Finite Volume Method Over Airfoil

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

February 1, 2024

A first- and second-order FVM solver was developed to solve the compressible Euler equations for subsonic and transonic flow over an airfoil. Multiple simulations were carried out to understand how the lift and drag coefficients evolved with different combinations of flux functions, limiters, and meshes. The transonic case was simulated on the medium and fine meshes using Roe flux function and Barth Jespersen limiter, which showed that more resolution is needed to fully resolve the steep gradients on top of the second-order solver with limiter. Solution based adaptive mesh refinement was carried out to highlight important features associated with transonic flow using both the first- and second-order cases.

1 Introduction

Finite Volume Method (FVM) is the technique based on cell averaged values by which the integral formulation of the conservation laws is discretized directly in the physical space. It continues to be the most widely applied method in CFD due to its generality, conceptual simplicity, ease of implementation on structured and unstructured grids, and, most importantly, that the conservation equations are automatically satisfied through the direct discretization of its integral form [4]. Jameson et al. [6] demonstrated the ability of FVM for solving the Euler equations in arbitrary geometric domains by developing a new combination of a finite volume discretization in conjunction with carefully designed dissipative terms of third order, and a Runge Kutta time stepping scheme, and using it to determine the steady transonic flow past an airfoil using an O mesh. FVM methods have since become a standard for solving for flow over bodies of varying complexity and considering different flow regimes. In the present study, a first- and second order FVM solver has been developed to study flow over a multi-element airfoil in subsonic and transonic regime, using various flux functions and limiters. Unstructured meshes with varying levels of refinement have been used to highlight the effect of mesh resolution on the results obtained.

2 Methods

2.1 Numerical Flux Function Descriptions

2.1.1 Roe Flux

Given two neighboring states, $\boldsymbol{u_L}$ and $\boldsymbol{u_R}$, the flux function is given as,

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

Here, $\boldsymbol{F_L} = \boldsymbol{F}(\boldsymbol{u_L})$ 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 as,

$$\vec{v} = \frac{\sqrt{\rho_L}\vec{v_L} + \sqrt{\rho_R}\vec{v_R}}{\sqrt{rho_L} + \sqrt{rho_R}}, \quad (2)$$

$$H = \frac{\sqrt{\rho_L}H_L + \sqrt{\rho_R}H_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \quad (3)$$

The eigenvalues of the flux Jacobian for the Roe averaged state are, $\mathrm{diag}([u+c, u-c, u, u])$, which allows Eq. 1 to be written as, in coordinateindependent form,

$$\hat{\boldsymbol{F}} = \frac{1}{2}(\boldsymbol{F_L} + \boldsymbol{F_R}) - \frac{1}{2}\begin{bmatrix} |\lambda|_3\Delta\rho + C_1 \\ |\lambda|_3\Delta(\rho\vec{v}) + C_1\vec{v} + C_2\hat{n} \\ |\lambda|_3\Delta(\rho E) + C_1 H + C_2 u \end{bmatrix} \quad (4)$$

The Roe flux allows for careful upwinding of the state, one wave at a time. However, to prevent expansion shocks in the system, an entropy fix is needed. To do so, we keep all the eigenvalues away from zero, by, if $|\lambda|_i < \epsilon$ then $\lambda_i = \frac{\epsilon^2+\lambda_i^2}{2\epsilon}$ for all $i \in [1,4]$. Here, we set $\epsilon = 0.1c$

2.1.2 Rusanov Flux

The flux function is,

$$\hat{\boldsymbol{F}} = \frac{1}{2}(\boldsymbol{F_L} + \boldsymbol{F_R}) - \frac{1}{2}s_{max}(\boldsymbol{u_R} - \boldsymbol{u_L}) \quad (5)$$

where the maximum wave speed is calculated as the maximum of the eigenvalues of the system,

$$s_{max} = \max[(u_L \cdot \hat{n} + c_L), (u_R \cdot \hat{n} + c_R)] \quad (6)$$

Using s_{max}, allows us to upwind every wave in the system. However, this can upwind systems with very different wave speeds too much and create a lot of numerical diffusion.

2.1.3 HLLE Flux

The flux function for the HLLE is,

$$\begin{aligned} \hat{\boldsymbol{F}} &= \frac{1}{2}(\boldsymbol{F_L} + \boldsymbol{F_R}) \\ &- \frac{1}{2}\frac{s_{max}+s_{min}}{s_{max}-s_{min}}(\boldsymbol{F_L} - \boldsymbol{F_R}) + \boldsymbol{F_R} \\ &+ \frac{s_{max}s_{min}}{s_{max}-s_{min}}(\boldsymbol{u_R} - \boldsymbol{u_L}) \end{aligned} \quad (7)$$

where,

$$\begin{aligned} s_{min} &= \min(s_{L,min}, s_{R,min}), && (8) \\ s_{L,min} &= \min(0, u_L - c_L), && (9) \\ s_{R,min} &= \min(0, u_R - c_R), && (10) \\ s_{max} &= \max(s_{L,max}, s_{R,max}), && (11) \\ s_{L,max} &= \min(0, u_L + c_L), && (12) \\ s_{R,max} &= \min(0, u_R + c_R) && (13) \end{aligned}$$

And the magnitude of the wave speed through the cell is,

$$s_{mag} = \max(|u_L| + c_L, |u_R| + c_R) \quad (14)$$

2.1.4 Inviscid Wall Boundary Flux

The airfoil wings are inviscid walls, so no flow can pass through. It is not a no-slip boundary condition. The flux on the airfoil is given as,

$$\hat{\boldsymbol{F}} = [0, p^b n_x, p^b n_y, 0]^T \quad (15)$$

where

$$p^b = (\gamma - 1)\left[\rho E^+ - \frac{1}{2}\rho^+|\vec{v}^b|^2\right] \quad (16)$$

2.2 First-Order FVM Solver

The first order FVM solver will be described briefly because it was also implemented and documented in AERO 523. First, node coordinates, the nodes defining each element, and each element's three neighbors (possibly including boundaries flagged with negative values) are extracted from the .gri file and read into matrices. From this information, a list of edges is made including the following information: adjacent elements, normal vector components, edge length, and if the edge is a duplicate. Next, the free stream state is calculated based on the stagnation values and far-field Mach number. The state matrix is initialized to this free stream value. The state is then updated by looping over all edges and calculating the chosen flux based on current states, unless the edge is a duplicate, in which case the flux is assigned a previously found value. Residuals R_j are then found on each element as a length-weighted sum of fluxes on that element's edges. Maximum wave speeds are also found during the flux evaluation and a total wave speed s_j for each element is similarly calculated. This information is then used to make a state update according to the following equation, where the CFL number is 1:

$$u_j^{n+1} = u_j^n - 2CFL\frac{R_j}{s_j} \quad (17)$$

This process is repeated until an L_1 residual norm of $1e-5$ is reached.

2.3 Second-Order FVM Solver

Second order FVM solvers are constructed by considering linear distribution of state inside the cell, rather than piecewise constant distributions as considered for first order FVM solvers. Starting with cell averaged data, the solution is piecewise linearly reconstructed by evaluating state gradients such that new extrema are not generated. The fluxes are then evaluated and the solution evolved one time step and averaged over the cell. [2].

To implement in code, the structure is largely analogous to the first order FVM. The largest change is the implementation of gradient calculations, with and without limiters, as discussed in the next sections. With the state gradient in each element available, the centroid and edge midpoints are extracted from the node and element matrices and used to estimate state values at the edge midpoints. These values are then used instead of the cell averages in the flux evaluations. To accomplish this, the edge matrix used in the first order solver now includes an extra column of information, the edge number of the edge that borders the same to elements. That is, if edge k_1 of element L borders element R, the edge matrix will store the identity of edge k_2 of element R, which borders L. Access to this information ensures that fluxes are evaluated using the midpoints of correct edges, not just the correct neighboring element. These fluxes are used to find residuals as in the first order FVM.

The second order FVM also used strong-stability preserving (SSP) RK2 instead of first order Euler time-stepping, to ensure that the method is fully second order. In SSP RK2 two residual calculations are made, first using the current state, and then again using the predicted state found from the first residual evaluation. The state is then actually updated using the average of the two calculated residuals. Again, this process is repeated until a sufficiently low L_1 residual norm is achieved.

2.4 Gradient Calculation for Second-Order FVM Solver

Calculation of the state gradient in the cells is the first step towards flux determination for second-order FVM, and are thus important to be carried out accurately. Several methods of gradient calculation exist, of which the Gradient Theorem and Plane Normal Methods were explored in the present study.

2.4.1 Gradient Theorem Method

The Gradient Theorem method evalutaes state gradient by utilising the Gauss Theorem that states that the outward flux of a vector field through a closed surface is equal to the volume integral of the divergence over the region inside the surface [1]. This is applied in the dimension of the domain (two-dimension for the present study), which gives the state gradient at cell centroid $\vec{L}_0$ by:

$$\vec{L}_0 = \frac{1}{A_0}\sum_{k=1}^{3}\frac{u_0+u_k}{2}\vec{n}_{0k}\Delta l_{0k} \tag{18}$$

where u_k and $\vec{n}_{0k}$ are the state value at and the vector joining the cell centroid to the k^{th} neighbouring cell centroid respectively.

2.4.2 Plane Normal Method

The Plane Normal method utilizes the coordinates of the centroid of the neighbouring elements and the value of the state stored at them to define a plane in the spatial-state domain (obtained by taking cross product of the vectors joining the centroids in the domain) and finding the spatial gradient of the same. Mathematically, a plane can be defined in the $x-y-u$ space having a normal vector $\vec{n}$ such that

$$\begin{aligned}\vec{n} &= [n_x, n_y, n_u]\\ &= [x_2-x_1, y_2-y_1, u_2-u_1]\\ &\quad\times[x_3-x_1, y_3-y_1, u_3-u_1]\end{aligned} \tag{19}$$

$$\vec{L}_{123} = -\frac{n_x}{n_u}\hat{x} - \frac{n_y}{n_u}\hat{y} \tag{20}$$

where x_k, y_k and u_k are the x- and y-coordinates of, and the state value at the centroid of the k^{th} neighboring element respectively, while $\vec{L}_{123}$ is the gradient vector obtained from using the neighboring elements. $\vec{L}_{0ij}$ can also be defined this way using the cell and the i^{th} and j^{th} neighboring elements, which is useful for boundary elements.

2.5 Limiters for Second-Order FVM Solver

Second- and higher-order upwind spatial discretizations require the use of limiter functions in order to prevent the generation of oscillations and spurious solutions in regions with large gradients such as shocks [3]. The limiter reduces the evaluated state gradient in the cell to the cell face/edge in order to constrain the solution variations, while preserving monotonicity. At strong discontinuities, it reduces the gradients to zero to prevent the generation of new extrema, in turn reverting to the first-order upwind scheme in the immediate vicinity of large gradients. Away from the presence of large gradients, the limited solutions are almost identical to the unlimited solutions.
For the present project, the Barth Jesperson and the MP limiters have been implemented, as further detailed

2.5.1 Barth Jesperson Limiter

The Barth Jesperson Limiter is amongst the most widely used limiter functions. The method involves scalar scaling of the state gradient to enforce a monotone solution. However, it is dissipative and tends to smear discontinuities. Numerical noise in smooth flow regions also presents problems in activation of the limiter.
The Barth Jesperson framework and its associated functions have been developed in C++ and integrated in the second-order FVM solver developed for the present project.
Description of Algorithm
The required inputs include: (1) array of node coordinates, (2) array of neighboring elements for each element, (3) array of node ids defining each element, and (4) value of a state variable at each element. The scaled state gradient for each element to be used for flux determination at the edge is output.

1. Arrays of size $n_{elem} \times 2$, *Ecent, grad_u* and *grad_u_lim* are initialized
2. The centroid of each element is calculated using two-dimension node coordinates:
$$x_k^c = \frac{\sum_{i=1}^{3} x_k^{n_i}}{3}, y_k^c = \frac{\sum_{i=1}^{3} y_k^{n_i}}{3} \tag{21}$$
where x_k^c and y_k^c are the x-and y- coordinate of centroid of k^{th} element, and n_i refers to its' i^{th} node.
3. The state gradient, $\vec{L}$ is determined using the state values and centroid coordinates of each element and its neighbouring elements. For the scope of the present project, the Gradient Theorem (GT) Method and the Plane Normal method was implemented as described in 2.4. The Plane Normal method was found to produce infinite gradients for certain set of elements, and was thus not used further.
4. The maximum and minimum of $[u_0, u_1, u_2, u_3]$ were identified for each element ($u_i = i^{th}$ neighboring elements' state) and stored as u_{max} and u_{min} respectively
5. State reconstructions at the nodes of an element, u_i^N were determined for each element as follows:
$$u_i^N = u_0 + \vec{r_i^N}\vec{L} \tag{22}$$
where $\vec{r_i^N}$ is the position vector of the i^{th} node w.r.t. cell centroid.
6. A scalar limiting factor corresponding to each node of the element, α_i^N is determined as follows:
$\alpha_i^N =$
$$\begin{cases} min\left(1, \frac{u_{max}-u_0}{u_i^N-u_0}\right) & \text{if } u_i^N - u_0 > 0 \\ min\left(1, \frac{u_{min}-u_0}{u_i^N-u_0}\right) & \text{if } u_i^N - u_0 < 0 \\ 1 \text{ otherwise} \end{cases} \tag{23}$$
7. The scalar limiting factor for the element, α was taken to be the minimum of the scalar limiting factors calculated for each of its nodes.
8. The scaled gradient for the cell, $\vec{L}'$ was determined for each element by multiplying the previously obtained vector, $\vec{L}$ with the α determined

2.5.2 Maximum Principle Limiter

The Maximum Principle (MP) limiter follows much of the same theory as the Barth-Jesperson limiter discussed above. For an average cell state,

a linear reconstruction can be created that identifies how that state is distributed within the cell. The maximum principle demands that that this reconstruction not be introduce new local extrema on the midpoints of it's edges
From this, it becomes apparent that the cell gradient must be limited to keep a monotone solution across cell edges. To this front, a "MP region" can be visualized on a cell as a location that a gradient vector must exist in to not exceed the neighboring cell's state [5]. For each edge k, the change in state between the neighboring cell across edge k and the current cell is measured. The region is constructed between the cell centroid, where a "zero" change in state would occur, and the magnitude of this measured change. Mathematically, this region is represented in the following equality:

$$min(u_k - u_0, 0) \leq \vec{r} \cdot \vec{L} \leq max(u_k - u_0, 0)$$

Essentially, it is a representation of the largest possible gradient that could satisfy the maximum principle condition in the direction of the cell edge from the centroid. The limiter function analyzes this region and scales the calculated gradient to ensure it falls within the region. Different methods of scaling exist. The simplest is Limited Central Difference (LCD) scaling, which is simply a scalar value applied to the measured gradient. Further scaling schemas attempt to project the gradient vectors into the MP region, which can improve simulation accuracy. For this project, the implemented MP limiter utilizes the LCD method due to its simplicity and the limited benefit gained from more complicated limiter schemes.

Description of Algorithm
The inputs require (1) node coordinates, (2) neighboring elements for each element, (3) node locations identifying the element, and (4) value of a state variable at each element. Again the scaled state gradient for each element is output.

Procedure:

1. The algorithm reads in the aforementioned inputs.
2. Cell area, side length, and side normal vector are calculated from node locations.
3. The neighboring cell states are read, and if one of the cell neighbors is a boundary, it is rejected and replaced with a repeated reading of a non-boundary neighbor.
4. Cell gradient is calculated through the Gradient Theorem (GT) method defined in Equation 18.
5. The cell centroid is calculated using Equation 21.
6. The maximum scaling factor for each side is calculated from the following:
   $\alpha_k =$
   $$\begin{cases} \frac{max(u_k-u_0,0)}{\vec{r_{0k}}\cdot\vec{L}} \text{ if } \vec{r_{0,k}} \cdot \vec{L} > max(u_k - u_0, 0) \\ \frac{min(u_k-u_0,0)}{\vec{r_{0k}}\cdot\vec{L}} \text{ if } \vec{r_{0,k}} \cdot \vec{L} < max(u_k - u_0, 0) \\ 1 \text{ otherwise} \end{cases} \quad (24)$$
7. The overall scaling parameter, α is chosen as the minimum of the calculated α_k's.
8. The gradient is scaled and returned, $\vec{L_{out}} = \alpha\vec{L}$

2.6 Solution Based Mesh Adaptation

Grid adaptation can enhance the computational efficiency since the adapted mesh is only dense in the important regions, such as those having large gradients of states. A strict control over the mesh size growth is indispensable, else the mesh size can significantly grow due to over-refinement thereby reducing the potential efficiency gain [7].
The solution-based mesh adaptation adopted for the present project, developed in MATLAB, utilizes the mesh adaptation framework developed for Project-1.

2.6.1 Description of Algorithm

The inputs required include (1) `.gri` file, (2) values of the state variables at all elements obtained from FVM solver, and (3) ratio of specific heats (γ). The final `.gri` file with updated list of additional nodes and elements generated and the values of state variables mapped on the refined mesh are output.

1. A modified version of `plotgri.m` developed and provided by Prof. Fidkowski was used to

read the input `.gri` file and store the node coordinates, list of nodes on the different boundary faces and the nodes making up the element corners

2. A matrix, *edge_info* of size $(3 \times n_{elements}) \times 6$ was defined to store the relevant information regarding the edges
3. For each element, three edges were defined between the three nodes of the element. The nodes at either end of the edge were stored in the first and second column of the *edge_info*, while the element on the left of the edge was stored in the third column
4. The duplicate edges obtained in the previous step were identified, and the left element for the duplicate edge was stored as the right element for the edge considered in the fourth column of *edge_info*. The identified duplicate edges were removed.
5. The boundary edges were assigned the boundary group by looping over the edges and finding the column in *node_b_check* for which both the rows corresponding to the nodes of the boundary edge were labelled '1', and stored as -1×(boundary group id) in the fourth column of *edge_info*
6. A $n_{elements} \times$ 3 sized matrix, *E1IB_inc* was defined to store the ids of the element edges. A copy of this matrix, (*E1IB*) was made.
7. A $n_{elements} \times$ 3 sized matrix, *ENE_inc* was defined to store the ids of the neighbouring elements. A copy of this matrix, (*ENE*) was made.
8. The edges (in *E1IB*) and neighbouring elements (in *ENE*) for each element were rearranged in counter-clockwise order based on the order of the nodes opoosite to them.
9. The length of all edges was calculated and stored in the fifth column of *edge_info*.
10. An array of length (n_{edges}), *edge_ref_c* was defined and initialized to '0'.
11. A matrix of size $n_{elements} \times 3$, *E2IB_split_check* was defined and initialized to '0'.
12. The input state file was read into the framework, and the pressure and Mach number were calculated for each element using the

$$p = (\gamma - 1)\left[\rho E - \frac{\rho|\vec{v}|^2}{2}\right] \qquad (25)$$

13. The error metric was calculated for each interior edge as follows:

$$\epsilon_e = |M_l - M_r|\sqrt{l_e} \qquad (26)$$

where M_l and M_r are the Mach no. on either of the elements adjacent to the edge, and l_e is the edge length. The edge length was squared to reduce the effect of large edge lengths (and thereby increase effect of sharp state gradient across the edge) on increasing the associated error metric with it.

14. The error metric for each edge on the main airfoil, slat and flap was calculated as follows:

$$\epsilon_e = |M_\perp|\sqrt{l_e} \qquad (27)$$

where $M_\perp$ was calculated by considering the component of the velocity perpendicular to the edge.

15. 3% of the total number of edges for which the error metric was calculated, and having the highest error metric values, were flagged for refinement by updating the sixth column corresponding to the edge in *edge_info* to '1'.
16. The elements associated with the edges flagged for refinement above were identified, and the remaining edges of those elements were flagged for refinement by updating all columns of *E2IB_split_check* in the row corresponding to the element to '1'. The other element associated with the second batch of edges flagged for refinement were identified, and the column(s) corresponding to those edges in *E2IB_split_check* for the other element were updated to '1'.
17. Copies of the node coordinate list, element-to-node list and element state matrix (*V_mod*,*E2N_mod* and *elem_state_mod* respectively) were made for applying additional updates to the same.
18. New nodes corresponding to the midpoints of the edges flagged for refinement were defined. Their ids' were stored separately in

edge_mp to correlate between the split edge and the midpoint, and the coordinates of the new nodes are added to *V_mod*.

19. The number of subelements an element was to be split into was identified by looping over the elements in *E2IB_split_check* and calculating the sum of all column entries for each row. If the sum for the row was equal to 3, all edges of the element needed to be split. If the sum for the row was equal to 1, one edge of the element needed to be split. If the sum for the row was equal to 2, two edge of the element needed to be split The new subelements were suitable defined, ensuring counter-clockwise listing of nodes The newly defined subelements were added to *E2N_mod*. For each new subelement added, the state of its parent element was added to *elem_state_mod*
20. The original elements marked for refinement were removed from *E2N_mod* and *elem_state_mod*.
21. The new nodes were added to the list of boundary group nodes by looping over the edges in *edge_info*, checking if a boundary edge had been split, and adding its' midpoint node id to the list of boundary group nodes.
22. The nodes on the boundary of the main airfoil, flap and slat were projected to the spline defining their geometry using the `nspline.m` function developed in Project 1.
23. *V_mod*, *E2N_mod* and the updated list of boundary nodes were used to write the output `.gri` file for the solution adapted refined mesh. *elem_state_mod* was used to write the updated state matrix mapped on the solution adapted refined mesh.

3 Results

3.1 Task 1

The flux functions were implemented and tested using unit tests. All fluxes passed their respective unit tests. The unit test we will present here is for the Rusanov flux (see 2.1.2). For simplicity, the states we are using $u_L = [1, 0.9, 0.4, 1.9]^T$ and $u_R = [1, 0.9, 0.04, 1.9]^T$, and $\hat{n} = \frac{1}{\sqrt{2}}\hat{x} + \frac{1}{\sqrt{2}}\hat{y}$. This gives the fluxes, truncated to four digits after the decimal, on each state to be, $F_L = [0.9192, 1.2275, 0.7680, 2.2669]^T$ and $F_R = [0.6647, 1.0209, 0.4493, 1.6391]^T$

Then, we can calculate the maximum wave speed follow Eq. 6, $s_{max} = \max[1.8094, 1.5795] = 1.8094$. Thus, we can use Eq. 5 to calculate the flux through the element, $\hat{\boldsymbol{F}} = F_R = [0.79195, 0.1242, 0.9343, 1.953]^T$.

This matches the maximum wave speed, `1.80941`, and the flux, `[0.79196, 1.12419, 0.934258, 1.9635]`, within round-off errors.

3.2 Task 2

The black lines in Figure 1 demonstrate the pressure coefficient along the individual pieces of the airfoil using the Roe flux function on the coarsest mesh in the subsonic regime. The pressure coefficient is calculated as,

$$c_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty |v_\infty|^2} \tag{28}$$

3.3 Task 3

The blue lines in Figure 1 demonstrate the pressure coefficient (see Eq. 28) along the individual pieces of the airfoil using the Roe flux function on the coarsest mesh in the subsonic regime. Here, the BJ limiter was also used. The second-order solver is showing higher pressures at the front of the slat and main elements of the airfoil, in comparison to the first-order solver. This is likely due to the ability to resolve more structures in these regions, even when comparing the same mesh.

3.4 Task 4

When comparing the various subsonic flow conditions on this airfoil, multiple phenomena are investigated using the lift and drag coefficients on this system. The lift coefficient is defined as,

$$c_l = \frac{L'}{\frac{1}{2}\rho_\infty |v_\infty|^2 c} \tag{29}$$

and the drag coefficient is defined as,

$$c_d = \frac{D'}{\frac{1}{2}\rho_\infty |v_\infty|^2 c} \tag{30}$$

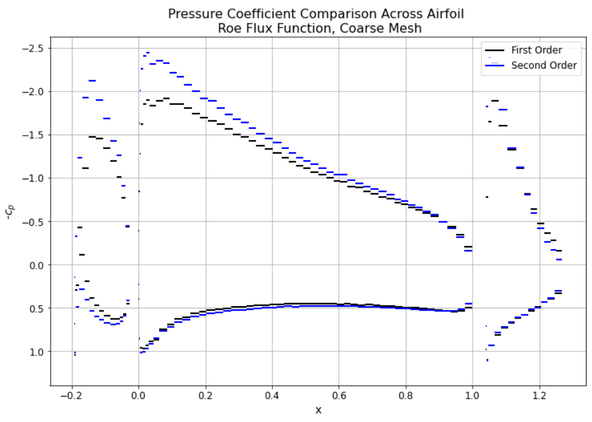

Figure 1: Pressure coefficient across airfoil comparing the first- (black lines) and second-order (blue lines) solvers using the coarse mesh, the Roe flux, and, for the second-order, the BJ limiter.

The reference chord length (c) of 1 is used. ρ_∞ and v_∞ are the free-stream density and velocity, and L' and D' are the lift and drag forces (respectively) over all three elements of the airfoil. The drag is aligned with the free-stream velocity while the lift is perpendicular to it. The force is computed by integrating the pressure over the airfoil boundaries. When considering the lift and drag coefficients, we generally expect the lift in this case to be near zero and the drag to be large because of the low-angle of attack of the incoming flow.

When comparing the different flux functions, the three differ in dissipation. In general, Rusanov is the most dissipative, HLLE is less dissipative than Rusanov, but still remains dissipative, and Roe is the least dissipative. This is evident when comparing the lift and drag coefficients between the flux function results. The Roe flux tends to have both larger lift and larger drag, than the Rusanov or HLLE fluxes. For example, when comparing the first-order solutions for the medium mesh (see Table 1), the Roe flux is closer to what is expected for the lift and drag to be in this case, as previously discussed. The lift is very low, while still being positive, as is expected for a positive angle of attack. The drag is much larger because there is flow going straight at the airfoil. As the mesh becomes finer, the lift and drag coefficients for all fluxes does increase, but not as significantly as the Roe flux. This same pattern is also seen in the 2nd order results, with both limiters.

Table 1: Lift and Drag Coefficients for 1st Order, Medium Mesh runs.

Flux Function	Lift	Drag
Roe	0.1436	2.5144
Rusanov	-0.1250	2.3306
HLLE	-0.1182	2.3558

To determine whether the solvers are first or second order, we calculate the rate of the convergence by comparing the finest converged mesh solution (the true solution) to the intermediate cases. This was done for as many of the converged cases as we received and were averaged together. The first-order solver converged with a rate of ≈ 0.7496, while the second-order solver was ≈ 1.6329. These are slightly below the expected rates of 1 and 2, however considering the simplicity of the implementation of the solvers, it is understandable.

The no limiter limiter does not allow for convergence in any case. This was expected considering that in the second order scheme for convergence the limiters are generally needed.

Similarly, for finer meshes, it was difficult for the MP limiter to converge properly. This is due to the simplicity of the MP limiter. If more complexity was added into the limiter, such as projection, then there possibly could be a higher level of convergence. For the cases that did convergence with the MP limiter, it tended give quite similar values to the first-order solution. For example, for the Rusanov flux on the 8k mesh, in first-order gave a drag coefficient of 2.3306, which was the same in the second-order. Similarly, this happened for the HLLE and Roe fluxes. Again, due to the simplicity of the MP flux limiter implemented here, the limiter did not improve the accuracy of the solution.

The BJ limiter tended to work the best in all cases, except with the Rusanov flux function. Likely this is due to the high level of dissipation evident in the Rusanov flux function. When comparing the first- and second-order lift and drag coefficients for the same flux functions, but using the BJ limiter brought the values closer to what was expected for the lift and drag coefficients to be. For example, the drag coefficient, using the HLLE flux function on the 8k mesh was 2.3558 with the first-order solver, where it was 2.6782 with the second-order solver. This is even between than how the Roe flux was performing in first-order. Thus, showing that using a more dissipative flux function with a highly accurate limiter can increase the accuracy of the results when compared to a less dissipative flux with a lower order solver.

3.5 Task 5

The transonic flow condition with $M_\infty = 0.5$ was solved for the medium and fine meshes with the Roe flux in the first order solver and the Roe flux with the BJ limiter in the second order solver. These were chosen because they were the most accurate and tended to converge. The output of the first order solution was used as the first step in the second order solution.

The lift and drag coefficients (Equations 29 and 30, respectively) are reported in Table 2. Interestingly enough, the lift and drag coefficients between the two solvers, for each mesh, as the same (this is further discussed at the end of this section).

Table 2: Lift and Drag Coefficients for Transonic Runs.

	Lift	Drag
1st Order, Medium Mesh	0.1608	2.8415
1st Order, Fine Mesh	0.2625	3.1488
2nd Order, Medium Mesh	0.1608	2.8415
2nd Order, Fine Mesh	0.2625	3.1488

The pressure coefficient plots (Figures 2 and) show how the pressure varies over the different elements of the airfoil for the different solvers and meshes. When comparing the medium and fine meshes over the main section and the flap section of the airfoil, the pressure coefficient increases substantially in the fine mesh over the top of the elements. The two solvers show similar results, irrespective of mesh size along the main and flap elements of the airfoil. Along the slat element, there is not a significant amount of variation between the different meshes or solvers, but they showcase the flow coming together at the back of the airfoil.

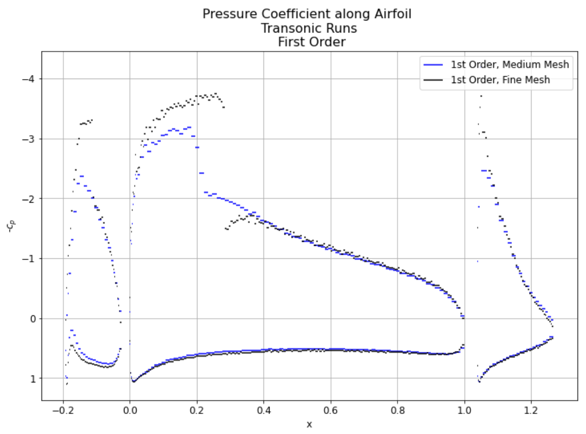

Figure 2: Pressure coefficient along airfoil for the transonic runs. Comparison of the first-order simulation with the medium and fine meshes.

Underneath the airfoil, there is very low pressure coefficient, that is almost stable under the slat and main elements of the airfoil. This can be considered to result in the negligible lift observed in Section 3.4. However, in the flap at the front of the airfoil, there is an increase in the pressure coefficient due to the angle of attack and the transonic nature of the flow.

It is observed from the Mach number plots for the first order runs (Figures 4 and 5) that the shocks begin to form on the upper surface of the main airfoil at the leading edge. With the finer mesh, these details also begin to get sharper and the maximum Mach number increases. Similar trend is observed in the Mach number plots for the second order runs (6 and 7). These plots showcase why we are seeing the different variations in the pressure coefficient in Figures 2.

As noted throughout, the first- and second-order results are nearly identical in each mesh for the transonic case. This is because for large gradients, like these shock-like structures that are developing over the airfoil, the second-order with limiters reduces to near first-order accuracy. Second-order solvers with limiters tend to improve results in

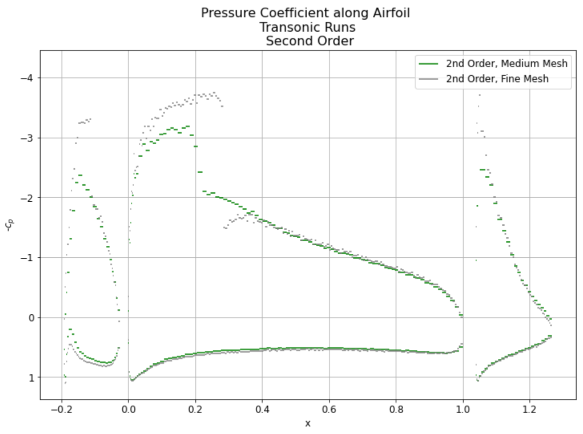

Figure 3: Pressure coefficient along airfoil for the transonic runs. Comparison of the second-order simulation with the medium and fine meshes.

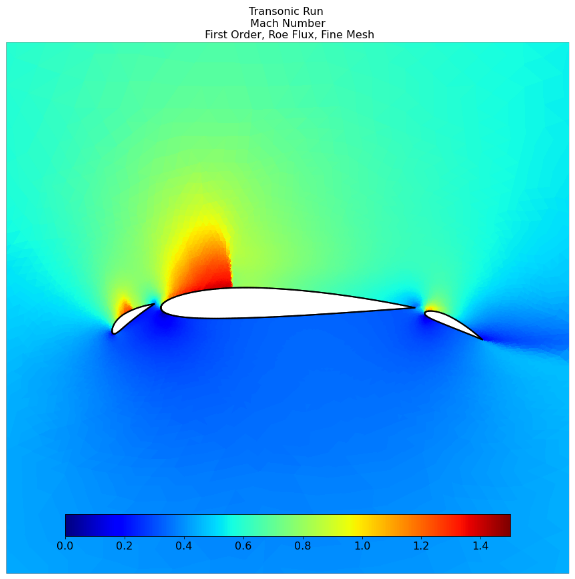

Figure 5: Mach number over airfoil on fine mesh for the transonic run with first-order solver.

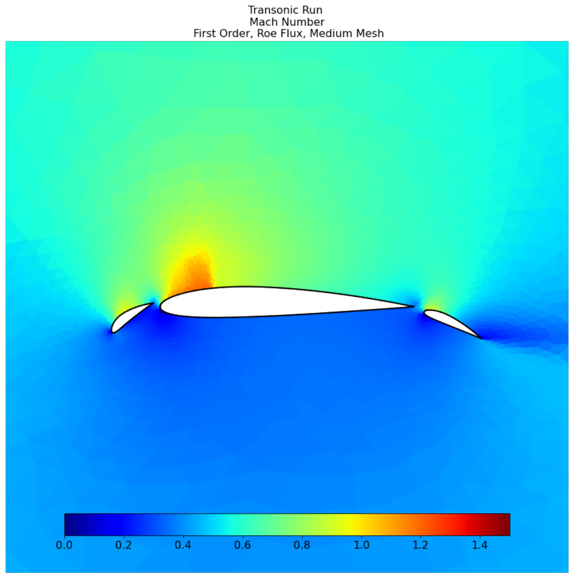

Figure 4: Mach number over airfoil on medium mesh for the transonic run with first-order solver.

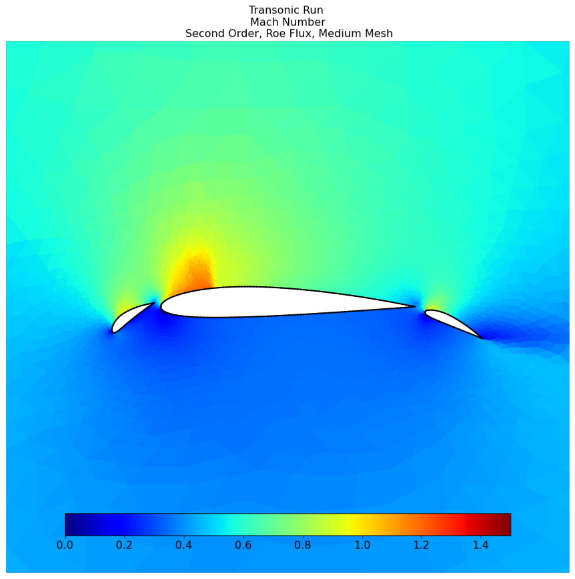

Figure 6: Mach number over airfoil on medium mesh for the transonic run with second-order solver and BJ limiter.

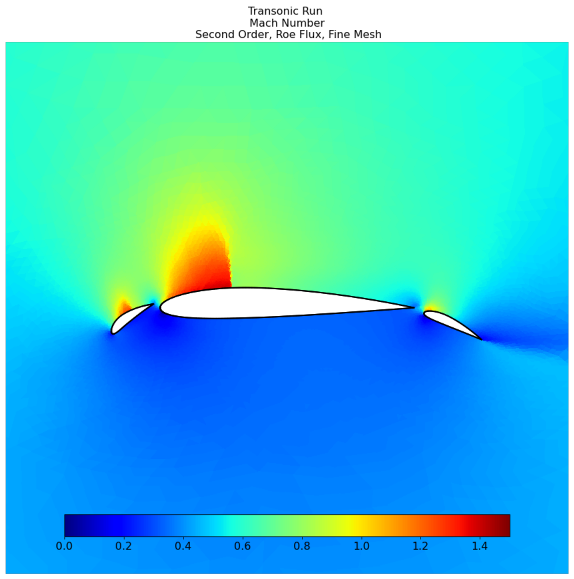

Figure 7: Mach number over airfoil on fine mesh for the transonic run with second-order solver and BJ limiter.

smooth flows. Therefore, to truly resolve any kind of discontinuity with high-accuracy, more resolution is needed. This is why the fine mesh is showing more of the shock structure than the medium mesh.

3.6 Task 6

In comparison to the uniformly refined mesh, the lift and drag coefficients increased significantly (see Table 3) after the five layers of adaptation. Likely, this is due to what was previously discussed, we are getting more resolution in the region over which there are sharp gradients with the adaptation, which showcases the life and drag on the airfoil further. This is further showcased in the comparison of the pressure coefficients (see Figure 8), which are increased at the front of each element of the airfoil. While the general structure of the pressure coefficient is the same between the uniform and adaptively refined mesh, it is much larger here.

4 Conclusions

In conclusion, for the subsonic runs, we found that the Roe limiter was the least dissipative because it had the most reasonable values for the lift and drag coefficients, for both the first- and second-order simulations. Additionally, we found that as the mesh becomes finer, the lift and drag coefficients become larger, in both solvers, since the meshes are able to resolve more structure in the runs. Using the finest mesh cases, we were able to show that the first-order solver is 0.7496 and the second-order solver is 1.6329. Furthermore, in the subsonic case, the no-limiter and MP limiter did not tend to converge for all cases. While the BJ limiter did. The BJ limiter also aided in bringing the lift and drag coefficients closer to what was expected.

For the transonic runs, the finer meshes were better able to resolve the large gradients due to the shock structures beginning to form. Most importantly, we found that the first- and second-order solutions were extremely similar. This is due to the fact that second-order solvers with limiters work the best for smooth flows and, thus, its accuracy is reduced to being a first-order solver. It is more important to have high resolution to understand

Table 3: Lift and Drag Coefficients for Adaptive Runs.

	Lift	Drag
1st Order	0.3900	9.2128
2nd Order	0.5766	10.5826

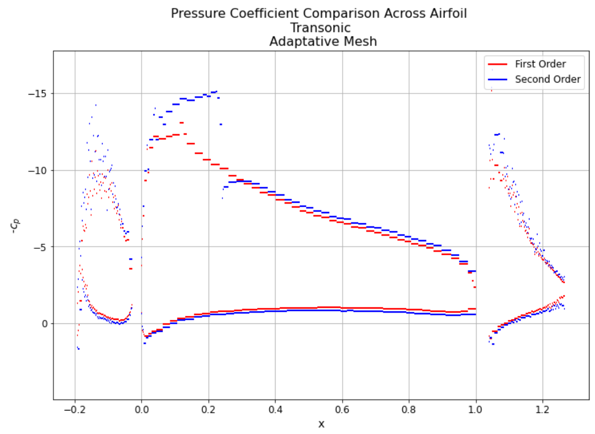

Figure 8: Pressure coefficient over airfoil elements for first- (red) and second-order (blue) solvers.

the large, sharp gradients that can occur in these systems. The adaptive mesh for the transonic runs showed that the high refinement near the airfoil is showcasing the steep gradients, which increased the pressure coefficient and the lift and drag coefficients.

5 Effort Breakdown

- Moon (MBH) developed the code for the Barth Jespersen Limiter (in C++) and the solution based mesh adaptation (in MATLAB). She wrote Section 1, 2.3, 2.4, 2.51 and 2.6 in the present report. She also organized the meetings for the present project at timely intervals to facilitate completion of the project milestones within reasonable time intervals.
- Elizabeth (EW) developed the Roe, HLLE, Rusanov, and wall flux functions (C++) and wrote the respective sections (2.1 and 3.1). EW simulated a majority of the tests (on Great Lakes cluster), visualized the results (in Python), and wrote the results (3.2, 3.3, 3.4, 3.5, 3.6). EW finalized the report and submitted.
- Ben (BB) wrote the first and second order solvers in C++. This included processing input and output files, integrating flux functions and limiters written by other group members, accessing geometry information, calculating residuals, time stepping, and following convergence. He also wrote sections of the report related to the solvers (Section 2.2, 2.3)
- Thomas (TG) developed and coded the MP Limiter (in C++). He wrote the sections relating this limiter in the report, as well as assisted in editing the full document. He also debugged the code, identifying and isolating several memory leaks, segfaults, and stack overflows. He also ran the finest mesh simulations when Great Lakes was unavailable.
- Vishwa (VMT) contributed to the development and debugging of the code for the flux functions Roe Flux, HLLE and Rusanov Flux (in C++). Worked with a version of MP limiter (in C++) using projection which he couldn't integrate due to approaching deadline. Further contributed to the report.

Table 4: Effort percentages.

	BB	TG	VMT	MBH	EW
development and simulation	35	10	10	10	35
coding	40	15	15	15	15
managing	15	15	15	40	15
report	20	20	20	20	20

References

[1] Finite volume method: A crash course. http://www.wolfdynamics.com/wiki/fvm_crash_intro.pdf.

[2] Timothy Barth and Dennis Jespersen. *The design and application of upwind schemes on unstructured meshes, 27th Aerospace Sciences Meeting.* AIAA, 1989.

[3] J. Blazek. Chapter 4 - structured finite volume schemes. In J. Blazek, editor, *Computational Fluid Dynamics: Principles and Applications (Second Edition)*, pages 77–129. Elsevier Science, Oxford, second edition edition, 2005.

[4] Charles Hirsch. Chapter 5 - finite volume method and conservative discretization with an introduction to finite element method. In Charles Hirsch, editor, *Numerical Computation of Internal and External Flows (Second Edition)*, pages 203–248. Butterworth-Heinemann, Oxford, second edition edition, 2007.

[5] M.E. Hubbard. Multidimensional slope limiters for muscl-type finite volume schemes on unstructured grids. *Journal of Computational Physics*, 155(1):54–74, 1999.

[6] Antony Jameson, W. Schmidt, and Eli Turkel. Solutions of the euler equations by finite volume methods using runge-kutta time-stepping schemes. *AIAA paper*, 1259, 01 1981.

[7] Hester Bijl Peter Lucas, Alexander H. van Zuijlen. An automated approach for solution based mesh adaptation to enhance numerical accuracy for a given number of grid cells applied to steady flow on hexahedral grids. *Computer Modeling in Engineering & Sciences*, 41(2):147–176, 2009.