Introduction to Computational Fluid Dynamics-- pyCFD

A Python CFD Solver – Corrado Mazzarelli

Introduction

Many industries rely upon a comprehensive understanding of fluid dynamics for their functioning. Modeling the behavior of fluids such as air, water, exhaust gases, steam, and a myriad of others is critical for the high efficiencies of modern technologies. Before the advent of computational fluid dynamics (CFD), experimental results and empirical rules reigned over a fluid dynamicist's life; however, around the 1950s computers began to have enough power to numerically solve the famous Navier-Stokes equations, which govern the motion of incompressible fluids yet lack a closed-form analytical solution. At that time, engineers in the nascent field of CFD worked to improve the efficiency, stability, and accuracy of the different methods used to solve the equations. Eventually, CFD developed to a stage where industry could adopt it, and it has become a cornerstone of modern design. CFD is now used in all aspects of industry ranging from medical devices to model blood flow within arteries[1], naval engineering to model cavitation over a submarine propeller, or the classic example of aviation for airflow over a wing. Companies such as Ansys create commercial solvers which modern engineers can use without thorough understanding of the calculations behind the scenes. To learn more about the inner machinations of commercial CFD solvers, a simple finite volume CFD solver was written in Python 3.10.10 to solve the flow field properties over the top half of a diamond airfoil experiencing supersonic flow.

Problem Formulation

Mesh creation was not part of this analysis. Three, 2D, structured, quadrilateral meshes representing the air over the airfoil were provided and read into Python. These meshes all represented the same geometry, but with varying degrees of fineness. The medium density mesh is displayed in Figure 1, and is the only relevant mesh for this analysis.

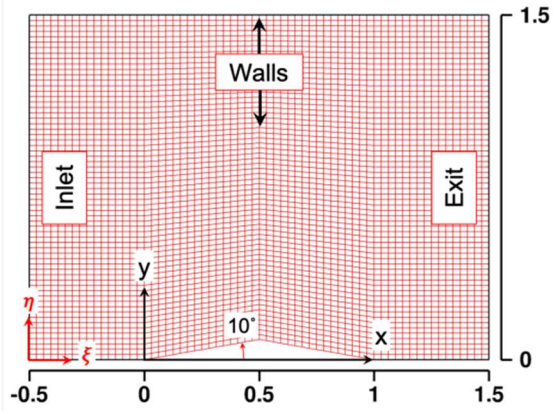

Figure 1: Medium Density Mesh from Project Problem Statement

As visible in the bottom left of Figure 1, in addition to the standard x, y Cartesian coordinate system, a generalized, body-fit "ξ, η" coordinate system was also specified. This is the generalized coordinate system referenced below. Certain cell area metrics were required to characterize the mesh and calculate relevant fluxes and time-steps for each cell, as will be discussed. These cell area metrics, including projected cell face area and cell volume (in 2D they are projected side length and cell area), were

[1] Note: the Stokes Hypothesis underlying the NS equations does not hold for non-Newtonian fluids such as blood.

calculated as described in the Topic 24.2 notes. To finalize the mesh preprocessing, halo cells needed to be added to allow for boundary condition enforceability. The halo cells are visible in Figure 2, outside of the black outlines.

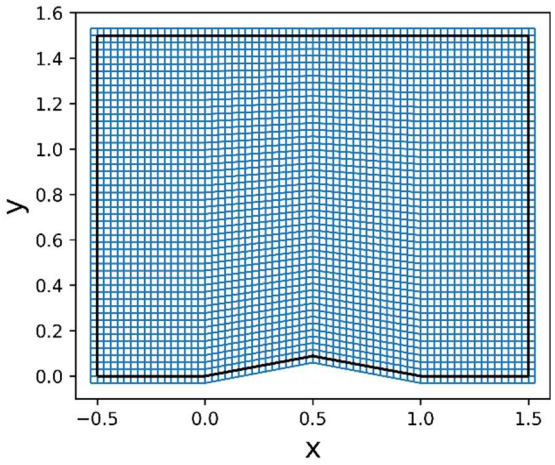

Figure 2: Medium Density Mesh with Halo Cells

To initialize the solution vector, the initial conditions specified in the project statement of Mach 2 horizontal flow at 300K and 101.325 kPa were used. That initial condition was also used as the constant inlet boundary condition, which was re-enforced after every iteration of the solver. The outlet boundary condition was enforced using first order extrapolation, per the problem statement. The boundary condition along the top wall, and the bottom walls that did not include the airfoil was an inviscid, adiabatic, slip-wall condition, representing streamlines in the freestream since no mass flow crosses through them. This was applied by solving the system of equations specified in the Topic 24.4 note and applying those conditions. The airfoil boundary conditions depended upon what case was attempted, out of the 5 specified in the problem statement. For this analysis, an adiabatic slip-wall condition was also applied to the airfoil, which is non-physical. The capability to enforce an adiabatic no-slip wall condition was also coded and will be discussed in the results section.

To model the flow field over the airfoil, the Navier-Stokes equations needed to be numerically solved, and to facilitate this, they needed to be written in a format that would allow easy coding. Thus, the equations were rewritten in standard vector form as described by the Topic 3 class notes, using generalized coordinates per the Topic 4 notes. Note, that while it is possible to use non-dimensional forms of the NS equations, all terms and equations in this analysis were dimensional in standard SI units. Once in this format, summarized by Eqn. 1 in the Topic 24.1 notes, either a finite volume or finite difference method could be implemented after combining the N-S equations with the ideal gas law to relate different states of the gas and make the system of equations determinant. For simplicity, the solver was first implemented without any viscous terms, and all terms referred to henceforth are convective. For this solver, a finite volume method was chosen to ensure that the discontinuity at the shockwave could be captured, since finite difference methods assume continuous differentiability. To ensure that the system of equations was stable, the implemented solving scheme needed to be ‘total variation diminishing,’ which essentially prevents non-physical oscillations from growing and propagating in the solution. A scheme with this property called the “Monotone Upwind Scheme for Conservation Laws” or MUSCL scheme, was implemented. A MUSCL stencil which allows for selecting

different orders of interpolation accuracy was chosen, as described in Equations 11 & 12 in the Topic 25 notes.

Fundamentally, a finite volume solver stores values at cell centers (in a mesh) and interpolates gas state values to cell faces to approximate the flux over each face in a cell. Then, taking the integral of all the fluxes across the surfaces over a given timestep dictates how the gas state values in that cell should change. Thus, to implement the solver the fluxes over each face of the mesh needed to be calculated and stored. For this solver, the Roe flux vector differencing (FVD) scheme was selected to calculate the fluxes, as opposed to a flux vector splitting scheme such as Steger-Warming. The Roe FVD scheme uses 'Roe average' values of the primitive variables such as pressure, temperature, density, etc. to account for the discrepancy seen when interpolating the solution vector at a given cell face from either the left or the right adjacent cells. These Roe averaged terms, whose formulas to calculate them from the left and right state primitive variables are located at the end of the Topic 23 notes, were derived such that they exactly conserve mass, momentum, and energy across the cell interfaces and satisfy the Rankine-Hugeniot relations when a shock is present [Topic 23]. With the Roe averaged values, and the knowledge that the flux vector is homogenous with respect to the solution vector Q, the flux vector can be represented as a product between its Jacobian with respect to Q, and Q. Identifying this Jacobian, denoted A, was integral in enabling a stable solution, since the Jacobian could be diagonalized into the product of L1, Lambda, and R1, where L1 and R1 are the left and right eigenvectors of A and Lambda is a diagonal matrix of the corresponding eigenvalues. The analytical forms of these matrices are shown in the Topic 26.1 notes, and they were incorporated into the code exactly as written. With the eigenvalues exposed in Lambda, they could be manipulated such that regardless of if the flow is supersonic, subsonic, or flowing left or right the scheme would always be 'upwind' and thus not unconditionally unstable. The final equation to calculate the fluxes per the Roe FVD scheme is Eqn 8 in the Topic 25 notes, which was implemented in both the ξ and η generalized directions.

Once the fluxes were calculated, the maximum allowable time-step for each cell was calculated per the procedure in the 24.3 notes. Since this was a steady-state problem and not a transient, time-accurate problem, local time-stepping could be used, which means each cell is changed by the maximum amount possible while remaining stable. Finally, with the fluxes and the time-steps all calculated, Eqn 1 from the Topic 24.1 notes was used to progress the solution towards steady state. This was repeated until the L_2 and L_∞ convergence metrics approached machine accuracy, which for Python is 1e-16 since Python is in double precision by default.

For the higher order schemes, the values of epsilon and kappa in the MUSCL stencils described by Equations 11 & 12 in the Topic 25 notes were modified. All the schemes possible with the different combinations of epsilon and kappa were attempted, but only the second order fully upwind scheme and the third order QUICK scheme converged without a flux limiter. The epsilon and kappa values to achieve those two schemes are (1, -1), and (1, 1/2), respectively. Several the flux limiters described in the Topic 18 notes were implemented and ran through a DoE to determine stability. However, none were stable in the original rudimentary implementation, and further reading of the Topic 25.1 and 25.2 notes would be necessary to implement flux limiters based on the velocity or pressure of the flow.

Results and Discussion

Figures start on the next page.

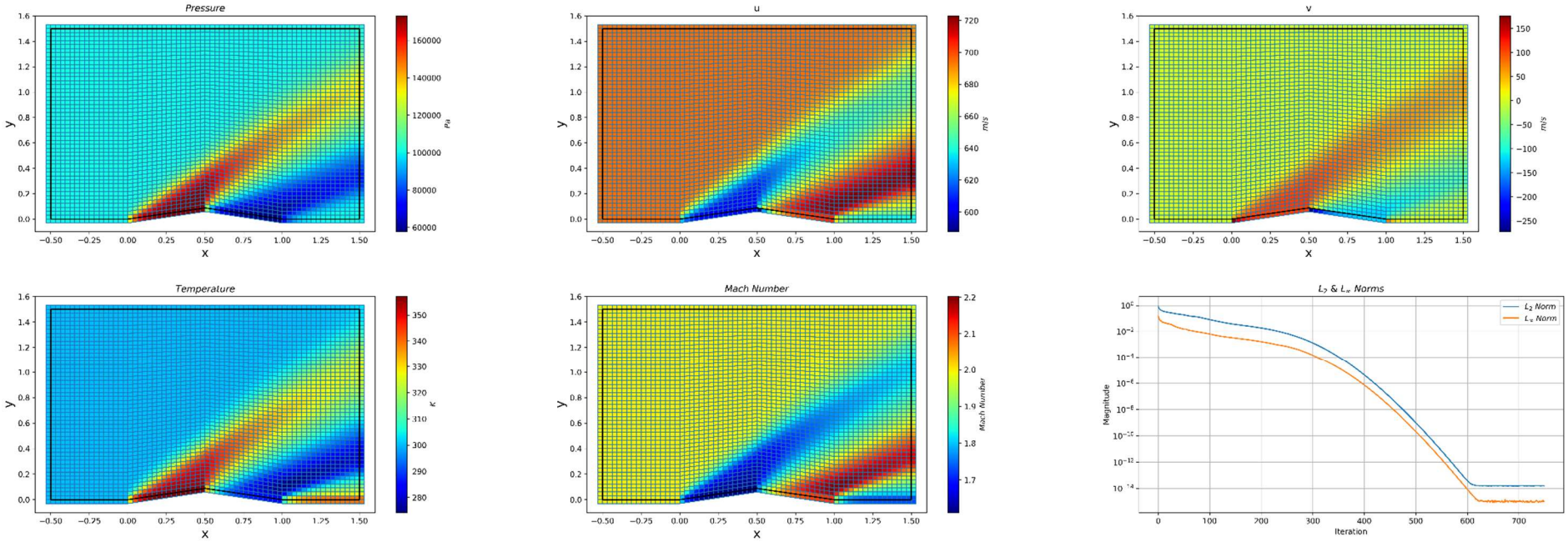

Figure 3: First Order Accurate Solution

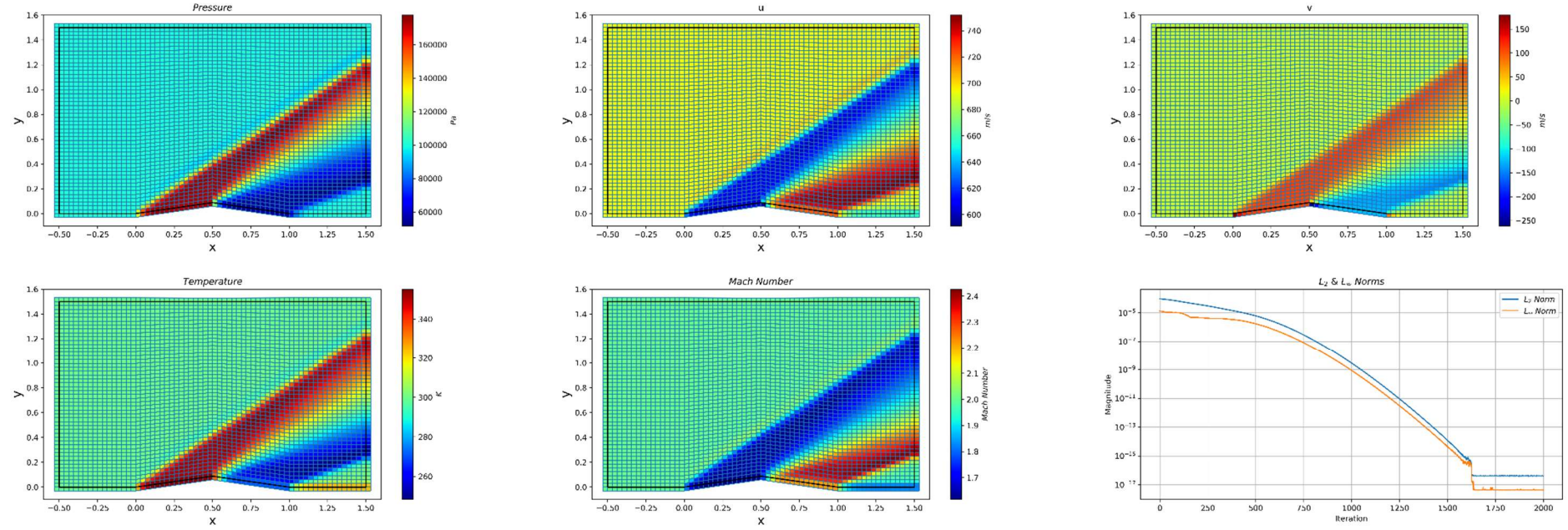

Figure 4: Second Order Accurate Full Upwind Scheme Solution

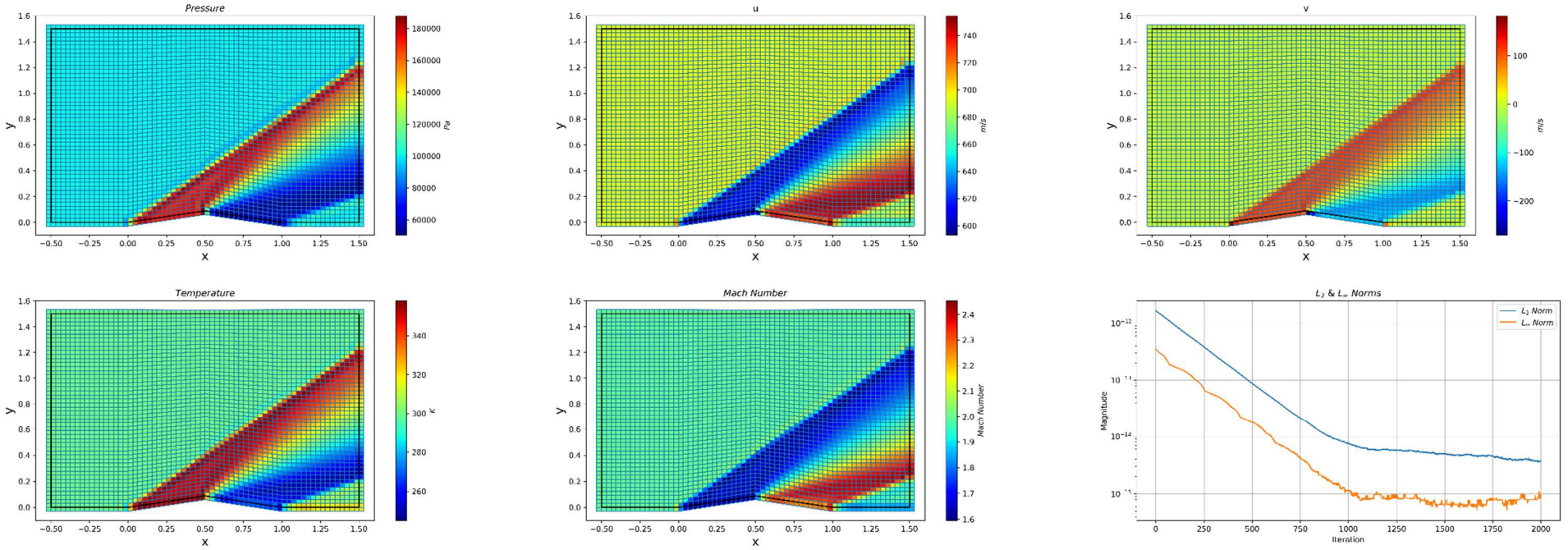

Figure 5: Third Order Accurate QUICK Scheme Solution[2]

Table 1: Comparison Between Numerical and Analytical Solutions for the Oblique Shock

	Wave Angle	M2	P2/P1	$\rho 2/\rho 1$	T2/T1	$P_t 2/P_t 1$
Analytical Solution	39.31	1.641	1.707	1.458	1.170	0.9846
1st Order Solution	35	1.632	1.693	1.441	1.175	0.9642
Percent Error	-11%	-0.557%	-0.829%	-1.147%	0.389%	-2.073%
2nd Order Full Upwind Solution	34	1.641	1.710	1.462	1.170	0.987
Percent Error[3]	-13.5%	0.005%	0.173%	0.283%	-0.043%	0.286%
3rd Order QUICK Solution	35	1.647	1.698	1.455	1.167	0.9891
Percent Error	-11%	0.365%	-0.540%	-0.210%	-0.263%	0.455%

[2] Very high-fidelity versions of these plots are included in the supplemental materials. They can be zoomed in without loss of quality.

[3] Percent errors were calculated before rounding

The three cases that will be discussed include a first order accurate upwind solver, a second order accurate fully upwind solver, and a third order accurate QUICK scheme, all without flux limiters, assuming inviscid flow, and with an adiabatic slip wall condition, as shown in the above plots

First Order Solution

The first order solution for the flow field, as visible in Figure 3, is highly dissipative and smears the shockwave dramatically after it initially forms at the leading edge of the airfoil. First order accuracy is generally required around a shockwave to handle the large gradients at that point; however, since it is only first order accurate, the solution gets smeared as the high dissipation of the scheme acts like an artificial viscous term and tries to remove large gradients. Note, in the convergence plot of Figure 3, the solution converged at around 600 iterations, which is much faster than the other two schemes. The first order solver was stable with a maximum CFL of 0.97 compared to a CFL of approximately 0.1 for the other two schemes, which translates to a larger timestep and faster convergence. Also visible in the convergence plot, the residuals only decayed to around 10^{-14}, which is slightly larger than machine accuracy. It is unknown why this scheme would not converge to machine accuracy, but the other ones would get closer. Table 1 shows a comparison between the analytical, ideal-gas, compressible-flow, oblique shock solution, and the numerical solutions achieved with the various schemes. Looking at the data for the first order scheme, everything except the shock angle is accurate to within $\pm 2\%$, measuring the post-shock values at Cartesian coordinates of (0.3, 0.1). This accuracy is encouraging, but that is a highly local phenomenon. As visible in the plot, as the shockwave continues out past the airfoil it rapidly decays from the ideal analytical solution. In reality, the shockwave would eventually dissipate, but not as quickly as shown in the figure. Thus, while the numerical first order solution is accurate close to flow disturbance (airfoil), the solution artificial viscosity of the solution leads to non-physical results above the airfoil. The second major inaccuracy here is the wave angle, which is 11% lower than the analytical solution. This inaccuracy occurred for all schemes, and it is unknown why this happened, especially since the rest of the results are much more accurate. Thus, caution must be taken using this CFD solver if the wave angle is important to the analysis, such as when designing supersonic inlets with shock reflections.

Second Order Solution

The second order solution to the flow field is much less dissipative than the first order solution. The shock wave continues out to the end of the mesh, and a more defined wave front is visible within the shock itself. The convergence plot in Figure 4 shows that the second order scheme converged to ~1e-16 accuracy in about 1700 iterations. It is unknown why this scheme was able to converge to a greater level of accuracy than the first order scheme. The convergence speed of this scheme was much lower because, as previously mentioned, the CFL was dropped by around an order of magnitude to make the scheme stable. The solution to this scheme was seeded with the results from the first order solver to facilitate quicker convergence and ensure that the initial conditions were not too far from reality such that the solver could not handle it. Given that the CFL needed to be lowered to ensure stability, the second order solution is inherently less stable than the first order solution. This stability issue is slightly remedied with flux limiters, which were briefly touched upon in this analysis. Compared to the analytical solution, this was the most accurate solver out of the three attempted. Excluding the shock angle, the solution was accurate to within 0.3%, measured at the same cartesian coordinates. Furthermore, this accuracy likely extends out further past the airfoil since the solution is much less dissipative than the first order solution and the shock extends to the bounds of the mesh instead of

decaying. The shock angle, however, is again off by >13%, indicating the issue seen with the first order solution is not exclusive to the first order solver.

Third Order Solution

The third order QUICK scheme solution is like that of the second order scheme. Again, the shockwave continues to the mesh boundaries and a very defined shock front is visible in the plots, even more defined than the second order solution. The convergence plot in Figure 5 indicates that the solution converged to the order of 1e-15, which again is slightly off machine accuracy, and different than the convergence values for both the first and second order solutions. The CFL was again set to 0.1 to ensure stability. The third order solver was seeded with the first order solution to speed up convergence, and run multiple times, so the true convergence time was likely around 5,000 iterations, not 1,000 as the convergence plot demonstrates. As was the case with the previous two schemes, the wave angle has the largest deviation from the analytical solution, at 11% lower. However, the other values are all within 0.5% of the analytical solution, again indicating the accuracy of the solution. The third order solution is slightly less accurate than the second order solution at the coordinates measured. It is possible that this is an artifact of where the post shock conditions were sampled.

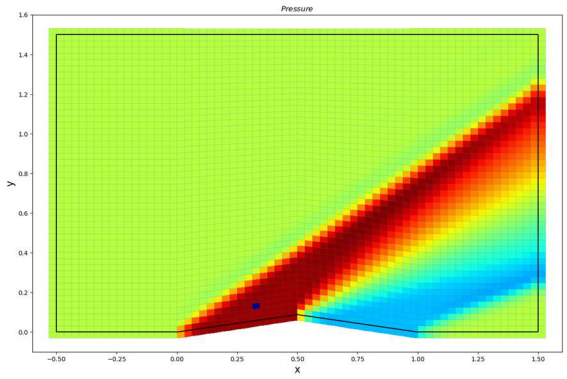

Figure 6: 2nd Order Post-Shock Property Sample Location (x = 0.3, y = 0.1)

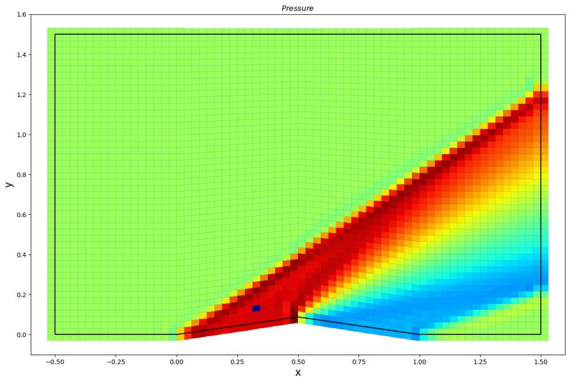

Figure 7: 3rd Order Post-Shock Property Sample Location (x = 0.3, y = 0.1)

Figure 6 shows the cell from which post shock properties were sampled in dark blue with the second order scheme as the background, and Figure 7 shows the same cell but with the 3rd order scheme as the

background. As evident from the figures, the second order scheme seems to have a less well defined shock front, since the dark blue cell is located within the larger dark red area. However, for the third order solution, the shock front is very well defined as the dark red area and then the rest of the space immediately behind the shock is a lighter red. Given how close the solutions are to the analytical solution, the 0.5% margin of error for the second or third order solution could be explained by choosing a different measurement location after the shock.

Minmod Flux Limiter

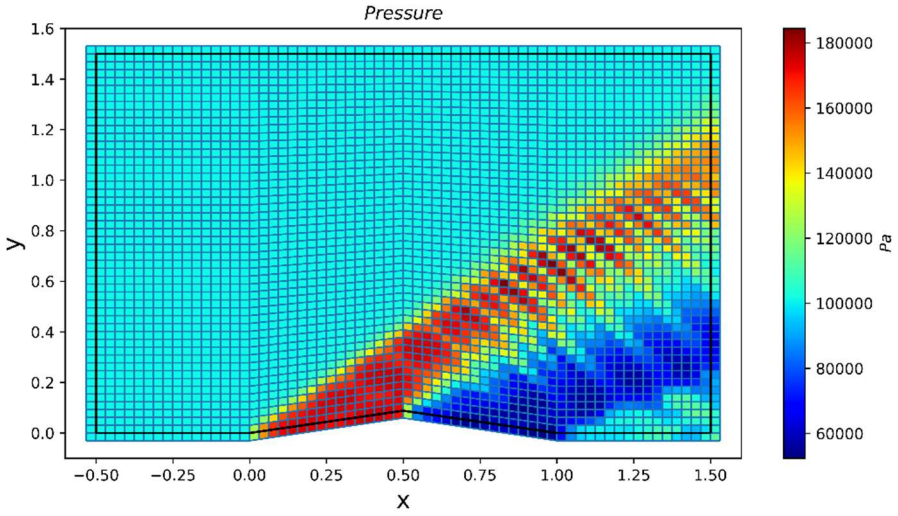

Figure 8: 2nd Order Accurate Solver with Minmod Flux Limiter

Flux limiters attempt to enforce the total variation diminishing property in higher order schemes. A basic minmod flux limiter was implemented in the second order scheme, with the results shown in Figure 8. As can be seen, this solution was oscillatory in nature, and the convergence plot (not shown) settled on residuals on the order of 1 and oscillated there. While not successful as a solver, this was an excellent visualization of non-physical, numerical artifacts within the solution borne as a result of the solving scheme and not the underlying physics. The pressure waves propagating in Figure 8 would likely be fixed by implementing the flux limiter as described in the Topic 25.1 and 25.2 notes, but this was not done due to time limitations.

No-Slip Boundary Condition

The no-slip viscous wall boundary condition can be applied to the airfoil in the code as an option at runtime. However, this led to convergence issues for the higher order solvers. The first order solver converged with a no-slip wall condition, but due to the artificial dissipation of the solution, and the lack of real viscous terms, the difference adding the no-slip viscous boundary condition was negligible, as shown in Figure 9.

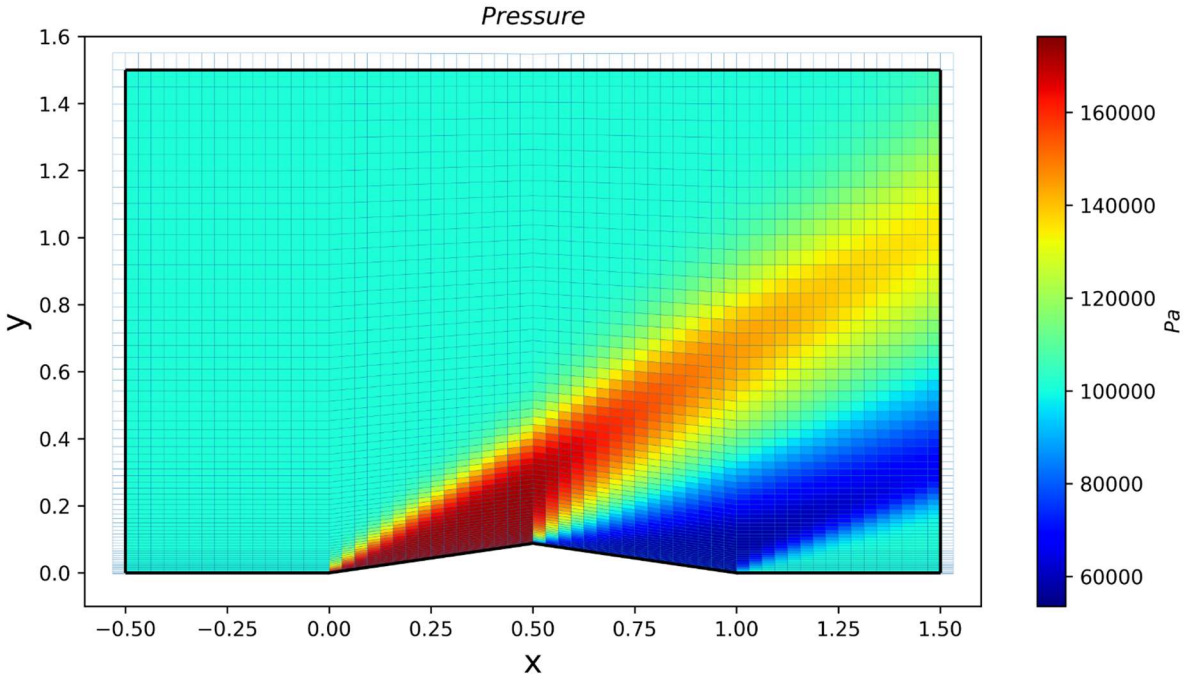

Figure 9: First Order Solution on the High Density Mesh with a No-Slip Airfoil Wall

The results of a no-slip wall and including the entire NS equations, not just the simplified Euler equations, should be explored further in future work.

Conclusions

Of the three solvers created, the second and third order accurate solvers were both of comparable accuracy within the given mesh domain compared to the analytical solution. The first order solver was highly dissipative, decaying and smearing the shock within the small domain of the mesh. The second and third order solvers propagated the shock with a defined front, out to the mesh outlet. The third order solver had the highest fidelity shock front. For a better comparison of the second and third order solvers, a larger domain should be generated and solved, such that the larger effects of the solvers can be seen. All the solvers had issues predicting the shock angle, underpredicting the analytical angle of 39.31 degrees by around 11%. It is unknown why all three solvers underpredicted the wave angle so dramatically, yet were within 2% of the expected post-shock conditions for a turn angle of 10^{o}.

The solvers shown were all without flux limiters and for inviscid flow, solving the Euler equations. Future work should expand the Euler equations to the full Navier-Stokes equations, modeling the flow viscosity as well. This would enable the no-slip boundary condition to apply, which is a physical condition enforced by viscosity. Furthermore, the higher order schemes require flux limiters to enforce the TVD property, which would enable more than just the second order full upwind and the third order QUICK scheme to be stable, such that their results could be compared as well.

Overall, pyCFD was a good learning experience for the author, requiring a strong grasp of Python coding principles to ensure that the code would run in a reasonable amount of time and be easily refactorable and comprehensible.