Discontinuous Galerkin Method of Solving for Flow over an Airfoil

Ben Bates, Thomas Greenwald, Vishwa Mohan Tiwari,
Moon Bakaya Hazarika, and Elizabeth Wraback
EcoLine

February 1, 2024

In this project, the finite-volume solver developed in Project-2 has been extended to obtain high-order approximation using the discontinuous Galerkin (DG) method. A DG solver has been developed that can run simulations of orders $p = 0, 1, 2, 3$ on both linear ($q = 1$) and curved meshes ($q = 2$). Frameworks for generating curved elements at boundaries and for carrying out solution-based mesh adaptation have also been developed. A post-processing framework is developed to adequately represent the high-order results. The solver and additional frameworks are used to study subsonic flow over a multi-element airfoil. The report discusses the concepts involved in the development of the different capabilities of the solver and the results obtained from the simulations carried out using it.

1 Introduction

Discontinuous Galerkin (DG) finite element methods can solve for flow field using high-order discretization. The finite element part of the solver allows for a variational framework already built into it, while the discontinuous approximation allows for the convective stability of the solution. DG can have high-order accuracy [5], error estimation [2, 3], *hp*-adaptation [6], stable viscous discretization [1, 8], and can be easily extended into a variational space-time algorithm [4,7], making it an effective tool for solving aerospace engineering problems.

2 Theory of DG

The 2-D Euler equations are solved using a conservative method in the present project. The 2D scalar field $u(\vec{x})$ for some initial condition $u_0(\vec{x})$ is given in Eq. .

$$\frac{\partial u}{\partial t} + \nabla \cdot \vec{F} = 0 \tag{1}$$

The state for an unsteady simulation can be approximated as,

$$\mathbf{u}(\vec{x}, t) \approx \sum_{m=1}^{N} \sum_{j=1}^{np} U_{m,j}(t)\phi_{m,j}(\vec{x}) \tag{2}$$

The PDE in Eq. 1 is solved in the weak form, which is found by multiplying the PDE by the basis functions and integrating it by parts. The resulting equation is given in Eq. , which can be further represented as Eq. .

$$\int_{\Omega_k} \phi_{k,i} \frac{\partial u}{\partial t} d\Omega - \int_{\Omega_k} \nabla \phi_{k,i} \cdot \vec{F} d\Omega + \int_{\partial\Omega_k} \phi^+_{k,i} \hat{F}(u^+, u^-, \vec{n}) dl = 0 \tag{3}$$

$$\mathbf{M}\frac{d\mathbf{U}}{dt} + \mathbf{R}(\mathbf{U}) = \mathbf{0} \tag{4}$$

In Eq. , $\mathbf{M}$ is the block-diagonal global mass matrix (given in Eq. 5) and $\mathbf{R}(\mathbf{U})$ is the residual vector (given in Eq. 6).

$$[\mathbf{M}_k]_{i,j} = \int_{\Omega_k} \phi_{k,i}\phi_{k,j} d\Omega \tag{5}$$

$$R_{k,i} = -\int_{\Omega_k} \nabla\phi_{k,i} \cdot \vec{F} d\Omega + \int_{\Omega_k} \phi^+_{k,i} \hat{F}(u^+, u^-, \vec{n}) dl \tag{6}$$

Section 3 provides details about the implementation of the aforementioned equations in the solver.

3 Methods

3.1 DG Solver

The input to the solver is the list of nodes, elements, the `I2E` matrix, and the `B2E` matrix, assuming uniform mesh order (i.e. q=2 everywhere, including on linear elements). The `I2E` matrix defines the elements over each edge and the `I2E` matrix defines the elements on the airfoil boundary or far-field boundary.

The solver is initialized using the converged finite volume method (FVM) solution (from Project 2) for the $p = 0$ solver for a given mesh. In the absence of an adequate FVM solution for initialization, the solution domain can be initialized using the free stream state for subsonic flow at every basis point. The free stream values use a specific heat ratio of $\gamma = 1.4$, a Mach number of 0.25, an angle of attack of $\alpha = 8°$, and convenient units to solve for the state. The solution for $p = 0$ is used for the initialization of simulations of order $p > 0$.

The present project solves the Euler equations,

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \vec{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{bmatrix} \hat{x} + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{bmatrix} \hat{y} \tag{7}$$

3.1.1 Pre-Computed Values

Basis Functions and their Gradients As previously discussed in Section 2, the solution needs to be approximated using basis functions to solve using DG methods, as given in Eq. 2. Full-order basis functions are used on equally space nodes in the present study. The full-order space consists of the span of basis functions $x^r y^s$, where $r \geq 0$, $s \geq 0$, and $r + s \leq p$. The solution is solved in reference space, on a unit right triangle, for which the j^{th} full-order basis function is given in Eq. .

$$\phi_j(\xi, \eta) = \sum_{s=0}^{p} \sum_{r=0}^{p-s} c_{k(r,s),j} \xi^r \eta^s \tag{8}$$

In Eq. , ξ and η are the reference space unit right triangle, and $c_{k(r,s),j}$ are the coefficients of the monomial expansion of the basis function j. These monomial expansion coefficients are solved from a matrix system that enforces the properties of the Lagrange system.

The gradient of the basis functions, i.e. the derivative of the basis function in the global space coordinate, is needed for parts of the residual calculation.

Quadrature Points The integration for the residual calculations is performed in quadrature for which quadrature points and weights are required. These are given by the Dunavant points, which are described in reference space, and thereby remain the same for all elements. Both the 1D and 2D quadrature points were used in the present study

Basis Functions at Quadrature Points and Gradients The basis functions are solved at the quadrature points to be used for integration. The gradient of these functions is determined by taking the gradient of the basis functions at the quadrature points with respect to the global coordinates. The results are given in reference space, so they remain the same for all elements. These are calculated for both 1D and 2D quadrature points in the present study.

Jacobians The Jacobian matrix, given by Eq. 9, is needed for the mapping from global to reference space.

$$\underline{J} = \frac{\partial \vec{x}}{\partial \vec{\xi}} = \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} \tag{9}$$

As the inverse mapping is also needed, the inverse Jacobian is evaluated as given in Eq. 10.

$$\underline{J}^{-1} = \frac{\partial \vec{\xi}}{\partial \vec{x}} = \frac{1}{\det(\underline{J})} \begin{bmatrix} y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix} \tag{10}$$

For curved elements, the Jacobian is evaluated as the sum of the product between basis point coordinates (in x and y) and the gradient of those points' basis functions in reference space. These can then be inverted as any 2x2 matrix.

Mass Matrix As previously discussed, the block-diagonal global mass matrix comes from the weak form of the PDE (Eqs. 3, 5). Each block-diagonal matrix is the integration in quadrature by taking the product of the basis functions are quadrature points, weights of the basis function, and the determinant. This is constant for all elements in reference space, but it must be scaled by a Jacobian when used. For curved elements, Jacobian scaling is applied at quadrature points, requiring individual integration for each curved element.

Normals The normal vectors are calculated for all locations of the basis function. For linear elements, these remain constant along an edge. For curved elements, these change along the curved edge. The normal vector was calculated using the Jacobian method, as given in Eq. 11.

$$\frac{d\vec{x_{edge}}}{d\sigma} = \frac{d\vec{x}}{d\xi}\frac{d\xi}{d\sigma} + \frac{d\vec{x}}{d\eta}\frac{d\eta}{d\sigma} \tag{11}$$

In Eq. 11, σ is the position along the element edge in reference space, and $\vec{x}_{edge}$ is the global nodes along the element edge. This can be easily calculated from the Jacobians already evaluated and the edge number.

$$\vec{n} = \frac{d\vec{x_{edge}}}{d\sigma} \times \vec{k} \tag{12}$$

Element Areas The area of both the linear and curved elements is calculated as the product of the determinant of the Jacobian and the weights of the basis points on each element.

3.1.2 Residual Calculation

As discussed above, the residual in a DG solver is calculated using Eq. 6. The first integral in the RHS represents the contribution of the state in the element interior to the residual and is evaluated using the 2D quadrature integration rule. The second integral represents the contribution of neighboring elements to the residual of the element and is evaluated using 1D quadrature integration along the edges of the elements.

Element Interior Closer inspection of the first integral in the RHS of Eq. 6 highlights the dependence of the element interior residual term on the gradient of the basis functions and the flux inside the element. Given that all calculations are performed for the reference element before mapping it to the respective element in the global domain, the gradients of the basis functions evaluated in the reference domain can be used in conjunction with the inverse Jacobian matrix at the element to map it to the global domain. As the integral is evaluated over the area of the element, the determinant of the Jacobian is used to scale the integration carried out in the reference frame for mapping it to the element in the global domain. The evaluation of the flux function requires the state of the element for the given iteration.

The integral can be numerically evaluated using the 2D quadrature rule. Considering N_r quadrature points for evaluating the integral, the integral can be rewritten as,

$$R_{k,i} = -\sum_{r=1}^{N_r} \left[\frac{\partial \phi_{k,i}}{\partial \vec{\xi}} \mathbf{J}^{-1} \right]_{\vec{\xi}_r} \cdot \vec{F}\left(\vec{\xi}_r\right) \left|J\right| w_r \tag{13}$$

The order of the solution (p) and of the geometry (q) determines the order of the Dunavant points sufficient for the summation in Eq. 13 to evaluate integral. The gradient of the basis functions

in the reference space, the inverse and determinant of the Jacobian, and the weights for the quadrature rule at the quadrature points can be determined in the precomputation. However, the evaluation of the flux at the quadrature point requires determining the states at those locations. This necessitates interpolating the state values evaluated at the basis to the quadrature points. The interpolation is performed using the values of the basis functions evaluated at the quadrature points.

Consider a state having N_{sv} state variables (e.g. 4 for an Euler problem). For a given element having a geometry of order q in the mesh and a solution of order p (N_p basis points), a matrix of basis function evaluation at quadrature points, Φ^T of size $(N_p \times N_r)$ is obtained using the `BasisQuad` function in the solver. The function `UQuad` takes the Φ^T and the state at basis points (U_{basis} of size $(N_p \times N_{sv})$) as inputs, takes the transpose of Φ^T to get Φ of size $(N_r \times N_p)$, and multiplies it with U_{basis} to obtain U_{quad} of size $(N_r \times N_{sv})$.

Functions `EulerFluxX` and `EulerFluxY` in the solver use the U_{quad} evaluated above and evaluates the x- and y- component of the Euler flux as given in Eq. 7 at the quadrature points.

The function `Res_Elem_Int_Indv` in the solver takes the basis function evaluations, evaluations of basis function gradients, inverse Jacobian and Jacobian determinant at all quadrature points in the element along with the state at its basis points as inputs, calls `UQuad`, `EulerFluxX` and `EulerFluxY` functions and carried out the summation in Eq. 13 over all the 2D quadrature points. The function `Res_Elem_Int_Total` in the solver splits the precomputed quantities for the entire mesh, extracts these quantities for the element in the global domain under consideration, and calls `Res_Elem_Int_Indv`. The matrix of residuals of the element, having the size $(N_p \times N_{sv})$, is appended to obtain the residual at the basis points for the entire mesh.

Interior Edge Residuals In the equation 6, the second integral at the element edges is implemented using 1D quadrature. The rows of a `I2E` matrix, which stores the information regarding the neighboring elements and their respective faces, are looped over to evaluate the residual at all the interior edges in the mesh. The states along the edge at the 1D quadrature point are evaluated from the left and right elements counterclockwise and clockwise, respectively Fig. 1. The state at quadrature points is obtained by multiplying the U_{basis} with a matrix of basis function evaluation at 1D quadrature points Φ of size $(N_r \times N_p)$. The fluxes associated with pairs of $\boldsymbol{u_L}$ and $\boldsymbol{u_R}$ are evaluated for each edge. As the numerical flux across the edge is required for residual calculation, the Roe flux 14 is calculated at the corresponding quadrature points. Given $\boldsymbol{u_L}$ and its neighboring state $\boldsymbol{u_R}$, the flux function is given by Eq. 14.

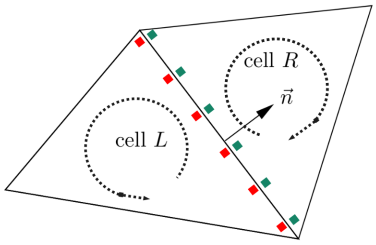


Figure 1: The representation of 1D integration using quadrature points at the edge of the left (red) and right element (green).

$$\hat{\boldsymbol{F}} = \frac{1}{2}(\boldsymbol{F_L} + \boldsymbol{F_R}) - \frac{1}{2}\left|\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}}(\boldsymbol{u}^*)\right|(\boldsymbol{u_R} - \boldsymbol{u_L}) \quad (14)$$

In Eq. 14, $\boldsymbol{F_L} = \boldsymbol{F}(\boldsymbol{u_L})$ and $\boldsymbol{F_R} = \boldsymbol{F}(\boldsymbol{u_R})$. $|\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{u}}(\boldsymbol{u}^*)|$ is the absolute values of the eigenvalues of the system, where $\boldsymbol{u}^*$ is a chosen intermediate state based on the two neighbor states. The Roe flux uses the Roe-average state, which for the Euler equations is given by Eq. .

$$\vec{v} = \frac{\sqrt{\rho_L}\vec{v_L} + \sqrt{\rho_R}\vec{v_R}}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \tag{15}$$

$$H = \frac{\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{16}$$

The eigenvalues of the flux Jacobian for the Roe averaged state are, $\text{diag}([u+c, u-c, u, u])$, which allows Eq. 14 to be written in coordinate-independent form as Eq. 17.

$$\hat{\boldsymbol{F}} = \frac{1}{2}(\boldsymbol{F_L} + \boldsymbol{F_R}) - \frac{1}{2}\begin{bmatrix} |\lambda|_3\Delta\rho + C_1 \\ |\lambda|_3\Delta(\rho\vec{v}) + C_1\vec{v} + C_2\hat{n} \\ |\lambda|_3\Delta(\rho E) + C_1 H + C_2 u \end{bmatrix} \tag{17}$$

To prevent expansion shocks in the system, an entropy fix is needed. To do so, all the eigenvalues are kept away from zero, by enforcing the condition that if $|\lambda|_i < \epsilon$ then $\lambda_i = \frac{\epsilon^2+\lambda_i^2}{2\epsilon}$ for all $i \in [1, 4]$. $\epsilon = 0.1c$ is considered for the present study.

Numerical integration for edge residuals is performed by summing the Roe flux, with appropriate weighting, at the quadrature points.

The residual at k^{th} element for the i^{th} basis function computed using 1D quadrature follows as given in Eq. 18.

$$R_{k,i} = \sum_{r=1}^{N_r} \left[\phi_{k,i}\left(\vec{\xi}_r\right)\right] \cdot \hat{F}\left(\vec{u}_L, \vec{u}_R, \vec{n}\right) \left|J_{edge}\right| w_r \tag{18}$$

In Eq. 18, $\vec{\xi}$ denotes the quadrature points and w_r are respective weight associated with those points. The edge Jacobian is $\left|J_{edge}\right| = \frac{ds}{d\sigma}$, here $\boldsymbol{s}$ is arc-length position along the edge and σ is the reference position along the edge.

These residuals are then added to the residuals from left element interior and subtracted from the right element interior residuals.

Boundary Edge Residuals The `I2E` matrix does not consider the edges next to a boundary element. Thus, we pass in the `B2E` matrix. In general, the calculation of the boundary edge residuals follows the methodology of the interior edge residuals (as given above), however, the numerical flux functions are treated differently.

The Roe-flux as given in Eq. 14 is still used for the far-field boundary. However, free stream state is used for the state on the right of the edge.

Inviscid wall flux is used for the boundaries of the airfoil, assuming that the wall is under the slip condition. The inviscid wall flux is given in Eq. 19.

$$\hat{\boldsymbol{F}} = [0, p^b n_x, p^b n_y, 0]^T \tag{19}$$

$$p^b = (\gamma - 1)\left[\rho E^+ - \frac{1}{2}\rho^+ |\vec{v}^b|^2\right] \tag{20}$$

3.1.3 Time Stepping

RK4 method was implemented for the time stepping in the solver as it allows higher levels of stability for the high-order elements. This requires four residual calculations on each element. Once the residual is calculated, it is multiplied by the inverse of the mass matrix. Using that result, the state update between each residual calculation can be calculated as per 21.

$$\mathbf{F}_0 = \mathbf{F}(\mathbf{U}^n, t^n) \tag{21}$$

$$\mathbf{F}_1 = \mathbf{F}(\mathbf{U}^n + \frac{1}{2}\Delta t\mathbf{F}_0, t^n + \frac{\Delta t}{2}) \tag{22}$$

$$\mathbf{F}_2 = \mathbf{F}(\mathbf{U}^n + \frac{1}{2}\Delta t\mathbf{F}_1, t^n + \frac{\Delta t}{2}) \tag{23}$$

$$\mathbf{F}_3 = \mathbf{F}(\mathbf{U}^n + \Delta t\mathbf{F}_2, t^n + \Delta t) \tag{24}$$

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \frac{\Delta t}{6}(\mathbf{F}_0 + 2\mathbf{F}_1 + 2\mathbf{F}_2 + \mathbf{F}_3) \tag{25}$$

The time step Δt is calculated on each element considering the CFL condition as per Eq. 26.

$$\Delta t_i = \frac{2\text{CFL}A_i}{|s_i|} \tag{26}$$

In Eq. 26, A_i is the area of the element and s_i is the sum of the maximum of each wave speed across the individual edges, as calculated by the numerical flux functions (see Section 3.1.2).

The solver continues time stepping until it reaches an L1 error norm of 1e-5, which we say is a converged solution.

The solver has been tested for $p = 0, 1, 2, 3$ using the free stream, free stream preservation, and adding boundary conditions on unit meshes and the actual mesh to ensure the ability to converge to a solution. Section 6.2 further elaborates on the demonstration of the testing.

3.2 Curved Elements

The capability of generating curved meshes as well as solving for them in the solver has been implemented to increase the accuracy of the solution around the airfoil, especially for higher-order solvers ($p > 0$). High-order meshes for $q = 2$ have been generated. First, the elements bounding the airfoil were determined. Then, on the straight edges opposite the curved edge, the high-order (HO) nodes were placed equidistant from the element nodes. For $q = 2$, the HO nodes are placed halfway between the element nodes on each linear edge. On the curved edge, the HO node was first determined as on the straight edge, but then snapped to the spline of the airfoil boundary, using the spline function.

Curved meshes have been generated for increasing mesh refinement levels, as detailed in Section 6.1.

4 Solution Adaptation

The framework of the solution adaptation, implemented in Project 2, is taken forward and modified to carry out solution adaptation for the present project. As opposed to the calculation of error metrics in the solution adaptation framework for the previous project, the error metric for each edge in the present study is obtained as an output of the post-processing and directly taken as input along with the states at all basis points of all elements. The original `.gri` file and the solution order are taken as inputs.

Given that for a particular element, there are N_p residuals per state variable for the present DG solver compared to the single residual per state variable in FVM, an interpolation function (`Basis_nn.m`) is developed that identifies the location of the basis points of the split element in the reference element based on the number of edges and the edge number(s) split, evaluates the value of the state variables at those points, and assigns them as the state of the split elements. The methodology of this interpolation is similar to that pursued to obtain the states at the plotting points for post-processing. The function to evaluate the basis functions at any given point in the reference element (`Basis.m`) is the same as the `Basis` function implemented in the main solver. `Basis_nn.m` takes the solution order, the state at the basis points of an element, and the array flagging which of its edges are split as inputs to determine the state of the newly generated elements.

The methodology for the remaining steps, i.e. extracting the edge and element mesh data, identifying the edges to split based on the error metric, splitting of elements based on the edge splitting criterion, and snapping of newly-generated nodes on the geometry splines, has been retained.

5 Visualization

Upon completion of the solver, and generation of a converged solution state, the next step is visualizing the data. The general procedure for solution visualization is the same for any order solution. First, the unrolled state vectors are read in and manipulated to separate the solutions per element. Then, the mesh is stepped through the element by element. Each element is transferred into reference space and subdivided into a chosen number of subelements. This allows the visualization of the unique, higher-order DG solutions across the element. In reference space, the solution state at the elements' basis points is interpolated to each of these subdivided plotting points.

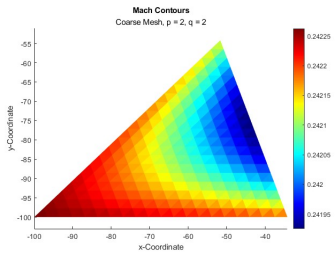


Figure 2: Discrete plotting over sub-elements

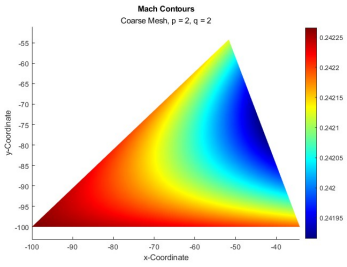


Figure 3: Linear interpolation plotting over sub-elements

Then, these reference space plotting points are transformed back into global space and plotted, and the function repeats for the next element. In Matlab, the function `patch` can be used to plot over the subelement points since each subelement itself is just a smaller triangle. This also allows a linear interpolation of a solution across the subelement triangle, which allows a linear approximation for the higher-order, nonlinear DG output of the full element state. This can be visualized in Figures 2 and 3. Both figures plot the mach contour over one cell with 20 subdivisions. In the first cell, Figure 2, the average mach over each subelement is plotted. In the second, Figure 3, the `patch` function performs a linear interpolation across the subelement. The result is a much smoother output, ideal for visualizing the complexity and variation of the solution across the cell.

Another task of the visualization code is to generate lift, drag, and airfoil coefficient of pressure data. For this task, the elements that have boundaries across their edges are identified and recorded in an array. This array is then sorted from lowest index to highest. While a super-linear complexity function call is not preferable, this function call only has to happen once in the entire runtime, and it allows the code to bypass a `find` function call for every loop. As the function begins to loop through every element in the mesh, if the current element index is the same as the first (and hence, lowest) index in the edge array, then the function calculates the pressure over the edge indicated in the original `B2E` array. This pressure is calculated by identifying the correct edge, then assigning 1-D quadrature points to that edge. The state across the whole cell is interpolated to these quadrature points, and then the pressure can be numerically integrated with the quadrature weights. Then, the index of the edge array is incremented, such that the next highest is flagged, and the main function loops until it hits that next highest. The lift and drag values are recorded, and after the loop, the coefficients are calculated. The coefficient of pressure is also plotted. with respect to the x-values of the cell edge it is measured across. Finally, the post-processing code outputs an error metric for mesh adaptation. By looping through the `I2E` and `B2E` files, the mach difference across the element-to-element side for two elements can be recorded in the case of an element-to-element side. For boundary sides, the normal mach number, which is calculated in the same loop when lift and drag are calculated, is instead reported. These values are scaled by edge length raised to a power r, and for these runs, r has been chosen to be -0.5. This adaptation metric file prints [$Elem_L$, $Elem_R$, Metric] for every element-edge combo. Boundary edges instead report [$Elem_L$, Boundary Flag, Metric].

6 Results

6.1 Task 1

The coarse meshes for $q = 1$ and $q = 2$ are presented in Figs. 4 and 5, respectively.

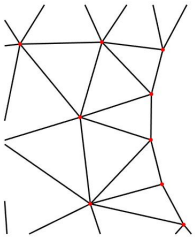


Figure 4: Leading edge of the main element of the airfoil for the coarse mesh with only linear elements. Red points represent the nodes.

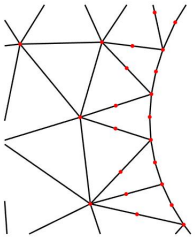


Figure 5: Leading edge of the main element of the airfoil for the coarse mesh with $q = 2$ curving along the edge. Red points represent the nodes.

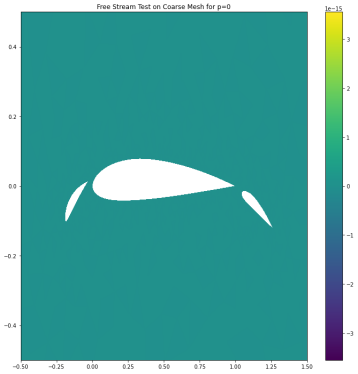


Figure 6: Residual values (colored) near the three-element airfoil for free stream test, for the first element of the state vector.

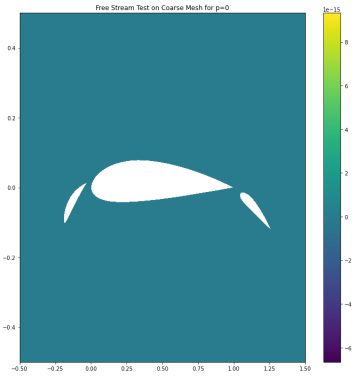


Figure 7: Residual values (colored) near the three-element airfoil for free stream preservation test, for the first element of the state vector.

6.2 Task 2

All versions of the solver were tested on unit meshes and the coarse mesh and found to converge to a solution for all p values. The results of the tests for the coarse mesh for $p = 0$ are presented.

The free stream test was performed by initializing the solution with the free stream state and setting all the boundaries to the free stream. Near-machine precision results were expected after residual calculation, which Fig. 6 is demonstrating for the first element of the state vector. The free stream preservation test was performed by continuing the free stream test for 1000 time steps. Again, near-machine precision results were expected to hold until that last time step, which is seen in Fig. 7.

The boundaries are then fully incorporated back into the solution while continuing to initialize with the free stream solution and only performing a finite number of time steps. Expected convergence was observed for the first 1000 time steps.

6.3 Task 3

We were unable to fully converge any solutions, however, we present semi-converged solutions ($\approx$ 10,000 iterations) for reference. Figs. 8, 9, 10, and 11 are the Mach number plots for increasing

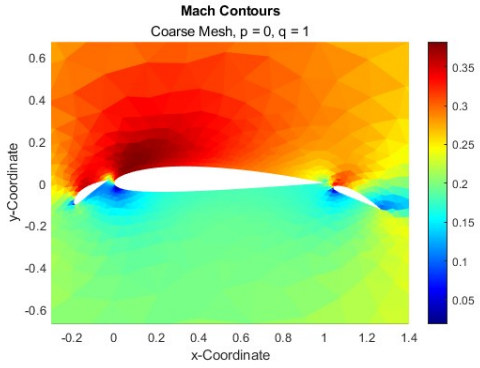


Figure 8: Mach number plot for $p = 0$, coarse mesh, $q = 1$.

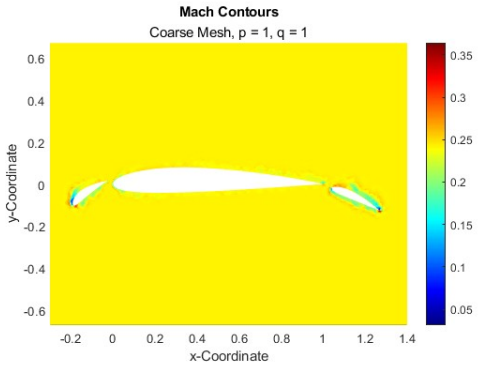


Figure 9: Mach number plot for $p = 1$, coarse mesh, $q = 1$

p values. Figs. 12, 13, 14 and 15 are the pressure coefficients over the airfoil elements for increasing p values. It is important to note that Figs. 11 and 15 is simulated with the curved elements. Table 1 shows the lift and drag coefficients on the airfoil.

The error in the $p = 1$ case is worth noting. There is an issue with the solution, which when compared to the solutions, shows the importance of having a converged solution in aerospace engineering problems.

Table 1: Lift and Drag Coefficients for increasing p, q on Coarse mesh.

Simulation	c_l	c_d
$p = 0$, $q = 1$	0.3166	0.0223
$p = 1$, $q = 1$	0.8234	5.4009
$p = 2$, $q = 1$	1.0282	0.0801
$p = 2$, $q = 2$	1.3800	0.2549

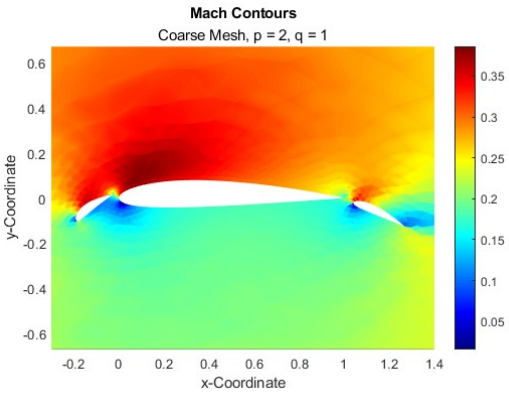


Figure 10: Mach Number for $p = 2$, coarse mesh, $q = 1$.

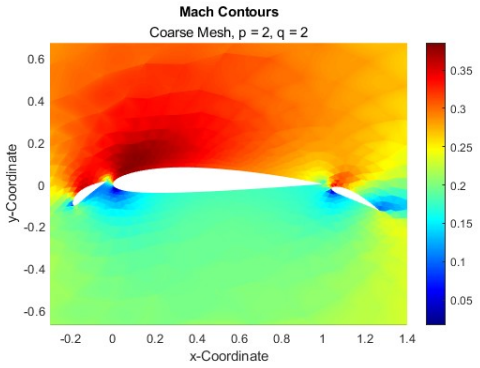


Figure 11: Mach number plot for $p = 2$, coarse mesh, $q = 2$

6.4 Task 4

We were unable to fully converge any solutions, however, we present semi-converged ($\approx$ 10,000 iterations) solutions for reference. Here we present the solution on the medium mesh for $p = 0$ solver in Figs. 16 and 17. The lift coefficient and the drag coefficient of 0.3533 and 0.1440 respectively. When increasing the order of the solver (p), the lift and drag coefficients are observed to increase

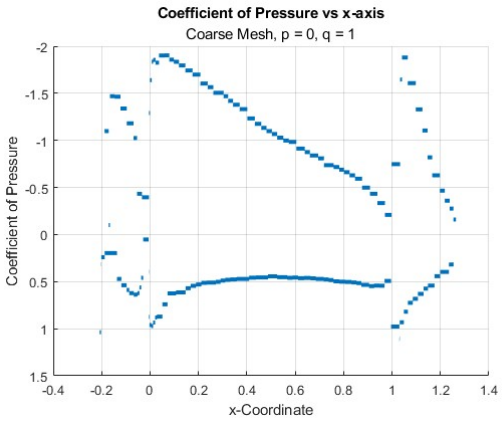


Figure 12: Pressure coefficient for $p = 0$, coarse mesh, $q = 1$.

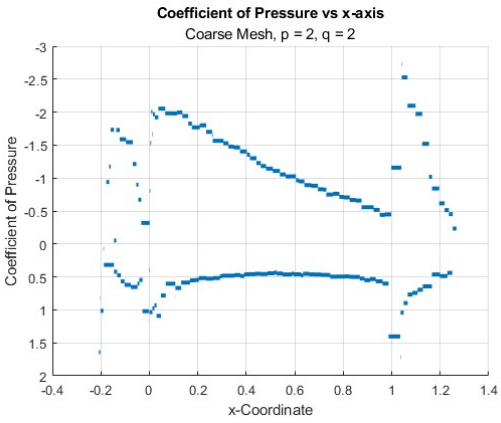


Figure 15: Pressure coefficent for $p = 2$, coarse mesh, $q = 2$.

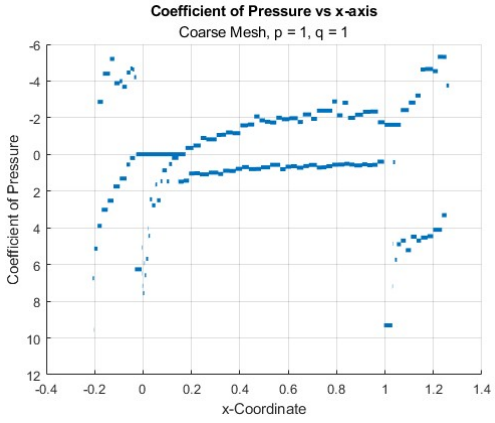


Figure 13: Pressure coefficient for $p = 1$, coarse mesh, $q = 1$.

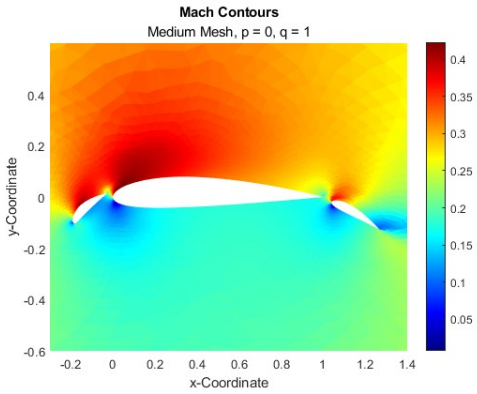


Figure 16: Mach number plot for $p = 0$, medium mesh, $q = 1$

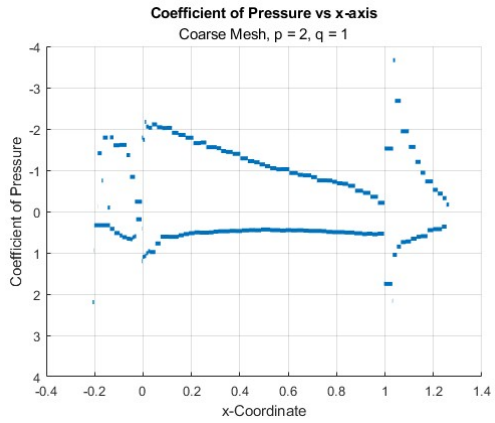


Figure 14: Pressure coefficient for $p = 2$, coarse mesh, $q = 1$.

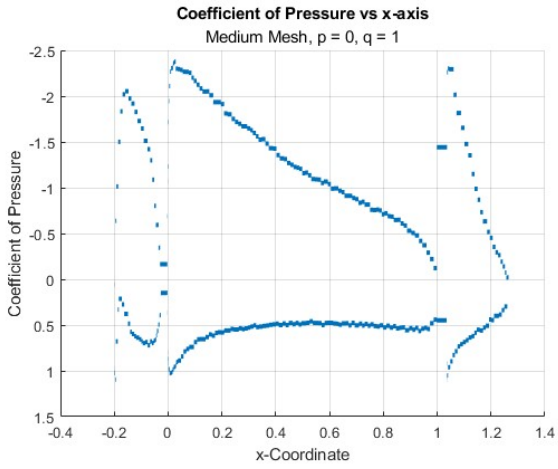
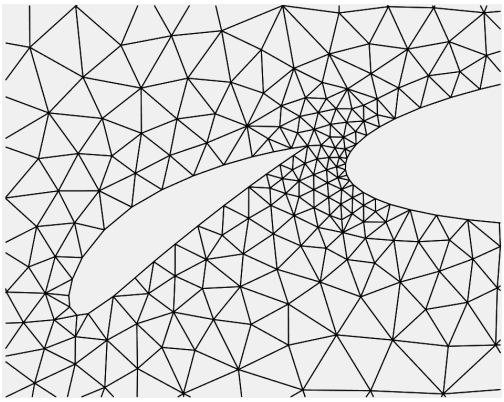
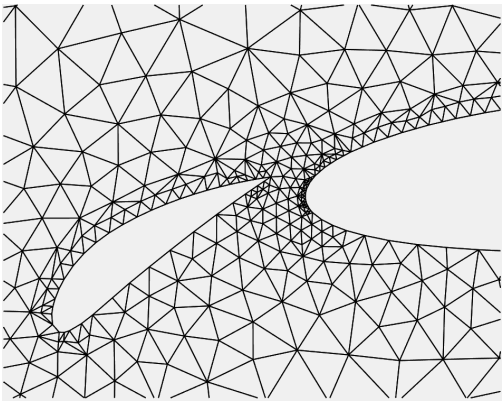


Figure 17: Mach number plot for $p = 1$, medium mesh, $q = 2$



(a) Before adaptation



(b) After 1^{st} iteration of adaptation

Figure 18: Solution-based mesh adaptation of coarse mesh for solution order $p = 2$

in magnitude. For example, when comparing $p = 0$ to $p = 2$ (see Table 1), the values of both the lift and drag increase. Additionally, when comparing the different mesh resolutions, the medium mesh also has higher lift and drag values than the coarse mesh (see first row of Table 1).

6.5 Task 5

Solution-based mesh adaptation is carried out for the coarse mesh for solution order $p = 2$. Figure 18 shows the mesh in the gap between the slat and main airfoil before and after the first iteration of mesh adaptation. Refinement in the mesh is observed at the leading edge of the main airfoil and the upper surface of both airfoils.

This adaptation can than be loaded back into the solver and run again, before being passed into the post processing code for the generation of the next adaptation.

7 Conclusions

A DG solver with the ability to solve for $p = 0, 1, 2, 3$ and on linear and high-order meshes is implemented in the present project. Between the different order solvers, the lift and drag coefficients are observed to increase in magnitude as the order of the solver increases. Between the different order meshes, the lift and drag coefficients are also found to increase in value with increasing resolution.

A mesh adaptation software is also implemented to work with an error metric and curved elements.

In conclusion, given more time, all solutions would have converged for all possible cases. This project merely demonstrates the capabilities of our DG solver, curved elements, and adaptation abilities. This provides different avenues going forward into our last project.

8 Effort Breakdown

- Moon (MBH) developed the functions for element interior residual calculation (apart from the pre-processing quantities) and the solution adaptation framework. She wrote Sections 3.1.2 and 4 detailing the methodology for the same. She assisted Elizabeth in organizing meetings and implementing schedules for deliverables.
- Elizabeth (EW) developed the skeleton of the solver, paying particular emphasis to the time-stepping algorithm and integrating all the individual codes into one solver, and developing the boundary residual calculation. She developed the ability to curve elements to $q = 2$. She also aided in debugging individual codes to ensure compiling and running ability. She wrote Sections 1, 2, 3.2, 3.1.1, 3.1.2, 6.1, 6.2 and helped with general editing. She organized meetings and imple-

mented schedules for deliverables.

- Ben (BB) wrote the pre-computation section of the code, including basis functions, mass matrices, Jacobians, normal vectors, and element areas. He also did the majority of debugging including implementing freestream and unit tests.
- Thomas (TG) wrote the function for post-processing, plot generation, and error metric calculation. He also debugged the main solver and solved several critical initialization and function call issues. He and (BB) worked closely to identify and solve persistent issues that plagued the unit tests. He also wrote 5 and helped proofread the report.
- Vishwa (VMT) developed the function that handles the interior edge residuals calculation for both curved and linear edges with Roe Flux, using pre-computed normals and basis functions from (BB). He carried out debugging on the interior edge with the help of (EW) and ran the simulations for a few cases. Further, he contributed to the report in section 3.1.2.

Table 2: Effort percentages.

	BB	TG	VMT	MBH	EW
development & coding	22	20	16	20	22
debugging	40	20	10	10	20
managing	12	12	12	24	40
report	18	18	18	22	22

References

[1] F. Bassi and S. Rebay. Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 40(1-2):197–207, September 2002.

[2] K. Bey. hp-Version discontinuous Galerkin methods for hyperbolic conservation laws. *Computer Methods in Applied Mechanics and Engineering*, 133(3-4):259–286, July 1996.

[3] Krzysztof J. Fidkowski. *A Simplex Cut-Cell Adaptive Method for High-order Discretizations of the Compressible Navier-Stokes Equations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2007.

[4] Krzysztof J. Fidkowski and Yuxing Luo. Output-based space-time mesh adaptation for the compressible Navier-Stokes equations. *Journal of Computational Physics*, 230(14):5753–5773, June 2011.

[5] Krzysztof J. Fidkowski, Todd A. Oliver, James Lu, and David L. Darmofal. p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier Stokes equations. *Journal of Computational Physics*, 207(1):92–113, July 2005.

[6] Paul Houston, Bill Senior, and Endre Süli. hp-Discontinuous Galerkin finite element methods for hyperbolic problems: error analysis and adaptivity. *International Journal for Numerical Methods in Fluids*, 40(1-2):153–169, September 2002.

[7] C. M. Klaij, J. J. W. van der Vegt, and H. van der Ven. Space time discontinuous Galerkin method for the compressible Navier Stokes equations. *Journal of Computational Physics*, 217(2):589–611, September 2006.

[8] J. Peraire and P. O. Persson. The Compact Discontinuous Galerkin (CDG) Method for Elliptic Problems. *SIAM Journal on Scientific Computing*, 30(4):1806–1824, January 2008.