen, for **S2)** 25% of the error is adapted, for **S3)** any element error above the mean is refined and lastly for **S4)** the threshold is modeled by

$$\frac{4(\text{total error})}{8+3n}, \tag{19}$$

where $n$ is the adaptive iteration. A convergence study is presented in **Figures 12** and **13** using the lift adjoint and entropy adjoint respectively.

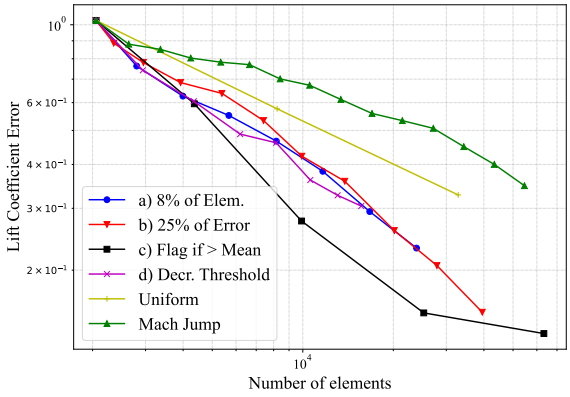


**Figure 12: Lift Coefficient Calculated Using Lift Adjoint**

In **Figure 12** there are multiple observed trends. Initially Strategy **S2** has the steepest convergence rate however it levels out after a few iterations. Strategy **S1** has close to a consistent convergence rate and is even superior to **S2** in these first few iterations. Strategy **S3** has a slightly less steep slope in the beginning but does not level off until the mesh size has grown significantly; it appears to plateau while the other targeting strategies do not. At the end of the adaptive iterations at approximately 40k elements strategy **S2** shows the most promise as it's slope becomes steeper. The decreasing threshold strategy **S4** is keeping up with the other strategies.

Looking at **Figure 13**, when adapting w.r.t. the entropy, many of the trends observed in **Figure 12** are shifted. The similarities being that strategy **S3** is superior to uniform refinement. The most striking difference is that the targeting strategies do not perform well, infact they behave similarly to the Mach number jump adaptation in Project 2. In order to make more significant conclusions about the indicator one would need to evaluate a far more targeting strategies. It becomes clear that different adjoint weighted residuals produce different outcomes depending on the targeting strategy.

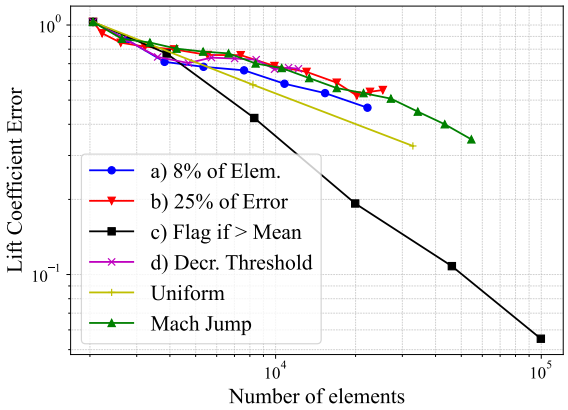


**Figure 13: Lift Coefficient Calculated Using Entropy Adjoint**

### 3.4 Indicators

**Figure 14** displays the adjoint contour plot for the $c_l$ indicator. It is apparent from the plot that the elements on the flap exhibit a higher sensitivity to the coefficient of lift, which is reasonable given that the flap is an aerodynamic device that generates lift. Furthermore, there is a significant sensitivity observed in the wake region, which is crucial for accurate computation of the coefficient of lift.

In contrast, as illustrated in **Figure 14**, the $c_d$ indicator does not seem to place much emphasis on the wake region. Instead, it appears to prioritize the trailing edge and the airfoil wall. The $c_l$ indicator seems to overlook the elements at the surface, on the other hand, the $c_d$ indicator places a higher emphasis on it which could account for the higher convergence rate of the $c_d$ indicator, as shown in **Figure 25**.

The moment indicator, on the other hand, exhibits minimal sensitivity near the surface wall of the airfoil, focusing instead on the leading and trailing edges. Moreover, it displays a higher magnitude than the aforementioned indicators. This higher concentration of sensitivity may explain why the moment indicator performs exceptionally well, as the targeting strategy is based on the sum of errors, which can lead to adapting cells with the highest error.

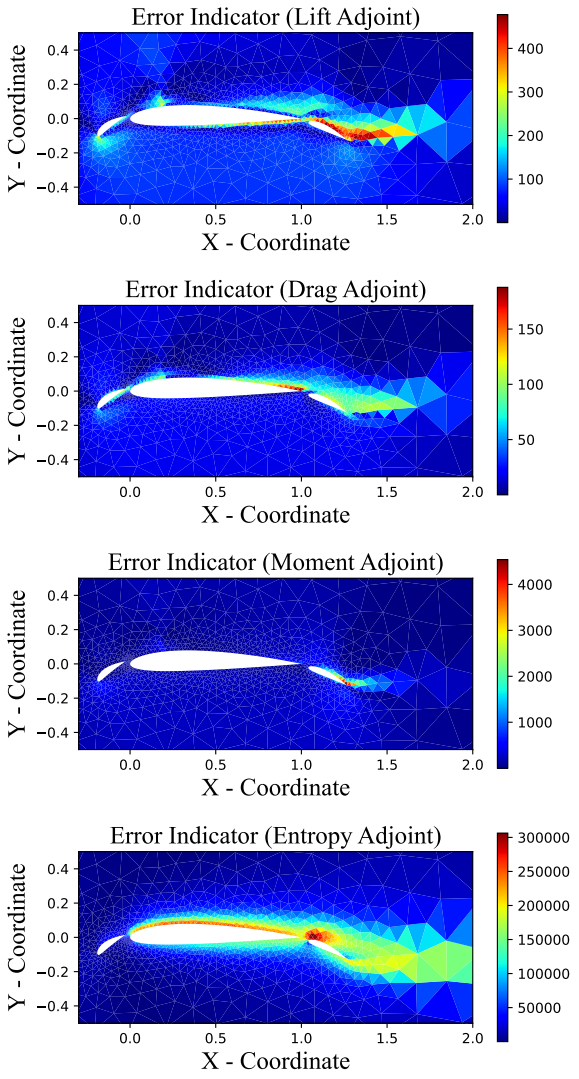


**Figure 14: Error Indicator $\epsilon_e = \left|\Psi_{h,e} R_{h,e}\left(U_h^H\right)\right|$ With Different Adjoints**

**Figure 14** demonstrates that the entropy indicator exhibits high sensitivity for all elements in close proximity to the airfoil elements, particularly the leading edge of the flap. Additionally, it has the highest sensitivity magnitude. This observation may explain why a targeting strategy with 25% of the sum of errors performs poorly, whereas 50% performs similarly to other indicators. A targeting strategy with 25% will only focus on the elements around the flap during every adaptive iteration.

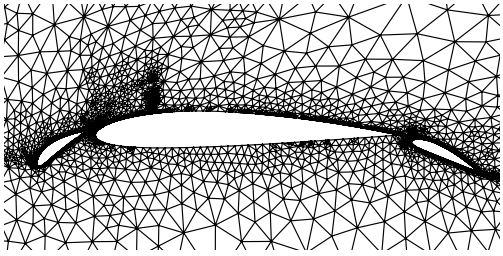

**Figure 15: Mesh Adaptation Using Mach Number Jump as Indicator with 27158 Elements**

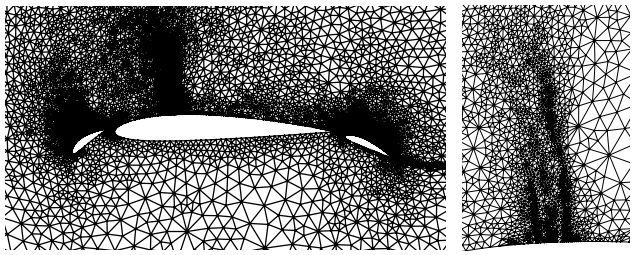

**Figure 16: Mesh After 9 Iterations for Using Lift Adjoint as the Indicator with 27918 Elements**

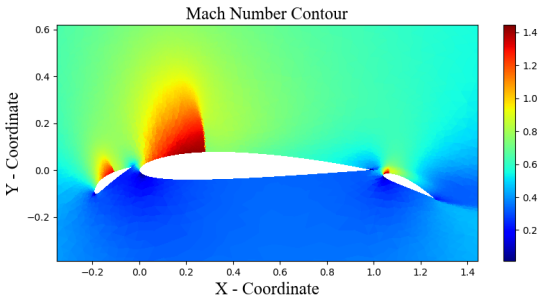


**Figure 17: Mach Plot After 9 Iterations Using Lift Adjoint as the Indicators**

**Figure 16** shows the mesh after 9 adaptive iterations. It shows that the lift adjoint, in conjunction with our targeting strategy, effectively targets the elements around the airfoil, particularly the shock on the upper surface. As a result of this mesh adaption, we can observe the shock captured, as illustrated in **Figure 17**. As a majority of the mesh has been refined around the airfoil surface, the lift and drag coefficients have been significantly improved over the course of the adaptive iterations.

Upon comparing with **Figure 15**, which employs the jump in Mach number as an indicator, it is evident that following 9 adaptive iterations, the total number

elements are at approximately 27-28k. The higher concentration of refined elements around the upper surface airfoil for $c_l$-based, as opposed to the Mach number jump-based. Consequently, the rate of decline in lift and drag coefficient error for $c_l$-based significantly outperforms that of gradient of the state-based methods.

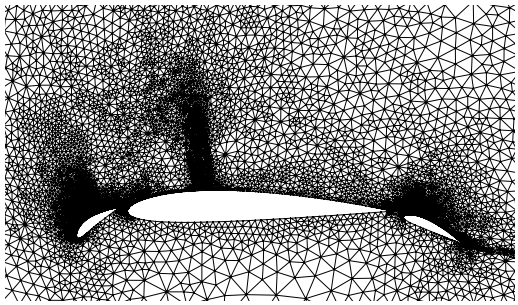

**Figure 18: Mesh After 9 Iteration Using Drag Adjoint as the Indicator (28783 Elements)**

The Mach plot for the $c_d$ as an error indicator produces the same results as the Mach plot using $c_l$ as seen in **Figure 17** qualitatively. However, the error of $c_l$ is higher in the case of $c_l$ as an indicator (3.18), whereas for $c_d$ as an indicator (3.25) the error is lower. It could be explained by the fact that **Figure 18** preciely targets the shock and has less smearing.

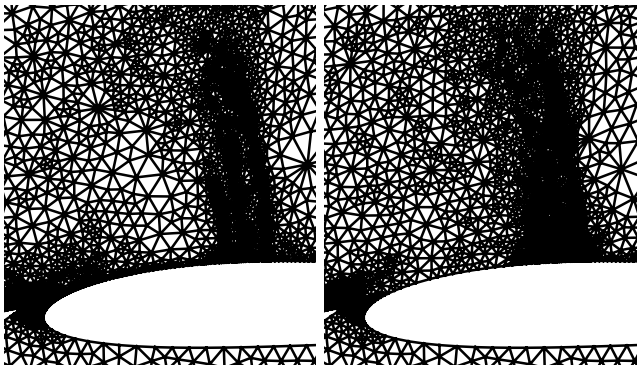

**Figure 19: Comparing Mesh Adaptation After 9 Iteration Using Lift (28783 Elements) and Drag Adjoints Respectively (27918 Elements)**

By comparing the mesh adaptation between the $c_l$ and $c_d$ as an indicator and focusing on the shock region for the main airfoil seen in **Figure 19**, it is evident that the $c_d$ as an indicator is able to target the elements at the wall of the airfoil between the start and the end of the shock. Moreover, the $c_d$ indicator is not targeting many elements inside the shock, whereas the $c_l$ indicator has a higher concentration of elements inside the shock, even though it has about 800 fewer elements than the $c_d$ indicator after the 9th adaptation iteration. Additionally, $c_d$ indicators have more element concentration on the tip and the trailing edge on all the elements as compared to the previous indicator. Therefore, it seems to resolve the shock edge, as trailing and leading edges are critical to better results for the coefficient of lift and drag.

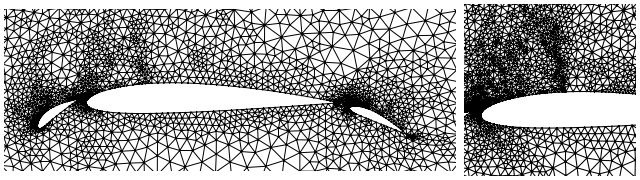

**Figure 20: Mesh After 9 and 11 Iterations Respectively for *Residual* as Indicator (30252, 78080 Elements)**

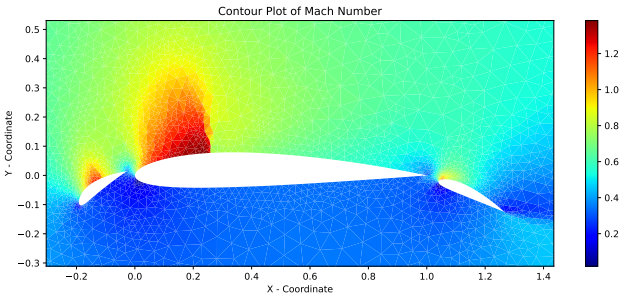


**Figure 21: Mach plot After 9 Iterations Using *Residual* as Indicator**

The residual indicator employs a distinctive targeting strategy whereby it initially focuses primarily on the leading and trailing edges of airfoil elements for the first few adaptive iterations before adjusting the elements near the shock. Comparing adaptive runs 9 and 11 via **Figure 20**, it becomes evident that there is minimal targeting near the shocks, which is confirmed by the resolution of the shock in the Mach plot in **Figure 21**. This may be attributed to the presence of high-magnitude errors on the leading and trailing edges. This assertion may be further supported by examining the progression of element increase between runs 8 and 11, i.e., 18762, 30252, 48484, 78080. The percentage increase in the first few adaptive runs was not substantial, potentially because only a small number of elements contributed to the majority of the error. Thus, as the targeting strategy was predicated on cumulative error, as the error magnitudes began to equalize among all the elements, the increase in the number of elements began to escalate. Moreover, it is observed that once the

residual indicator began to address the elements near the shock edge, the coefficient of lift error decreased even further to a value of $c_l = 3.21$.

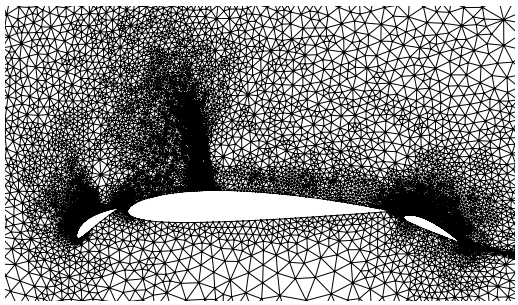

**Figure 22: Mesh After 9 Iteration Using Moment Adjoint as Indicators (30668 Elements)**

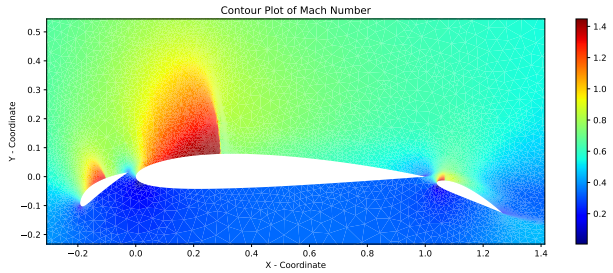


**Figure 23: Mach Plot After 9 Iteration Moment Adjoint as Indicator**

The coefficient of moment indicator performed the best among all the indicators. It overcame all the problems faced in the aforementioned indicators. It targeted the leading and trailing edges, and it manages to adapt the elements at the wall of the airfoil while resolving the shock edges and inside adequately; see **Figures 23** and **24**.

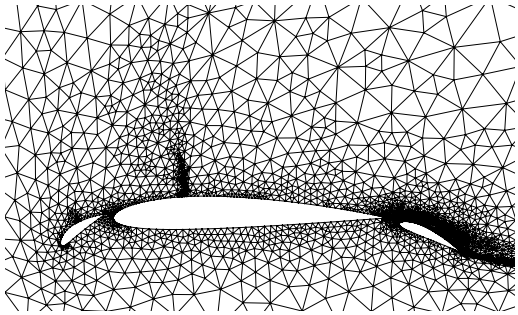

**Figure 24: Mesh After 9 Iterations Using Entropy Adjoint as Indicator (16986 elements)**

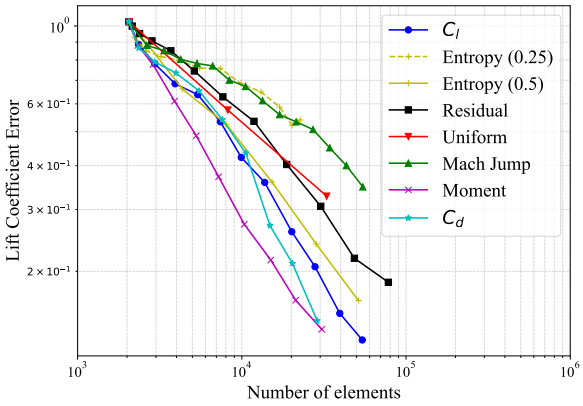


**Figure 25: Comparing Various Indicators for Adaptive Strategy with Lift Coefficient**

**Figure 25** plots the convergence history of the coefficient of lift error against the number of elements. It further strengthens the idea that moment indicator was the best-performing among the other indicators. However, even though $c_d$ indicator convergence was slow at the beginning, it seems to have the highest convergence rate by the end of the 9th adaptive iteration, and may very well outperform the moment indicator. The residual indicator appears to exhibit an equivalent rate of convergence as uniform refinement. While the $c_l$, $c_d$, $c_m$, and entropy indicators demonstrate superiority to uniform refinement, a better convergence rate was expected. This may be attributed to the fact that no indicator specifically targets the front end of the shock and section below the airfoil. Furthermore, with the exception of the entropy indicator, none of the indicators strongly focus on the wake behind the airfoil, which is a critical factor in obtaining precise $c_l$ measurements. However, it is to be noted that entropy was the least successful to resolve the shock.

**Figure 26** plots the convergence history of the coefficient of drag error against the number of elements. The graph demonstrates that the rates of convergence using $c_d$, $c_l$, and $c_m$ are nearly identical, with the residual indicator following closely. With the exception of entropy, all indicators exhibit superior performance compared to uniform mesh refinement. Notably, the indicators exhibit a higher rate of convergence for the coefficient of drag error compared to the coefficient of lift error. This observation may be attributed to the indicator's apparent focus on the leading and trailing edges of airfoil elements, where the magnitude of the force in the y-direction is significant, as opposed to the surfaces of said elements. The lack of targeting near the leading edges of slat and main airfoil may explain the

poor performance of the entropy indicator in terms of $c_d$ convergence. It is possible that the entropy indicator's strategy of targeting elements in proximity to the airfoil is not suitable for accurately capturing the flow phenomena around the leading edges of the slat and main airfoil, which are important for computing $c_d$.

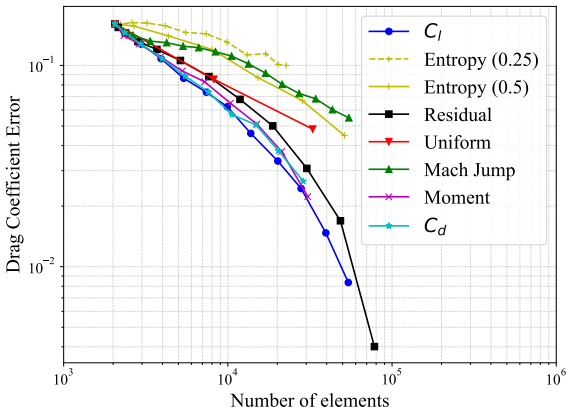


**Figure 26: Comparing Various Indicators for Adaptive Strategy With Drag Coefficient**

# 4 Conclusions

The selected targeting strategy involved refining 25% of the error, as it exhibited reliable convergence for most outputs. However, an analysis of the entropy reveals that refining elements with an error above the mean is a more effective strategy. This underscores the need to adjust the targeting strategy based on the output of interest and the adjoint sensitivity vector.

It is observed that moment and $c_d$ indicators perform well based on our targeting strategy. However, their convergence rates were lower than expected. Notably, the coefficient of drag indicator had a significantly greater convergence rate, suggesting that while all the indicators resolved the leading and trailing edges of the airfoil elements, they were inadequate in resolving the wake and elements near the surface of the airfoil. Furthermore, none of the indicators approached the actual $c_l$ value of 3.39, although the moment adjoint indicator achieved the closest value of 3.26.

Unfortunately, due to the choice of a poor linear solver, it was not feasible to generate a convergence plot over time. However, using a new linear solver and integrating it with the gradient theorem is expected to yield faster convergence times.

# 5 Future Work

## 5.1 Implicit Time Stepping

In terms of future work, since the residual Jacobian matrix (RJM) is already available one may implement an implicit time marching scheme such as Newton's method. The goal is to drive the residual to zero,

$$\underline{R}\left(\underline{u}_0 + \Delta\underline{u}\right) = 0 \tag{20}$$

where

$$\Delta\underline{u} = -\left[\frac{\partial \underline{R}^u}{\partial \underline{u}}\right]^{-1} \underline{R}. \tag{21}$$

The unsteady residual Jacobian matrix (URJM) is defined by:

$$\underbrace{\frac{\partial \underline{R}^u}{\partial \underline{u}}}_{\text{URJM}} = \frac{\underline{\underline{M}}}{\Delta t} + \underbrace{\frac{\partial \underline{R}}{\partial \underline{u}}}_{\text{RJM}}, \tag{22}$$

where $\underline{\underline{M}}$ is the mass matrix consisting of the element's area in the FVM and $\Delta t$ is a scalar time step; this is called pseudo time stepping (PTC).

# References

[1] Marian Nemec, Michael Aftosmis, and Mathias Wintzer. Adjoint-based adaptive mesh refinement for complex geometries. In *46th AIAA Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics, January 2008.