***Potential Flow – Transonic Small Disturbance***

I. Problem Statement and Theory

*I.A Problem Statement and Potential Flow*

This work attempts to solve the steady aerodynamics over an airfoil in transonic regimes. The goal is to capture local supersonic flow and the associated sonic and shock discontinuities in the flow through numerical solutions to the small disturbance potential equation. First, we must develop the governing equations from the full velocity potential equation by assuming mass continuity. In cartesian coordinates:

A math equations with numbers

AI-generated content may be incorrect.

This equation must first be closed by substituting for density. By considering inviscid momentum (first), irrotational effects (second), and barotropic fluid approximations, we yield the final equation below(third) in order as listed:

A black and white text

AI-generated content may be incorrect.

A black text with a white background

AI-generated content may be incorrect.

A mathematical equation with symbols

AI-generated content may be incorrect.

Finally, we assume that the gas is a perfect gas and the flow is isentropic to eliminate pressure and the speed of sound from the equation above. The definition of density is now:

A black text on a white background

AI-generated content may be incorrect.

*I.B Assumptions*

The assumptions made in the flow shape the domain of the solution’s applicability. This flow is inviscid, so we ignore fictional effects, which for low Mach number flows (0.5-1.3) and for attached flow, is reasonable. The flow is irrotational, which lends itself appropriately to flow over an airfoil, but caution must be taken on constructing the wake. Finally, the flow is assumed isentropic, which is physically unrealistic over a shock, but for low Mach numbers, a reasonable approximation because very little entropy is produced. This bounds the domain of applicability to low-alpha, M = 0.5 to ~1 flows over 2D airfoils.

*I.C Transonic Small-Disturbance Derivation*

Now, we must use the full potential equations to derive a PDE that solves the transonic flow. First, we define a perturbation to the flow from freestream:

A group of math symbols

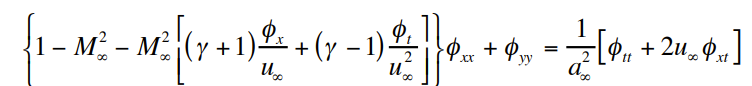
AI-generated content may be incorrect.

This solver will then solve the disturbance velocities. Now, with a definition of density from I.A, we expand and combine the full potential form from the beginning of the article to write the following equation (note the dropped prime notation for convenience).

A math equations and formulas

AI-generated content may be incorrect.

We now assume cross-derivative terms (e.g are small and that the temporal terms are neglected to form the steady-state equation. Below is the first intermediate step, followed by the equation to be solved:



A math equation with black text

AI-generated content may be incorrect.

The RHS of the first equation will vanish along with the term, yielding the transonic small disturbance potential equation. This equation will solve for the perturbation velocity, or disturbances, from freestream, rather than the exact velocity. Similarly, the equation is nonlinear, allowing for a transonic shock (sonic line or shock) in the solution space. As such*, there is no exact solution to determine transonic behavior, thus requiring a numerical solution*. The next section will detail boundary conditions and numerical implementation to solve the problem.

II. Numerical Algorithm

*II.A Boundary Conditions*

The initial flow conditions are set to the free-stream flow everywhere, . The left boundary condition is the freestream, and the transverse flow at the airfoil surface is imposed as zero upstream and downstream from the airfoil (invoking no vorticity assumptions). At the airfoil surface, tangency is written with , where (dy/dx) is the local slope along the body. At the top, if no wind tunnel approximation is being made and if a wind tunnel approximation is assumed. The next section details the discretization scheme where such boundary conditions will be applied.

*II.B Discretization and Transonic Handling*

First, we examine the characteristic PDE to determine zones of dependence and influence; this will help guide the discretization of the scheme.



Setting and M < 1, writes the above as an elliptic equation. Therefore, the zone of influence is everywhere in space spreading from a single point. For subsonic flow, the space may then be discretized with a central scheme. Conversely, if M > 1 and we set , then the above produces a hyperbolic PDE, where the zone of dependence is upwind and influence, downwind. In supersonic flow, an upwind scheme must be deployed to capture the wave-like nature of the sonic line. For this regime, we may march in the x-direction to match the physical behavior of the sonic line. The y-direction does not experience this Mach-dependence, and thus, a central scheme may be deployed everywhere. Using a central scheme in the subsonic points and an implicit scheme in the sonic points, the following stencils are used:

A diagram of a subsonic stencil

AI-generated content may be incorrect.A diagram of a number of lines and dots

AI-generated content may be incorrect.

Now, we must conduct a scheme that can ‘switch’ between the two stencils and solvers based on the local flow speed. Let’s define a nonlinear coefficient on the PDE to determine if the flow is locally super or subsonic, and therefore, if the equation is hyperbolic or elliptic.

A black and white image of a smiley face

AI-generated content may be incorrect.

A group of math equations

AI-generated content may be incorrect.

Therefore, we can write the subsonic and supersonic regimes as:

A math equation with numbers and symbols

AI-generated content may be incorrect.

A math equation with numbers and symbols

AI-generated content may be incorrect.

These are combined with a ‘switch’ such that or to write the following implicit time marching scheme:

A math equations and formulas

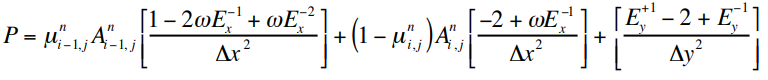
AI-generated content may be incorrect.

To fit this into a traditional tri-diagonal solver that directly combines delta phi for an update, the following P and residual operations are defined



A math equations with numbers

AI-generated content may be incorrect.



This residual term, L, should shrink to zero with continued iteration to appropriately capture the steady-state assumption, . In this residual term*,* ***if A > 0, the flow is subsonic, and the 2nd term will vanish***from the switching term. Only the central scheme will remain. Conversely, ***if A < 0, the flow is supersonic and the switching term will cause the first term to vanish***, leaving the upwind scheme intact. The y discretization is preserved in both cases. Finally, these equations may be iterated under classical relaxation schemes to solve the steady state 2D, transonic flow. The next section will detail the iteration process.

*II.C Iteration Scheme*

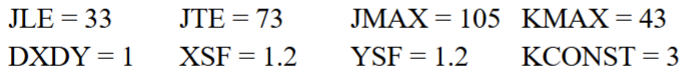
This simulation iterates with a SLOR scheme. Because the solution is swept in the x-direction (as the y-direction sees no direct Mach dependence), the entire vertical line may be relaxed at one time. In SLOR, a relaxation parameter, , multiplies the residual to converge the solution by working to dampen low frequency errors much faster than without the term. This manifests above with the term in front of the L operator. For our problem, where the grid is uneven and nonlinear terms exist, no exact exists, but appears to converge quick. The caveat, however, is that the relaxation factor may only be applied to central schemes, or the subsonic flow, in this problem. When the flow is supersonic, to avoid instability issues. This regime will converge slower than the subsonic regime. To iterate, a tri-diagonal solver determines the delta from the P-L equation above, which then performs the update.

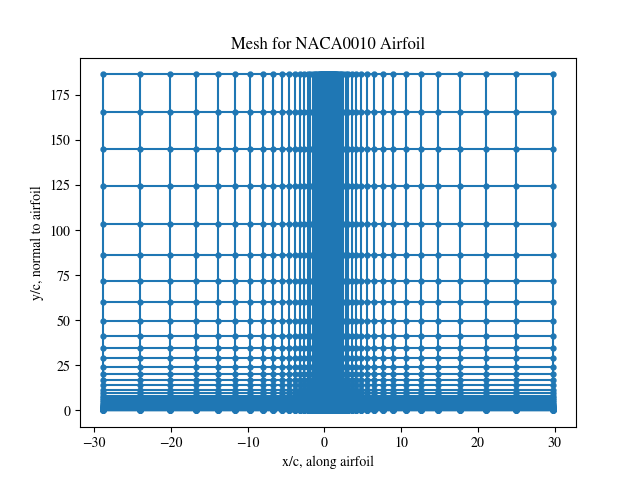
III. Implementation

All results are performed on an 11th Gen Intel i5, 2.40GHz processor.

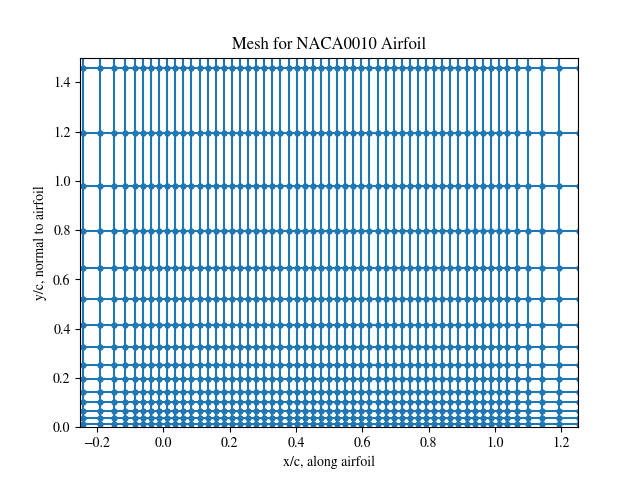
*III.A Software Implementation*

The discretization and iteration scheme were first implemented in MATLAB, building on an existing code that solved the subsonic linear small disturbance equation. When the results appeared close to expected, the results were the piecewise translated to python 3.8. This process was done stepwise: first, the meshing was implemented and checked that it matched the MATLAB implementation. Here, the mesh is constant over the airfoil, then stretched by a stretching factor away from the airfoil. The following parameters were used to create the mesh below:





*Fig 1: Constructed ordered mesh to solve the transonic problem*



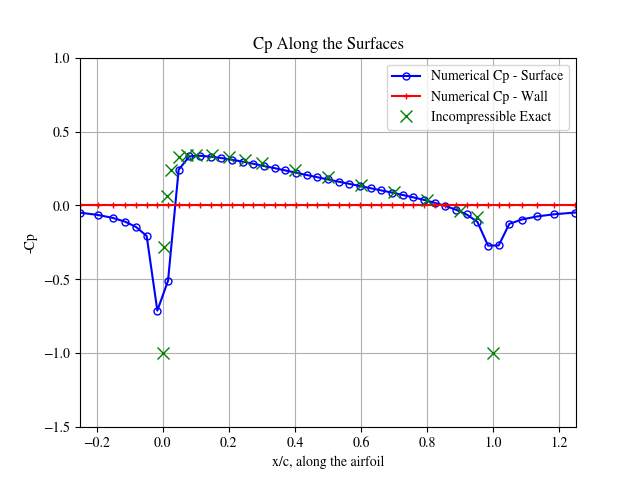
*Fig 2: Detailed view of fig 1*

Next, the tridiagonal solver was coded up and the SLOR relaxation scheme was implemented. Again, working step-by-step, I first determined phi at the boundary conditions and confirmed that they matched the expected value. Then, I coded up one iteration of SLOR and confirmed that the resulting phi and residual matched the MATLAB implementation of the transonic case.

Finally, the mesh generation and tri-diagonal solver were split into their own subroutines that are called by the script. The main body of code, then, is just the transonic flow solver.

*III.B Verification*

To confirm that the results match the known subsonic case, I compared the results to existing work I conducted. By setting gamma = -1, the subsonic LSD case may be recovered for an effective comparison. Gamma = -1 as simulated for 500 iterations, plotted below. The incompressible exact solution from Abbott and von Doenhoff was overlaid to verify the results:



*Fig 3: Incompressible verification case results*

The results appear in good agreement! The code thus far seems to be accurately implemented. In addition, I implemented a condition to stop if the residual drops below 0.01% of the original value. Finally, I have the code print the residual every 50 iterations to confirm convergence. Next, I will explore the transonic flow.

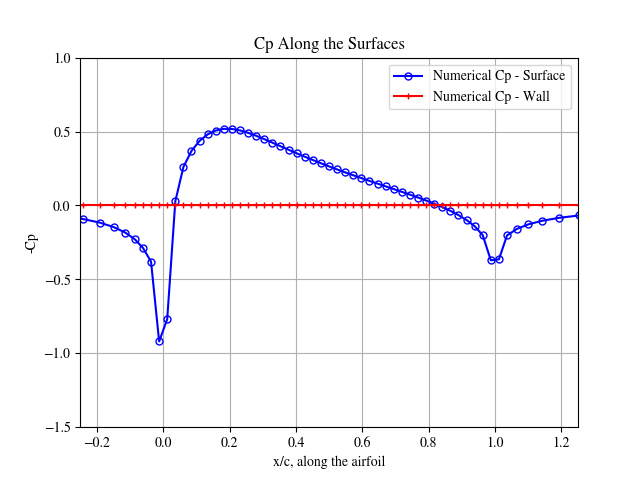
IV. Results, SLOR Analysis, and Discussion

*IV.A Set-up and Initial Result:*

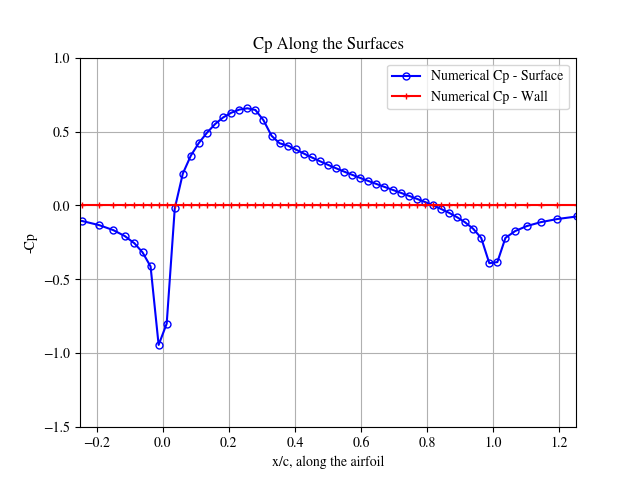
The code was then initialized with the mesh above. 4 cases are initially explored: two airfoils and 2 freestream Mach numbers. The simulation details are reported first in the table below. The pressure coefficient along the surface is then tracked and plotted for the 4 cases.

*Table 1: Simulation cases run and the computation details*

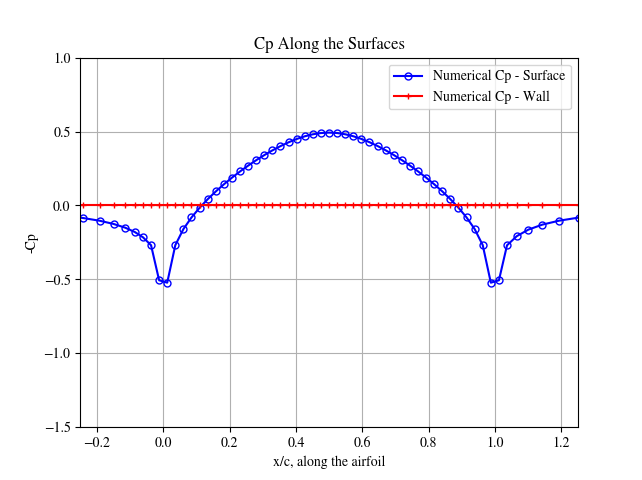
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Simulation # | Airfoil Type |  | CPU Time (sec) | Supersonic Points | SLOR Iterations |
| 1 | NACA0010 | 0.75 | 91.33 | 0 | 1000 |
| 2 | NACA0010 | 0.80 | 77.36 | 21 | 1000 |
| 3 | 10% Biconvex | 0.75 | 132.63 | 0 | 1000 |
| 4 | 10% Biconvex | 0.80 | 141.11 | 32 | 1000 |



*Fig 4: Case 1 results*



*Fig 5: Case 2 results*



*Fig 6: Case 3 results*

A graph of a graph with a red line

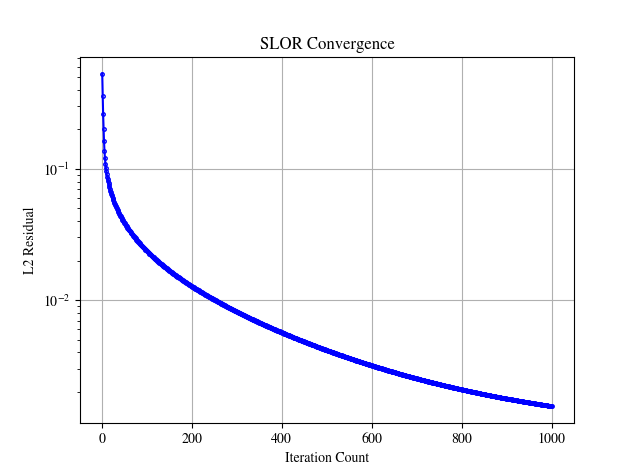
AI-generated content may be incorrect.

*Fig 7: Case 4 results*

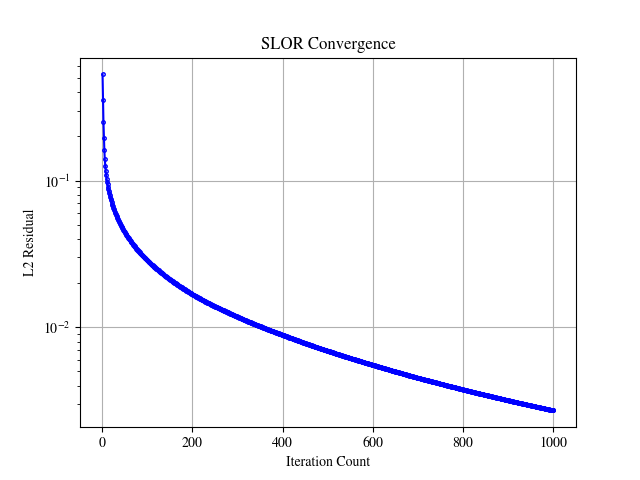
Initial discussion: the code appears to capture the transonic shocks! In M = 0.8 cases, a sonic and shock line are clearly identifiable. These results also appear to corroborate existing literature. NASA technical memorandum 86320 “Effects of Airfoil Shape, Thickness, Camber, and Angle of Attack on Calculated Transonic Unsteady Airloads” simulated similar cases to the ones in this report[[1]](#footnote-1). Shown in fig 4 of their report, the calculated Cp values match mine almost identical. Interestingly, the biconvex case took longer to solve than the NACA airfoil case!

*IV.B Convergence*

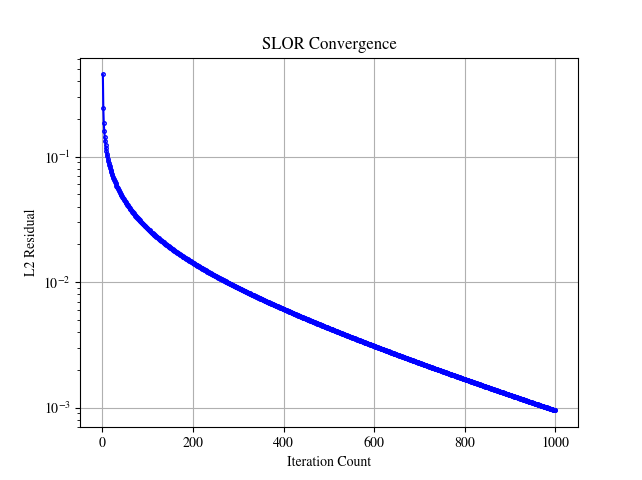
Now, the convergence is tracked for each computation. Because this is a steady-state problem, the residual should converge to or approach 0; the efficacy of the solution may then be determined by plotting the L2 of the residual over iteration count. Following the same cases in table 1, the convergence plots are below:



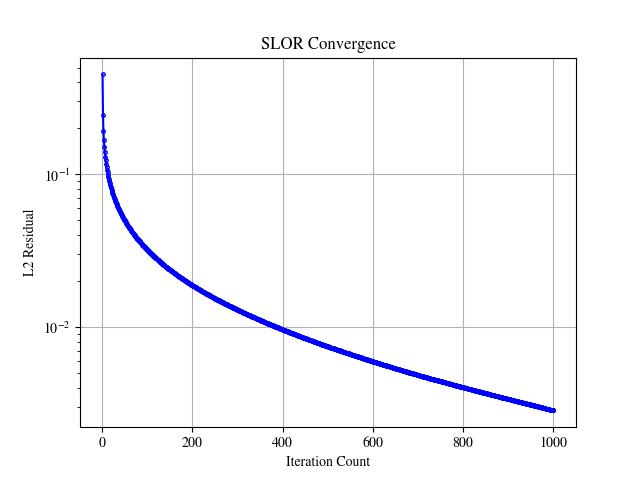
*Fig 8: Case 1 convergence results*



*Fig 9: Case 2 convergence results*

**

*Fig 10: Case 3 convergence results*

**

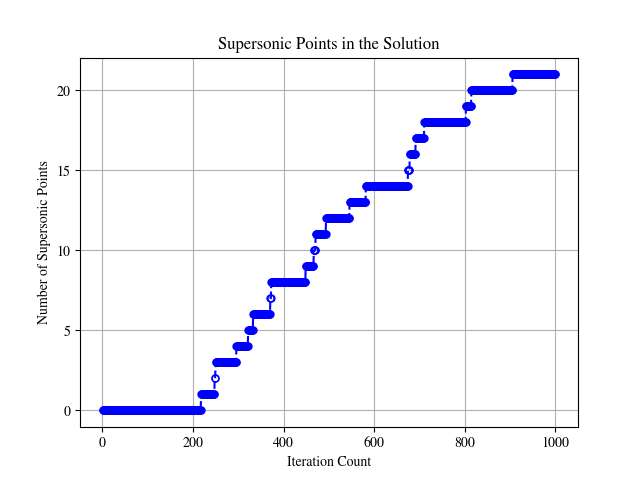
*Fig 11: Case 4 convergence results*

Discussion: The convergence results are strikingly similar between all 4 cases. They each converged to around 0.001, dropping 1 log in the first 10 counts, and another log in about 400 counts. The latter convergence in the biconvex cases appears slightly slower than the NACA airfoil cases. In all, the simulation dropped 3 logs in about 1000 iterations. These results are about what I expect – the subsonic regions that can use the SLOR parameter converge fast as shown in the early iteration count, but the supersonic regions are slower, leading to a longer convergence tail. This corroborates plots in the next section, too.

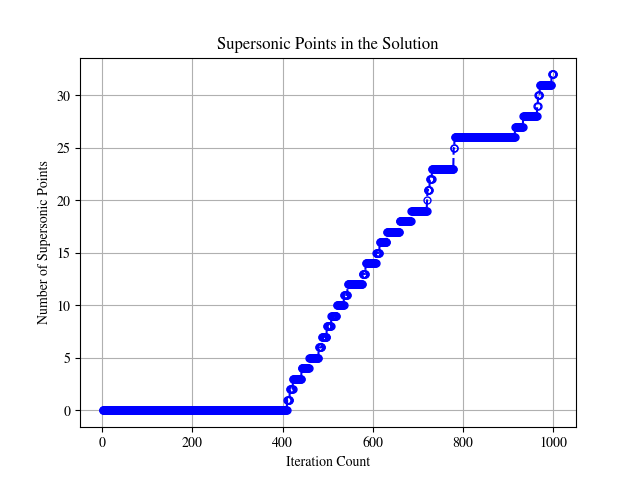
*IV.C Supersonic Point Count*

For M = 0.8 cases, the number of supersonic points in the solution space are tracked and plotted to build a better intuition on how well the solution is developing supersonic points. Figures found on next page for convenience.

Discussion: The NACA airfoil sees supersonic points develop faster in the simulation compared to the biconvex airfoil by about 200 iterations. Further, when the biconvex airfoil does develop supersonic points, it does so at a faster rate compared to the NACA airfoil. In fact, the biconvex airfoil looks like it could continue to develop additional supersonic points, and I would recommend running the simulation longer.



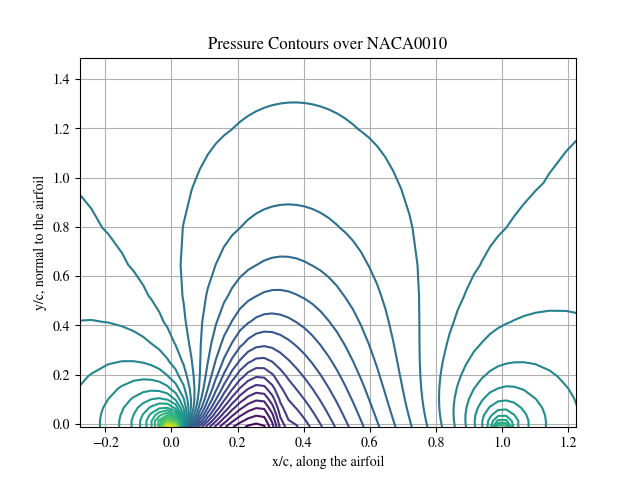
*Fig 12: NACA 0010 airfoil, M = 0.8 supersonic point count*

**

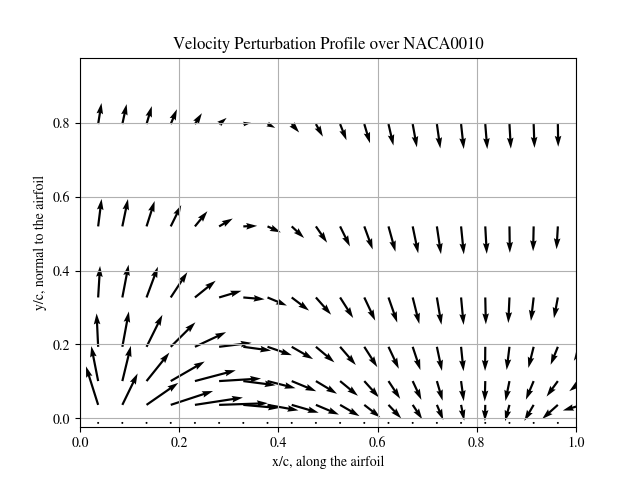
*Fig 12: 10% thickness biconvex airfoil, M = 0.8 supersonic point count*

*IV.D NACA0010, = 0.8 Solution Details*

For further details of the solution, I plotted the Cp pressure contour lines and the velocity perturbations over the airfoil. The velocities were found by taking a 2nd order, 1st derivative of phi in both the x and the y at each mesh point. Note that only every 3 points are plotted in the figure below.



*Fig 13: Pressure contour of the transonic flow over the NACA0010 airfoil*



*Fig 14: Perturbation velocity field over the NACA0010 Airfoil*

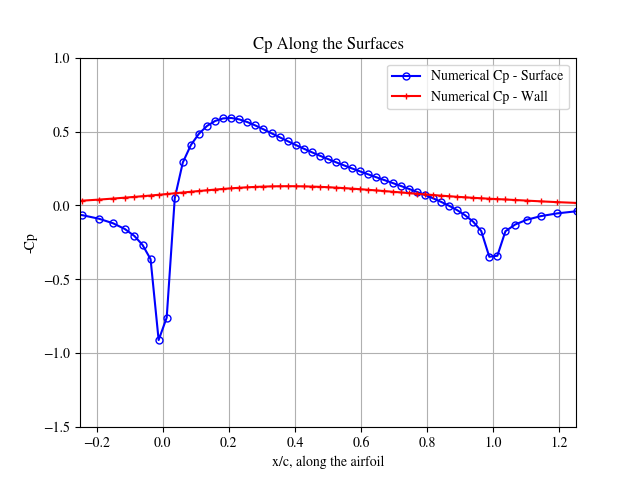
Discussion: The lowest points in the pressure contour (deep purple colors) are where we expect to see the largest, supersonic flow. This is reflected in the large arrows around the supersonic point in the velocity field. Together, these results are what I expect.

*IV.E Exploration Cases*

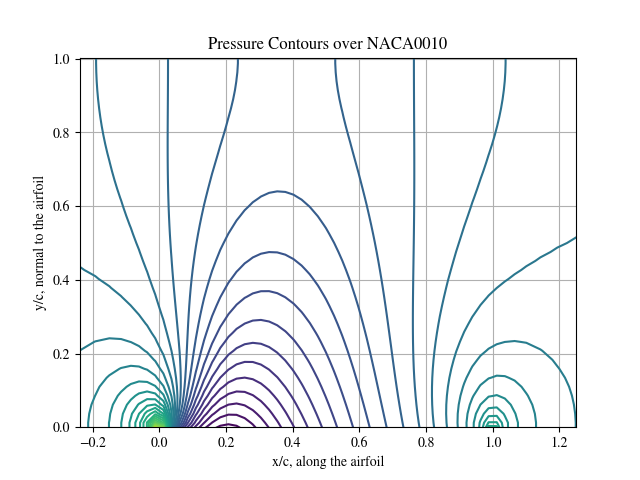
To test the limits of this implementation, I ran two extended cases:

* M=0.75 over an NACA0010 airfoil in a wind tunnel
* Modified stretching factors and meshing at M = 0.8

First, I took the existing implementation and added a wind tunnel constraint, where phi at the top must be 0, as opposed to a freestream boundary condition as used throughout the report thus far. Cp along the surface is first plotted, then the pressure contour above the airfoil is shown. The figures show expected results: the pressure contours at the wind tunnel wall are normal, reflected in the elevated calculated Cp along the wind tunnel upper wall. Interestingly, the pressure along the airfoil is reduced! We would expect the lift generated in the wind tunnel to be less than freestream. Interestingly, the simulation took nearly twice as long to solve.

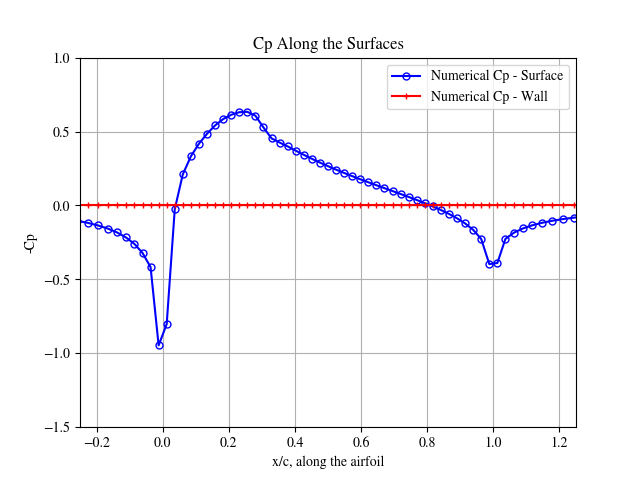


*Fig 15: Cp over a NACA0010 Airfoil in a Wind Tunnel at M = 0.75*



*Fig 15: Pressure contours NACA0010 Airfoil in a Wind Tunnel at M = 0.75*

For the stretching, I re-ran case 2 (back to assuming freestream at the upper boundary), but cut the stretching factor down to 1.05 in both the x and y directions. I want to investigate the impacts on the Cp solution and the number of supersonic points found. The Cp results are below. Interestingly, the solution converged slightly slower than with the 1.2 stretching factors, and there were a few less sonic points in the solution space. The shock looks slightly diminished as well. Together, this makes sense, because the mesh spacing around the airfoil was kept identical (thus, the discretization accuracy should be the same there), so upwind effects and changes to accuracy far above the airfoil should be the main driver of any differences.



*Fig 16: Cp along an NACA0010 airfoil with a reduced mesh stretching , M = 0.8*

*IV.F SLOR Efficiency*

SLOR works well enough for this transonic problem. The simulation converges in a reasonable amount of time and produces results that seem verifiable. However, it could be faster by considering additional methods to speed up convergence in the supersonic regions, which currently relies on a G.S style scheme and is quite slow. Something like ADI may be faster to avoid such problems. These convergence characteristics are generally what I expect: the solution is quite quick to converge at the beginning of the iteration as the SLOR converges the subsonic regions. But, the convergence slows as the upwinding scheme iterates through the supersonic regions.

What if I don’t set omega = 1 in the supersonic regions? Theory likely dictates that the solution would become unstable because the solution is being advected too quickly between points. But, when I attempted to run this, the solution converged like it typically did! The solution remained stable, indicating two things: 1) there could be an error in my implementation or 2) the upwinding does remain stable because points that go unstable may become subsonic, where the central scheme stabilizes them. Answers to wind tunnel and grid parameter effects are found in the previous section

V. Summary of Report

In this report, I first build the governing equations for compressible flow over an airfoil from first principles, making assumptions limiting vorticity and frictional effects, which makes this solver reasonable for M = 0.3-0.9 flows over a low angle of attack airfoil. Caution must be taken when considering wake effects or the potential of flow separation, which cannot be modeled well here. Next, I constructed a stencil of the solver for subsonic and supersonic points in the mesh with a term that “switches” on or off a stencil based on the local flow speed.

This was then implemented in Python on an intel i5 processor. A structured, stretched grid mesh was first constructed over the airfoil’s upper boundary, and care was taken to ensure this matched previous, verified work on mesh construction. Then, by setting gamma = -1, the incompressible flow could be extracted, and an exact solution was compared. This agreed well, so the transonic flow was simulated next.

Four transonic cases were run: M = 0.75 and M = 0.8 with an NACA or a biconvex airfoil at 10% thickness in both cases. The key computation results are found in table 1 (timing, number of supersonic points, and convergence) with the results found in the following figures. The results appear to match independent NASA reports, indicating that the solver appears correctly implemented! Extending this, pressure contours simulated in and out of a wind tunnel appear to match theory and expectations. The fastest parts of the flow match the low pressure areas, and the shock line is well resolved in these plots. Caution must be taken to how the mesh is constructed: when the stretching was limited before the airfoil, the shock appeared weaker than larger stretching.

This code appears appropriate for moderate M, zero AoA airfoil simulations, but there two extensions that could make it more interesting or generalizable to larger flight envelopes. First, this assumes a steady shock solution, but by keeping temporal derivatives in the full potential flow, we may use this code to create an unsteady TSD to see how the supersonic behavior develops. Additionally, this code could be extended to include nonzero AoA flows by modifying how the terms are solved in the full-potential equations. To calculate lift in these cases, we would need to then simulate both the top and bottom of the airfoil.

VI. Appendix

Listed in the attached pdf. The inputs are found in the commented subheadings and the outputs are 6 figures that detail the solution.

1. Report found here: https://ntrs.nasa.gov/api/citations/19850010648/downloads/19850010648.pdf [↑](#footnote-ref-1)