AE 623 Project 3: Discontinuous Galerkin

March 17, 2023

In this project, the Discontinuous Galerkin Method was used along with the implementation of curved meshes, to resolve the flow around an airfoil and a Gaussian bump. The solver had the implementation for $p = 0, 1, 2, 3$, in order to fulfill the convergence study. It was observed that the rate of convergences is proportional to the order of spatial approximation (p). Additionally, functionality for computing the C_l and C_d, and C_p plot for airfoil was implemented. Further, the capability for mesh adaptation was introduced, and the results after several iterations were compared with data obtained from higher order p and finer mesh.

1 Introduction

The primary objective of this project was to extend the finite-volume solver to high-order approximation using the Discontinuous Galerkin (DG) method. This project builds on the previous finite-volume project, using the compressible Euler equations and either Roe or HLLE flux solved using SSP-RK2 as the local time stepping. The DG method uses at minimum quadratic curved elements to better represent the curved boundary of the airfoil.

For **Tasks 1, 2, and 3** airfoil geometry was used. However, for **Tasks 4 and 5**, Gaussian Bump was used as geometry with **inflow condition** of $M = 0.25$ and $\alpha = 1$, and for **outflow condition** was set to $p_b = 0.9 p_t$ both the bump and the upper wall employed inviscid boundary conditions.

Algorithm 1 Quasi-Newton Hessian reset

```
1: while $||\nabla f||_\infty \geq$ tolerance do
2:     compute search direction
3:     perform line search
4:     if $—\nabla f^T p_k| \leq$ 1e-8 then
5:         reset = TRUE
6:     else
7:         reset = FALSE
```

Algorithm 2 Optimization Algorithm Set-up

```
1: Initialize gradient
2: f, $\nabla f$ = func(x0)
3: $\phi$, $\nabla \phi$ = f, $\nabla f$
4: ($\nabla \phi(0)$ can computed by $\nabla \phi \bullet p_k$)
5: while $||\nabla f||_\infty \geq$ tolerance do
6:     compute search direction
7:     $\alpha^*$, $\phi$, $\nabla \phi$ = line search($\alpha^*$, $p_k$)
8:     $||\nabla f||_\infty = ||\nabla \phi||_\infty$
```

2 Methods

2.1 Mesh Curving

Mesh curving is performed by first looping over the elements of the mesh. The code checks whether any element touches a boundary for any of the three elements of the airfoil. If an element does touch a boundary it is then new nodes are generated. The amount of nodes generated depends on the desired geometric order (q). In this project all elements start as linear elements ($q = 1$), then mesh curving is applied to make the boundary elements quadratic ($q = 2$) or cubic ($q = 3$). **Figure 1** shows the structure and numbering convention for the newly generated nodes.

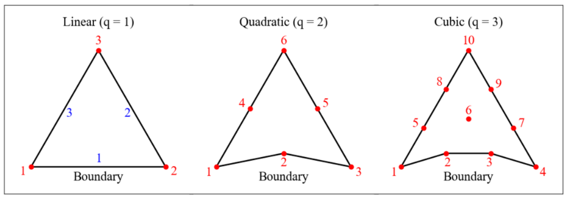

Figure 1: Element Node Arraignment

The new nodes generated on the boundary face are snapped to the spline of the airfoil to produce the curve. On the two faces not adjacent to the boundary the new nodes are generated equidistant in-line with the corner nodes. Additionally in the case of cubic mesh curving, an additional node is generated at the centroid of the element, which is then shifted by the weighted average of the shift of the 2 nodes that snapped to the spline. All the nodes generated for the modified elements are then saved to a new E2N array. An updated array containing the new node coordinates is also generated.

2.2 Discontinuous Galerkin

The solver begins by precomputing the basis function, inverse matrix, and normal for each curve element. Afterwards, the state at each Lagrange point for each element is initialized to the free stream. The time marching is performed using the SSP-RK2 method, which is stable for low CFL values and has faster computation than the RK4 method. The order of the solution on an edge is represented by (p). The Discontinuous Galerkin Algorithm is shown in **Algorithm 3**.

Algorithm 3 DG - Solver

```
1: Initialize free stream condition and order of integration for curve and linear elements.
2: Obtain 2D and 1D quadrature points.
3: Compute basis function for 'quad1d.c' (CW and CWW) and 'quad2d.c'.
4: Compute normals for curve elements and edge Jacobians in reference space.
5: Compute inverse mass matrix for each element.
6: Initialize $\mathbf{U}$ at each Lagrange node.
7: if $p = 0$ then
8:     Initialize state as $\mathbf{U}_\infty$.
9: else
10:    Initialize state from a lower order of $p$.
11: while $|R| > 10^{-5}$ do
12:    Compute residual $(R_0)$ at Lagrange point.
13:    Compute $f_0$ then $\mathbf{U}_1$.
14:    Compute residual using $\mathbf{U}_1$ then $f_1$.
15:    Compute the full step by $f_0$, $f_1$, and dT.
16:    Compute the total residual.
```

2.2.1 Preprocessing

The order of integration for linear elements is given by $(2p + 1)$ for both the edge and interior. However, the curve element's interior uses $(2p + 1) + (2(q - 1))$ quadrature points, and the edge calculation uses $(2p + 1) + (q - 1)$ quadrature points. Depending on the number of quadrature points, coordinates are obtained from 'quad1d.c' and 'quad2d.c' and stored as a vector in the form of: $[x_1, y_1, x_2, y_2, \ldots]$.
Basis Function: Basis functions are precomputed by providing the quadrature point to 'shape()' - 'shapeL()' defined in file shape.c, which outputs the basis function as a vector. This is repeated for each quadrature point, and appended at the end of the previous quadrature point. The resulting vector stores the basis function as: [basis q_1, basis $q_2, \ldots$].
However, when computing for 'quad1d.c', which stores only one value for each coordinate, the other value can be computed as shown in **Algorithm 4**.
Gradient for Basis Function: Gradients are calculated by providing a quadrature point to 'shape()' - 'gradientL()' defined in the file shape.c. This returns the gradients of the Lagrange shape functions for a given order p as a vector. The vector first stores all the x derivatives are stored followed by all the y derivatives. Similar to the basis function vector, each gradient vector is appended at the end of the vector from the previous quadrature point.

Algorithm 4 Quad1D Coordination Generation

```
1: if CW Basis Function then
2:     Input is side and quad1d point $(s)$
3: else if CCW Basis Function then
4:     Input is side and quad1d point $(1 - s)$
5: if side= 1 then
6:     $x_{\text{ref}} = [s, 0]$
7: else if side= 2 then
8:     $x_{\text{ref}} = [1 - s, s]$
9: else if side= 3 then
10:    $x_{\text{ref}} = [0, s]$
```

Jacobian: For both linear and cutve elements the Jacobian can be computed using **Equation 1**:

$$\sum_{i}^{n_q} \vec{x_i} \frac{\partial \phi_i}{\partial \vec{\xi}} \left(\vec{\xi}\right) \tag{1}$$

Algorithm 5 is used to compute the Jacobian for any element, where the inputs are: order of the curved element (q), order of the solution on the curved element (p), the coordinates of the element in global space (x), and the coordinates of the point in reference space (x_{ref}).

Algorithm 5 Jacobian Computation

```
1: Inputs are $q$, $p$, $x$ and $x_{ref}$.
2: Initialize $\mathbf{J}$ matrix with zeroes
3: Compute the gradient using 'gradientL($x_{\text{ref}}, q$)'
4: Compute $p_{\text{rank}} \leftarrow (q + 1) * (q + 2)/2$
5: for $i = 0$ to $(p_{\text{rank}} - 1)$ do
6:     $\mathbf{J}$ += $x[i] \cdot x_{\text{ref}}$
7: return $\mathbf{J}$
```

Normal for curve elements: The normal calculation for the curved elements is shown in **Formula 2**. The values of $\frac{d\xi}{d\sigma}$ and $\frac{d\eta}{d\sigma}$ change depend on the local face number (LFN) in reference space, while $\frac{d\vec{x}}{d\xi}$ is equal to the first column of the jacobian matrix and $\frac{d\vec{x}}{d\eta}$ is equal to the second column of the jacobian matrix.

$$\frac{d\vec{x}_{\text{edge}}}{d\sigma} = \frac{d\vec{x}}{d\xi}\frac{d\xi}{d\sigma} + \frac{d\vec{x}}{d\eta}\frac{d\eta}{d\sigma} \tag{2}$$

$$\text{where} \begin{cases} \frac{d\xi}{d\sigma} = 1, \frac{d\eta}{d\sigma} = 0 \text{ if LFN } = 1 \\ \frac{d\xi}{d\sigma} = -1, \frac{d\eta}{d\sigma} = 1 \text{ if LFN } = 2 \\ \frac{d\xi}{d\sigma} = 0, \frac{d\eta}{d\sigma} = -1 \text{ if LFN } = 3 \end{cases}$$

Then the vector that stores node information for curved elements, E2NC, can be looped through, allowing the normal's for each curved face at all the quadrature points to be computed. The **Edge Jacobian** can then be computed by taking the magnitude of the normal for each curved element and storing it in a vector in accordance with E2NC.

Initializing State at each Lagrange Nodes: The state at each Lagrange point can be calculated using **Equation 3**.

$$u(\vec{x},t) \simeq \sum_{p=1}^{N_e}\sum_{j=1}^{N_p} U_{m,j}(t)\phi_{m,j}(\vec{x}) \tag{3}$$

First, basis functions are computed at each Lagrange node in reference space, similar to how basis functions were computed at each quadrature point. Then **Algorithm 6** is used to update the state for element interiors.

Algorithm 6 Initializing state to $\mathbf{U}_\infty$

```
1: Compute the basis functions at each Lagrange node
2: for $i = 0$ to $(N_q * N_p)$ do
3:   for $y = 0$ to $N_p$ do
4:     Initialize $\mathbf{U}_{\text{state}}$ vector with zeroes
5:     for $j = (y * N_p)$ to $((y * N_p) + N_q)$ do
6:       for $k = 0 : 4$ do
7:         $\mathbf{U}_{\text{state}}[k]$ += $\text{basis}_L[j] \cdot \mathbf{U}_\infty[k]$
8: return $\mathbf{U}_{\text{state}}$
```

2.2.2 Residual

To implement the residual in the Discontinuous Galerkin method, the quadrature point basis functions, the mass matrix, and the normal vector are all precomputed. The residual is calculated on the interiors, interior edges, and boundary edges of the elements. To calculate the contributions from interior elements, we iterate over the total number of elements. The state is then interpolated to the quadrature points, and the flux at the state is evaluated. The gradients of the basis functions must be computed at the quadrature point and multiplied by the inverse Jacobian matrix. Finally, the residuals from the interior contributions to elements are calculated. The algorithm for evaluating residuals at each unknown value given by the element interiors is shown in **Algorithm 7**.

Algorithm 7 Residuals from element interiors

```
1: Initialize an R vector with zeroes
2: for $i = 0$ to $N_q$ do
3:   Check order of element ($q$)
4:   if $q = 1$ then
5:     Use linear precompute quantities
6:   else
7:     Use curved precompute quantities
8:   for $j = 0$ to $N_p$ do
9:     Compute the index
10:    Interpolate U at quadrature point
11:    $U_r = \mathbf{Phi} * \mathbf{U}[\text{index}]$
12:    Calculate Flux vector at quadrature points
13:    $\mathbf{J} = \text{jacobian(p, q, coords, coords}_{\text{ref}})$
14:    $\mathbf{R}[\text{index}]$ += $(\mathbf{GPhi}[i] * \mathbf{J}_{\text{INV}}) * \mathbf{F} * \mathbf{J}_{\text{det}} * \text{wq}[i]$
15: return R
```

In contributions from interior edges, precomputed basis functions are needed on each local edge or orientation to make the proper flux evaluation at each 1D quadrature point for the left and right element. It starts with looping over the interior edges. The basis function at the quadrature point is recalled for the left and right elements. Next, the states at the quadrature points are created to be the input to the flux evaluation after which, the residuals are calculated. **Algorithm 8** shows the algorithm to evaluate residuals at each unknown contributed by the interior edges.

Algorithm 8 Contributions from interior edges

```
1: Compute $N_{ti}$, the total number of interior edges
2: for $ie = 0$ to $N_{ti}$ do
3:   elemL = $\mathbf{I2E}[ie][0]$, elemR = $\mathbf{I2E}[ie][2]$
4:   edgeL = $\mathbf{I2E}[ie][1]$, edgeR = $\mathbf{I2E}[ie][3]$
5:   Get normal vector and length
6:   for $i = 0$ to $(N_p * N_{q1})$ do
7:     Get index
8:     Evaluate residual at unknown IL, IR
9:     Get PhiL, PhiR (CW, CCW) which correspond to edgeL, edgeR
10:    Interpolate U to each quadrature point
11:    $\mathbf{UL} = \mathbf{PhiL} * \mathbf{U}[\text{IL}]$, $\mathbf{UR} = \mathbf{PhiR} * \mathbf{U}[\text{IR}]$
12:    $\mathbf{F}$ = fluxfunction(UL, UR, n)
13:    $\mathbf{R}[\text{IL}]$ += $\mathbf{PhiL} * F * l * \text{wq1}[i]$
14:    $\mathbf{R}[\text{IR}]$ −= $\mathbf{PhiR} * F * l * \text{wq1}[i]$
15: return R
```

For boundary edges, the same manner as interior edge residual calculations has been applied, but in this case, the types of elements (linear and curved) have different numbers of 1D quadrature points. The pre-computed quantities that correspond to the element type are required. Also, the type of flux evaluation depends on the boundary group. **Algorithm 9** shows how to evaluate residuals at each unknown contributed by the boundary edge.

Algorithm 9 Contributions from boundary edges

```
1: Compute $N_{tb}$, the total number of boundary edges
2: for $be = 0$ to $N_{tb}$ do
3:     elemB = B2E[be][0], edgeB = B2E[be][1]
4:     Check linear or curved, and set precompute quantities vector
5:     if curved then
6:         Use curve normal and Jacobian edge vector
7:     else
8:         Use normal and length vector
9:     for $i = 0$ to $(N_p * N_{q1})$ do
10:        Get index IB
11:        UB = PhiL * U[IB]
12:        Calculate Flux at quadrature points
13:        if farfield then
14:            F = fluxfunction(UB, $\mathbf{U}_\infty$, n)
15:        else if wall then
16:            F = inviscid(UB, n)
17:        R[IB] += PhiL * $F$ * $\mathbf{J}_{\text{edge}}$ * wq1[$i$]
18: return R
```

2.2.3 Time-stepping

In this project, we chose RK2-SSP to drive the state to the steady state. **Algorithm 10** shows the time marching implementation.

Algorithm 10 Time Marching

```
1: $N_p$ = number of basis function
2: $N_{\text{dof}}$ = Element number*$N_p$
3: for ie = 0 to total number of element do
4:     Looping over each unknown in the element
5:     for i = 0 to total number of basis function do
6:         F[index] += iM[index][i]*R[index]
```

2.3 Post-Processing

The results shown in this project include Mach number contours, c_l, c_d, and the c_p distribution. The post-processing steps are similar to the previous project, except that since in high-order solutions, the states distribute non-linearly along the edges, therefore, numerical integration was needed during the force calculations. The equation for numerically integrating the force is as shown in **Equation 4**

$$[F_x, F_y] = \int p\vec{\mathbf{n}}dl = \sum_{\text{edge}}\sum_{\text{quad}} p_q \vec{\mathbf{n}}_q J_q w_q \tag{4}$$

where $[F_x, F_y]$ are the forces interested in x and y direction, p_q, $\vec{\mathbf{n}}_q$, J_q, and w_q are the pressure, normal vector, edge Jacobian, and weight interpolated on the quadrature nodes, respectively. States were integrated on the quadrature points to compute c_l and c_d. The steps of doing the numerical integration in code for force is also shown in **Algorithm 11**.

Algorithm 11 Numerical Integration for pressure

```
1: for edge in B2E do
2:     if the edge not far-field boundary then
3:         Extract the states on that edge from U
4:         Find the quadrature points and weights
5:         Find the quadrature coordinates in reference space
6:         Compute the states on quadrature points $\mathbf{U}_r$ using basis functions
7:         Compute the pressures on quadrature points $P_r$
8:         Numerically integrate the pressure using Equation 4 to get $F_x$ and $F_y$
```

The equation to calculate mach numbers on Lagrange points is shown in **Equation 5**.

$$\underline{M}_p = \underline{\underline{\Phi}}\underline{M} \tag{5}$$

where $\underline{M}_p$ is a vector of the Mach number on each sub-element point, ϕ is the basis function matrix, and M is a vector of the Mach number on each Lagrange point.
When plotting the Mach contour for the high-order solutions, variations within each element were shown. Finer sub-meshes were made for the Mach contour. States in those sub-elements were computed using interpolations with states stored on Lagrange nodes. This method was similar to interpolations for 2D quadrature points. Both the states and the coordinates of the new sub-elements were first computed in reference space, and then converted into global space and plotted.
For the airfoil, c_d and c_l were computed. However, for **Task 4** the force components are used to compare between different runs for Gaussian bump.

2.4 Adaptation

The mesh adaptation methodology for DG involves several key steps that differ from the process used in the finite volume method. The process can be summarized by **Algorithm 12**. The HLLE flux function was used just as in the uniform mesh sequences.

Algorithm 12 Mesh Adaptation

```
1: for edge in I2E + B2E do
2:   if edge is interior then
3:     Determine states at quadrature points of adjacent elements $U_r$
4:     Formulate integral as difference of the two element's states
5:     Compute edge-error using numerical integration and multiply with quadrature weight
6:   else if edge is non-inlet/outlet boundary then
7:     Compute state at boundary quadrature points
8:     Evaluate boundary pressure to compute mach number in normal direction $M^\top$
9:     Compute edge-error $\epsilon_e$ using numerical integration
```

The edge-error ϵ_e described above is calculated using **Equations 6** and **7**. In order to determine the best edge length scaling factor r, several adaptations must be performed, visually inspecting the refined cells in the mesh as well as looking at the mach number contours. After this point, the refinement is performed just as in the previous project however 2% of total edges are flagged instead of 3%. This is by virtue of the discontinuous states at edges which are being better at isolating problematic regions in the domain as opposed to the finite volume method.

$$\epsilon_e = h_e^r \int_{\text{edge}} |M_{k^+} - M_{k^-}| \, dx \tag{6}$$

$$\epsilon_e = h_e^r \int_{\text{edge}} |M_{k^\top}| \, dx \tag{7}$$

A special case occurs when curved elements are flagged for refinement. In this scenario, the refinement is performed in reference space where the children of the curved element are also curved. An illustration for the case $q = 2$ is shown in **Figure 2**.

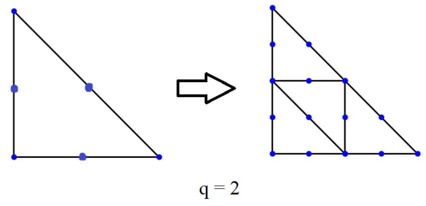

Figure 2: Process of uniformly adapting a $q = 2$ element in reference space

After each adaptive iteration, a good initial guess is used which involves transferring the states from the parent's Lagrange nodes to the children's. All adaptive runs are run at order $p = 2$.

3 Results

3.1 Task 1

The following figures are zoomed-in plots of q = 1 and q > 1 meshes in uniform refinement sequence

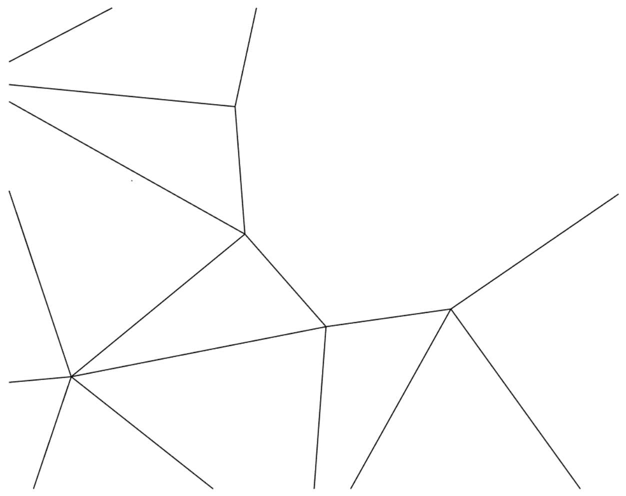

Figure 3: Zoomed-in Curved mesh for coarsest mesh (q = 1)

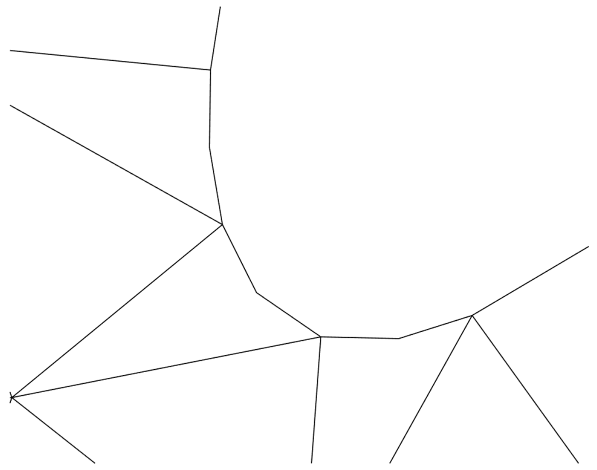

Figure 4: Zoomed-in Curved mesh for coarsest mesh (q = 2)

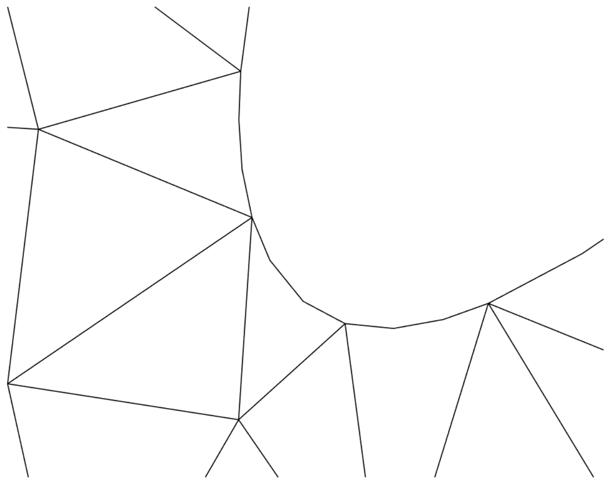

Figure 5: Zoomed-in Curved mesh for coarsest mesh (q = 3)

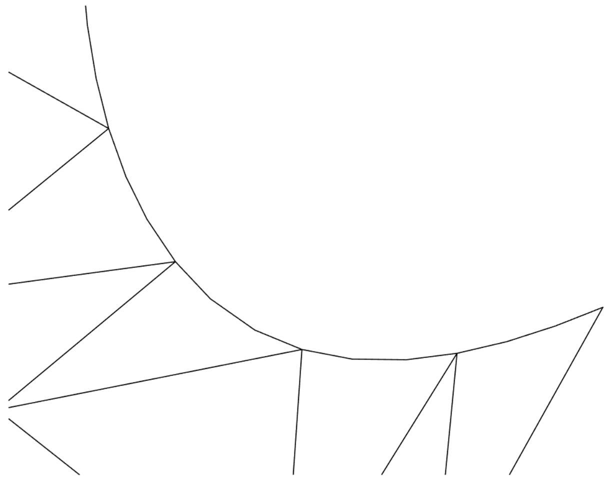

Figure 6: Zoomed-in Curved mesh for medium mesh (q = 3)

3.2 Task 2

Free-stream test was set up by changing all the boundary flux functions to HLLE. It was performed for all the mesh and order of spatial approximation ($p = 0, 1, 2, 3$), and in all cases, machine precision was obtained with a value $\sim 10^{-14}$ to 10^{-12}. Hence, verifying the application of the residual function.

Then, implementing the time-stepping into the solver keeping the same boundary condition as above, the free-stream preservation test was performed as shown in **Figure 7**. The residual norm hovered around 10^{-14} to 10^{-12}, thus verifying the implementation of the time-stepping implementation.

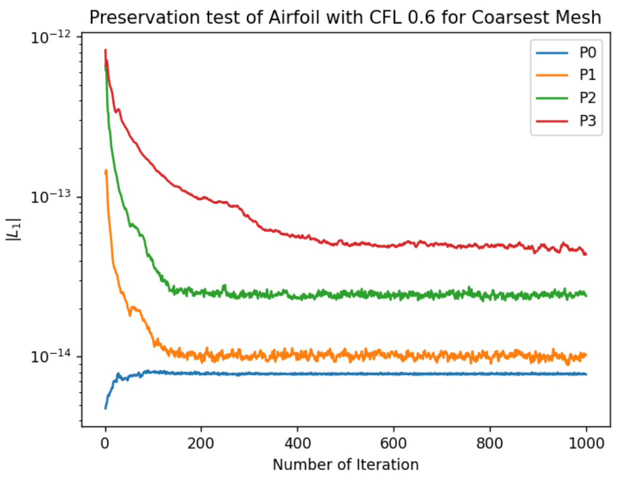

Figure 7: Preservation test for the Coarsest Mesh

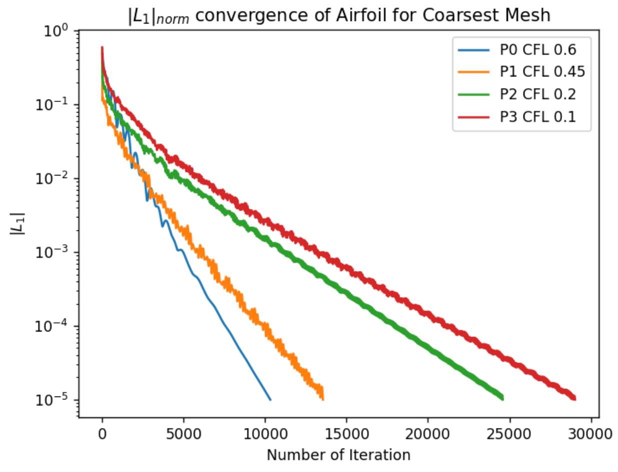

Figure 8: Residual norm convergence history

The solver was run for the coarsest mesh for all orders of spatial approximation ($p = 0, 1, 2, 3$) with the CFL number of 0.1 for each run until the residual norm was below 10^{-5}. **Figure 8** shows the convergence history of all cases

3.3 Task 3

Table 1 shows the calculated outputs for the coarsest mesh using ($p = 0, 1, 2, 3$)

Table 1: Output $(\mathbf{c_l}, \mathbf{c_d})$ for coarsest mesh $(\mathbf{p} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3})$

p	c_l	c_d
0	2.30240241	0.34334371
1	3.26862023	0.27934371
2	3.43735676	0.29334371
3	3.16388859	0.25581990

Figure 9 shows the pressure coefficient plots for $(p = 0, 1)$ for the coarsest mesh.

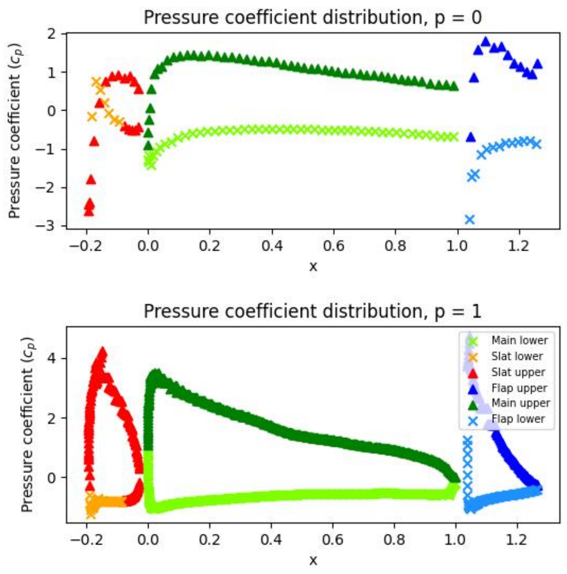

Figure 9: Pressure Coefficient Plots of Subsonic Three-Element Airfoil For $(\mathbf{p} = \mathbf{0}, \mathbf{1})$ Using The Coarsest Mesh.

Figure 10 shows the pressure coefficient plots for $(p = 2, 3)$ for the coarsest mesh.

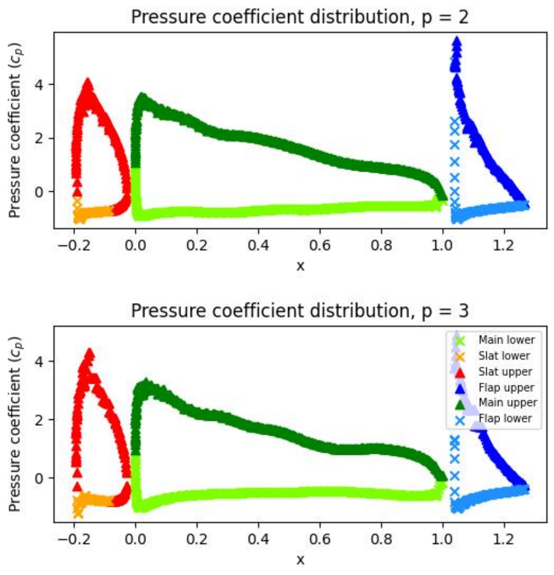

Figure 10: Pressure Coefficient Plots of Subsonic Three-Element Airfoil For $(\mathbf{p} = \mathbf{2}, \mathbf{3})$ Using The Coarsest Mesh.

The higher order of spatial approximation (p) was able to resolve the flow much better at the leading and trailing edge of the airfoil elements. There was a significant difference between $p = 0$ and $p = 1$, especially the leading and trailing edge. Thus, explaining the substantial difference between the c_l and c_d results from $p = 0$ to $p = 1$.

Figures 11, 12, 13, and 14 are the Mach contour plots on the coarsest mesh for $(p = 0, 1, 2, 3)$ respectively.

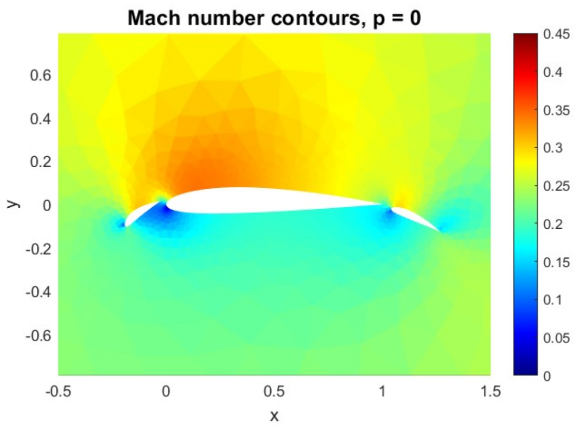

Figure 11: Mach Contour Plot for $\mathbf{p} = \mathbf{0}$

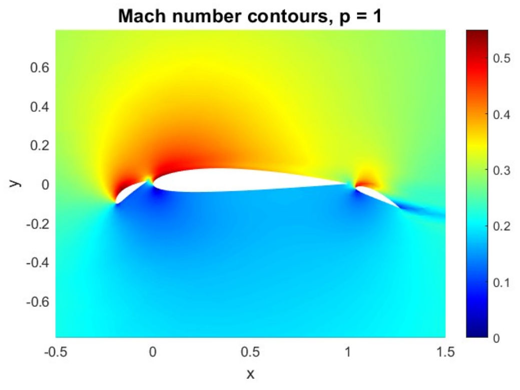

Figure 12: Mach Contour Plot for p = 1

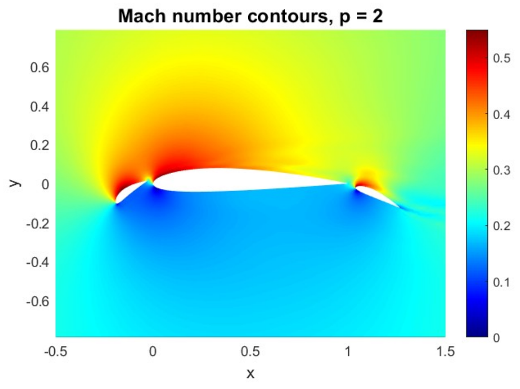

Figure 13: Mach Contour Plot for p = 2

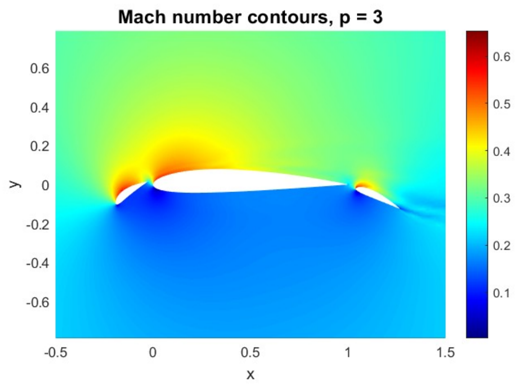

Figure 14: Mach Contour Plot for p = 3

3.4 Task 4

Due to the long run times, the convergence study is conducted on the bump. Lift and drag coefficients are less intuitive magnitudes for this case and as a consequence, the force in horizontal and vertical directions will be used to compare convergence rates. **Table 2** shows the F_y value and **Table 3** shows the F_x for each run.

Table 2: Convergence Study of the Uniform-Refinement for F$_y$

p	Real	Mesh 0	Mesh 1	Mesh 2
0	-1.921902839	-1.92885889	-1.92637350	-1.92429721
1	-1.921902839	-1.92212998	-1.92211311	-1.92201158
2	-1.921902839	-1.92193622	-1.92193789	-1.92191450
3	-1.921902839	-1.92193258	-1.92191696	-1.92191024

Table 3: Convergence Study of the Uniform-Refinement for F$_x$

p	Real	Mesh 0	Mesh 1	Mesh 2
0	$3.9 * 10^{-9}$	$2.74 * 10^{-3}$	$1.74 * 10^{-3}$	$5.46 * 10^{-4}$
1	$3.9 * 10^{-9}$	$7.37 * 10^{-5}$	$3.78 * 10^{-5}$	$1.87 * 10^{-5}$
2	$3.9 * 10^{-9}$	$4.34 * 10^{-6}$	$2.83 * 10^{-6}$	$2.83 * 10^{-6}$
3	$3.9 * 10^{-9}$	$4.71 * 10^{-7}$	$3.03 * 10^{-7}$	$8.32 * 10^{-8}$

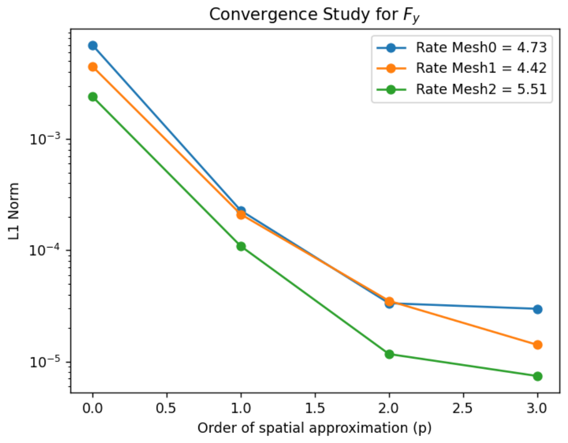

Figure 15: Convergence Study for order of spatial approximation (p = 0,1,2,3) w.r.t. to F_x

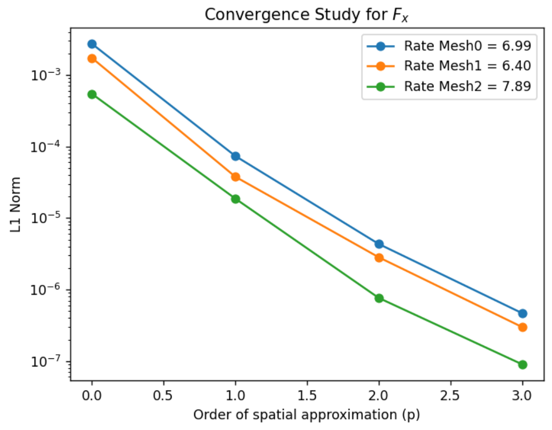

Figure 16: Convergence Study for order of spatial approximation (p = 0,1,2,3) w.r.t. to F_y

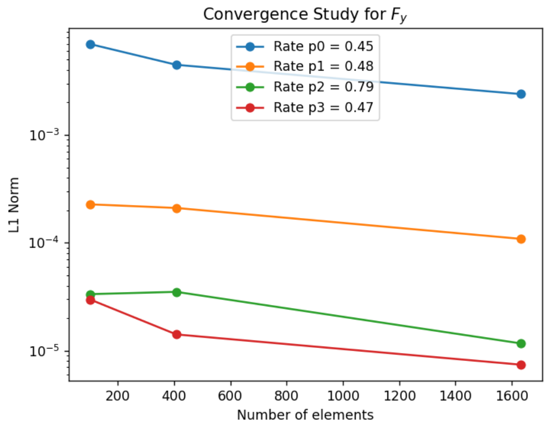

Figure 17: Convergence Study for increase in mesh elements w.r.t. to F_x

Figures 15 and 16 show the convergence study with respect to the increase in the order of spatial approximation (p) for all the meshes on a semi-log scale, as it was easier to visualize than a log-log scale. **Figure 17** shows the convergence study for the increase in mesh elements. The *real-solution* solution was taken to the value from running the finest mesh at $p = 2$. It is evident from the plot that as p increases the solution converges to *real-solution* faster. The rate of convergence is computed between $p = 1$ and $p = 2$, for all the plots.

To perform the convergence study, we first need to define a sequence of meshes with decreasing element sizes. Next, we solve the numerical problem and compute the error norm between the numerical solution and the exact solution. The L1 error norm is defined in **Equation 9**.

$$L_1 = \sum_{i=1}^{N} \frac{|u_i - u_{(exact,i)}|}{N} \tag{8}$$

where N is the number of solutions, u_i is the numerical solution, $u_{\text{exact},i}$ is the exact solution, which in our case, we evaluate the final output (c_l, c_d)

Next, we plot the logarithm of the error norm versus the logarithm of the element number and p to estimate the rate of convergence (r) using **Equation 9**.

$$r = \frac{-log(\frac{L_{1error2}}{L_{1error1}})}{log(\frac{h_2}{h_1})} \tag{9}$$

where $L_{1error1}$ and $L_{1error2}$ are the L1 error norms on two consecutive meshes with the number of elements and the two consecutive order p, h_1 and h_2, respectively.

For F_y, as the p increases beyond p = 3, the rate of convergences starts to decrease. However, this was not observed for F_x.

Moreover, for both F_x and F_y rate of convergences in the case of convergence study w.r.t to increase in p, increases as mesh becomes finer.

Thus, the order p influences the convergence rate, and it has been observed that increasing p improves the convergence rate. Increasing the order p generally results in a more precise solution because the polynomial basis functions can better approximate the true solution. However, increasing p also increases the number of unknowns in the linear system, which can increase the computational cost.

3.5 Task 5

The edge length weight factor r was chosen to be $r = -0.5$ as it produced refinement in areas of high discontinuity without targeting cells far away from the bump or strictly targeting nodes on the bump's boundary.

At iteration zero, using the coarsest mesh and order $p = 2$ the Mach contour and mesh is presented in **Figures 18 and 19**. Similarly, **Figures 20 and 21** shows the contour plots and mesh after 5 iterations.

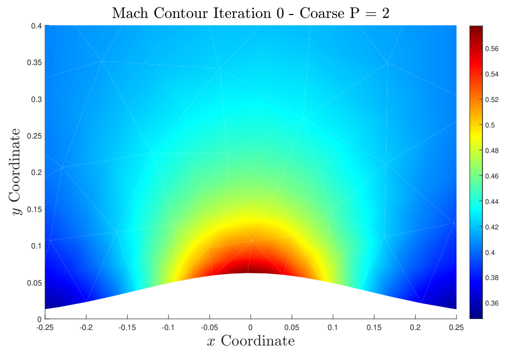

Figure 18: Contour plot of $p = 2$ solution on original coarse mesh.

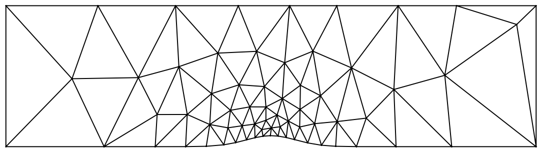

Figure 19: Starting coarse mesh.

After five adaptive iterations the mach contour and mesh is presented in Figures 20-21.

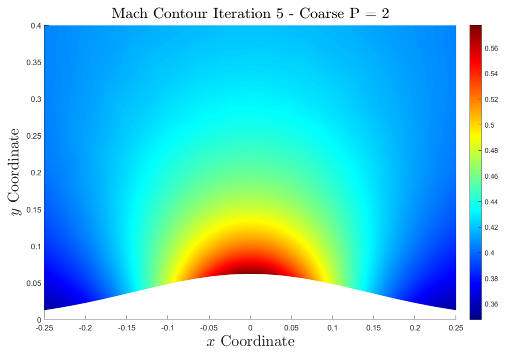

Figure 20: Contour after 5 adaptive iterations.

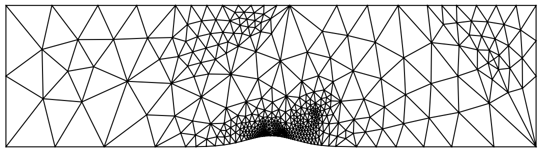

Figure 21: Mesh after 5 adaptive iterations.

The force acting on the bottom wall in y and x direction before and after adaptation are presented in Table 4.

Table 4: Force in x and y direction of bump at iteration $N = 0$ and $N = 5$

N	F_y	F_x
0	-1.92193618	1.34752128E − 06
5	-1.92194124	7.70849230E − 07

Comparing these results to that of the uniform mesh sequence, it becomes clear that the results are similar but differences do exist. After adaptation, the force in y direction is comparable to that of the medium-fine mesh at $p = 2$ solution in **Table 2** which sheds light on the benefit of adaptive refinement. However, uniformly refining a mesh will result in increasing the number of unnecessary computations performed by the solver. The pressure coefficient is computed for both iteration zero and iteration five and may be seen in **Figure 22**.

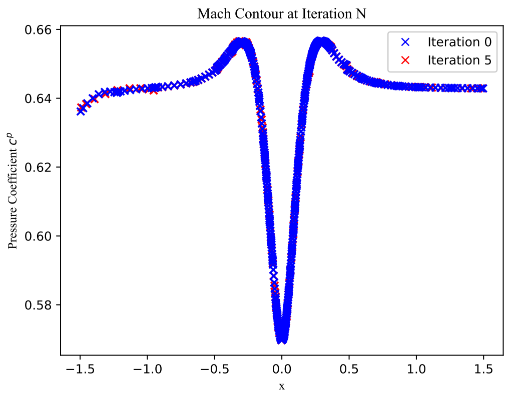

Figure 22: Coefficient of the pressure of the bottom wall of the bump.

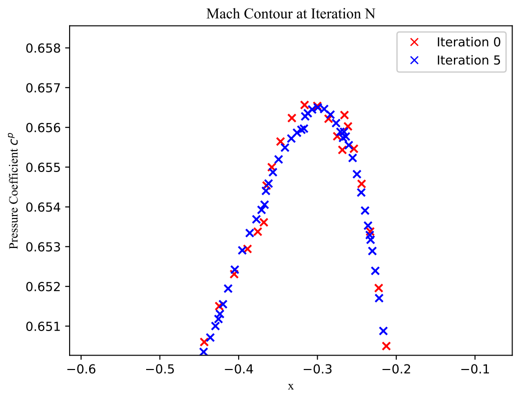

Figure 23: Coefficient of the pressure of bottom wall of the bump; zoomed-in view of the right side of the bump.

The adaptive iterations have alleviated the sharp discontinuities present at the first iteration; see **Figure 23**. As a final comparison, a boundary pressure plot of the last adaptive iteration is compared to that of uniform refinement at the medium-fine mesh; see **Figure 24**. They are in great agreement with each other but it's clear that uniform refinement is slightly superior as one sees the adaptive result separate at around $x = 0.35$.

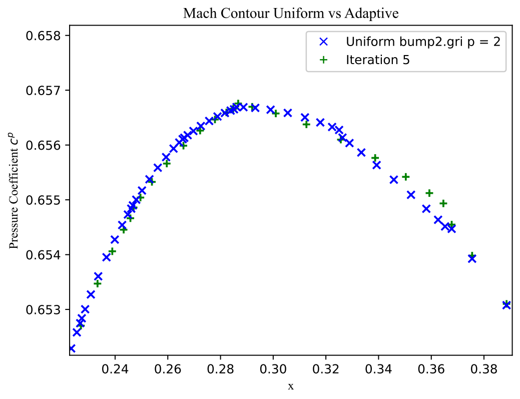

Figure 24: Boundary pressure of bottom wall, comparing adaptive and uniform results; ; zoomed in view of right side of bump.

4 Conclusions

Higher-order solutions perform great, even at coarser meshes and it is for that reason that it is an industry standard. Run times increased significantly with the increasing order of accuracy by virtue of the large jump in the number of degrees of freedom.

The convergence study depicts that increasing the order of spatial approximation p increases the rate of convergence to the *true solution*. However, the convergence rate decreases as p becomes greater than 2 and begins leveling out.

Mesh adaptive runs for higher order solutions such as $p = 2$ showed less of a difference than expected. This might be because of the simplicity of the bump or it could be the superiority of using a higher-order solver.