Online Euler Solver Activity

Introduction

As an engineer, the main focus of your job is to solve problems in a way that is not only results-driven but also feasibility-driven. After all, what is a great product that is too expensive to produce and market or a cheap product that does not do the job well?

One of the ways that modern technology helps engineers is to give us computational tools. This allows testing on a large scale for multiple different traits without the expense of having to build and break prototypes, and the technology has come so far that we are now able to have the computer make "intelligent" choices as to size and shape to further economize the design process.

However, while computational engineering is a growing field, the results still must be tested against experimental outcomes until we have determined that the mathematics adequately represents the physics.

Background Information

The previous statement is a very important one and deserves to be discussed further. In order to use a computer to emulate physics, a system of equations must exist that can be solved to give important values (such as velocity, energy, and density, the case of the two-dimensional Euler equations) for the flow field at a given time. These values can then be compared with experimental values for the same CAD model that was built and tested in a wind or water tunnel.

Fortunately, the work of mathematicians and physicists on finding model equations has been extensive. Even more importantly, both through mathematical and experimental means, many of these sets of equations have had enough use that eccentricities and assumptions have been reasonably documented.

The equation set we use here is named after Leonhard Euler (1707-1783), and it is a reduction of the Navier-Stokes Equations that govern fluid flow. We use this set for a number of reasons, amongst the most important of them being that the assumptions associated with it are well-documented and solution times are reasonable.

This system allows us to look at four variables at any given point (density, x-velocity, y-velocity, and energy), but these may be combined to give us even more relevant information (temperature, pressure, coefficient of pressure, etc.). The only major concept that is assumed away here is that of viscous effects. Despite the fact that it is difficult to see with the naked eye, all fluids have some viscosity, which means that they "stick" to the medium they are passing over. This can cause turbulence (which is chaotic and difficult to model accurately every time), vortex shedding, and other phenomena.

While the Euler equations do not adequately predict these phenomena, and thus more complex methods are often employed to learn very detailed information for high performance aerodynamic applications, they do predict something very important: shocks. You may have heard a sonic boom before, where the pressure waves emanating from a jet as it crosses the threshold from subsonic to supersonic are compacted. This actually creates significant pressure fluctuations that contribute to the lift and drag on the aircraft. These can be seen as spikes in the pressure coefficient (Cp) plots that the Euler Emulator produces.

Solution Methods

It is interesting to note that your Calculus course is at the very root of many of the concepts that go into solving the Euler equations. In fact, the three equations that make up the Navier-Stokes System are integrals over a surface (two-dimensional integral) or over a volume (three-dimensional integral). The equations that form the three fluid conservation laws we are interested in are as follows:

Conservation of Mass

$$\frac{d}{dt}\iiint_{V(t)} \rho \, dV + \iint_{S(t)} \rho \left(\vec{u} - \vec{u}_s\right) \cdot \vec{n} \, ds = 0$$

Conservation of Momentum

$$\frac{d}{dt}\iiint_{V(t)} \rho \vec{u} \, dV + \iint_{S(t)} \rho \vec{u} \left(\vec{u} - \vec{u}_s\right) \cdot \vec{n} \, ds - \iiint_{V(t)} \rho \vec{f} \, dV - \iint_{S(t)} \vec{P} ds = 0$$

Conservation of Energy

$$\frac{d}{dt}\iiint_{V(t)} \rho\left(e + \frac{1}{2}\vec{u} \cdot \vec{u}\right) dV + \iint_{S(t)} \rho\left(e + \frac{1}{2}\vec{u} \cdot \vec{u}\right)\left(\vec{u} - \vec{u}_s\right) \cdot \vec{n} \, ds$$
$$- \iint_{S(t)} \vec{u} \cdot \vec{P} ds - \iiint_{V(t)} \rho \vec{u} \cdot \vec{f} \, dV + \iint_{S(t)} \vec{q} \cdot \vec{n} ds = 0$$

Now, these are actually Integro-Differential Equations, and they are complex enough that they cannot be solved exactly by any direct method we currently have. So, we must solve them approximately using iterative methods.

The first step in solving such a system is to discretize the space in question. This is much like the Riemann Sum method of solving a regular integral. As is shown in Figure 1, to find the area under the given curve, one can use easily computed rectangular areas that both overshoot and undershoot the function. Then, either by using maximum and minimum values or left and right endpoint values, the actual area under the curve is computed, with the approximation getting better as $\Delta x \to 0$.

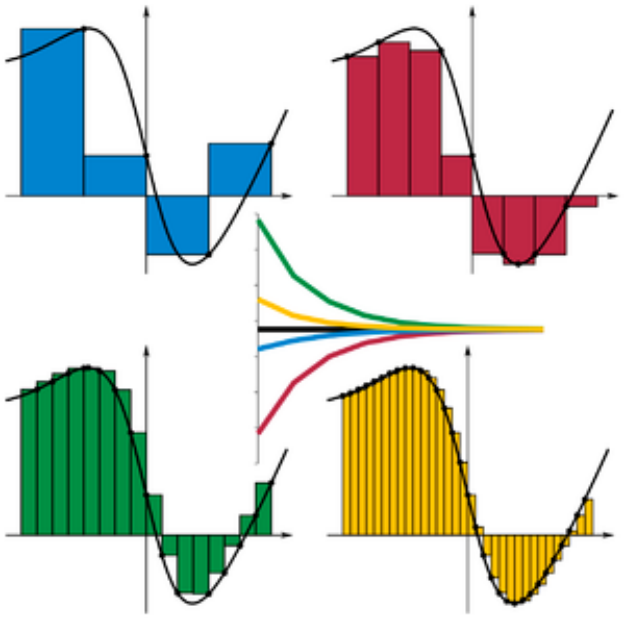

Figure 1: Riemann Sums

While the control areas/volumes that are used to discretize space in a real-world problem are not as trivial as rectangles, their area/volume can still be computed easily and effectively with the help of vector mathematics. As such, we can then determine the flux of properties through the space, and one technique for doing just that is very similar to the Riemann problem. Effectively, we view the value of the dependent variables in each area/volume as piecewise constant and use different methods to determine the flux through the faces (where the value changes and thus is discontinuous). The details of how this is done and the differences between 1st and 2nd order extrapolations is left out, but handling discontinuities that occur in our discretization is paramount. Not only does this make it possible to observe shocks (which are inherent discontinuities) but it also makes for a consistent evaluation of fluxes between volumes with different dependent variable values.

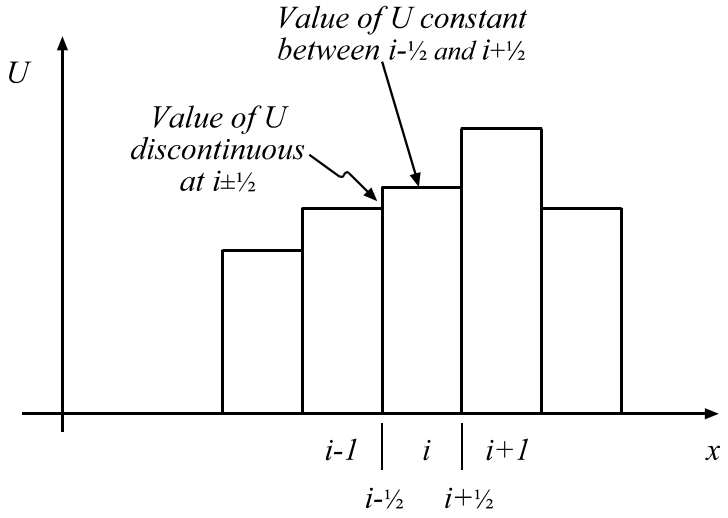

Figure 2: Riemann problem relates discontinuities between control volumes

The second major step in determining a solution involves using either an iterative or a direct method to solve the system of equations. While direct methods are definitely simpler, iterative methods must be used due to the non-linearity of the problem.

While these methods are employed on systems of equations using linear algebra, a more intuitive example can be given using scalar equations. Consider the following linear equation:

$$5x - 10 = 0$$

This can easily be solved by a direct algebraic method:

$$\begin{array}{l} 5x - 10 = 0 \\ +10 \qquad +10 \\ \_\_\_\_\_ \\ 5x = 10 \\ \div 5 \quad \div 5 \\ \_\_\_\_\_ \\ x = 2 \end{array}$$

Now, consider the following quadratic equation:

$$x^2 + x - 2 = 0$$

This cannot be solved directly due to the non-linear nature of the equation. Since this is such a common problem, methods like factoring and using the quadratic formula have been developed to ease the need for iterative solutions. However, these methods do not extend beyond fourth degree polynomials, since by the Abel-Ruffini Theorem, there is no guarantee of a general solution to fifth degree and higher polynomials. So, generality is best served by using an iterative method, such as Newton's Method.

Newton's method can be expressed as:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Where x_0 is the initial guess, $f(x_0)$ and $f'(x_0)$ are the function and its derivative evaluated at the initial guess, respectively, and x_1 is the next closer value to an actual root of the function. See Figure 3 for a visual representation of Newton's method, where the "X" is the final, desired solution (root).

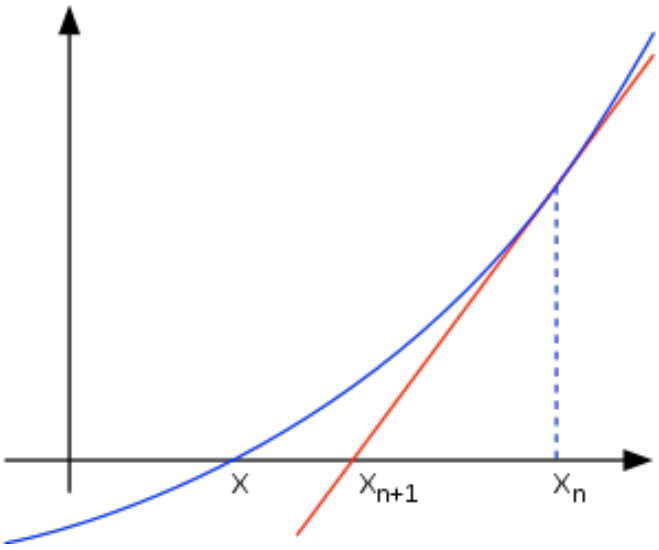

Figure 3: Newton's Method

It is useful to note that this method is an algebraic manipulation of the slope of a secant line (or a tangent line as $x_0 - x_1 \to 0$) where the value of the function at x_1 is a root ($f(x_1) = 0$):

$$f'(x_0) = \frac{f(x_0) - 0}{x_0 - x_1}$$

Now, Newton's method should converge fairly quickly if we make a good initial guess. In solving the Euler equations, we often use the free-stream parameters as our initial guess or we use the results from a pervious solution that was similar as the initial guess. Here, since we know that the roots of $x^2 + x - 2 = 0$ are $x = -2$ and $x = 1$, let us choose 0 as our initial guess. Recall that $f'(x) = 2x + 1$.

$$x_1 = 0 - \frac{f(0)}{f'(0)} = -\frac{-2}{1} = 2$$

So, we have not hit either target on the first try, so let us continue:

$$x_1 = 2 - \frac{f(2)}{f'(2)} = 2 - \frac{4}{5} = 1.2$$

We are now much closer to one of our roots. Let us continue:

$$x_1 = 1.2 - \frac{f(1.2)}{f'(1.2)} = 1.2 - \frac{0.64}{3.4} \approx 1.012$$

If we were to continue down this path, it should be obvious that we will converge to $x_1 = 1$. It is instructive to see that if we begin at -3, we will end up at the other root such that $x_1 = -2$. Try this on your own!

The above process, even though it is scalar, shows us glimpses of some important phenomena regarding iterative methods. The first piece of information is that your initial guess changes your final outcome. While this seems to make the method non-deterministic and thus less than useful, it can be shown that so long as you pose your problem well (and use a slightly "smarter" iterative method), you will get the proper solution regardless of your initial guess.

Another facet of Newton's method that can be paralleled to a concept in our Euler Solver is the amount of times that you solve the formula before you determine that it is close enough to the number you want and stop. This can be compared to choosing the number of point-iterative iterations in the solver interface, where one stops the iterations even before the solution is fully converged since it is close enough to move on to the next step.

The final concept that we will draw from this demonstration is how our choice of CFL number in the solver interface will affect both our ability to converge and how quickly we do so. We often speak of the CFL number as the "time step", even though we are not advancing our solution through time. This "pseudo time" is simply the states that the solution goes through before it stops changing and reaches steady state. Choosing your time step wisely allows you to reach a solution in a reasonable amount of time without moving so quickly that the physics is left in the dust. The parallel to be drawn between this concept and Newton's method is that if we were to try to advance along the function too quickly, we might actually skip over the root we were trying to find or even get a number that causes division by zero. This must be avoided when using iterative methods.

Converging to a Usable Solution

One major point that has been glossed over in the preceding examples is that just because your solution has reached a steady state does not guarantee that your solution is as highly resolved as the physics can be. This is due to the fact that your discretization itself was a guess. In fact, there can be a lot more going on in the flow field than your initial steady state solution lets on.

There are multiple ways of rectifying this situation. The first is to simply continue to refine your grid and solve on finer and finer meshes until the phenomena you are looking to reproduce show up in the final solution (i.e, a shock). This is not the best method, however, since 1) it causes your system of equations to grow very large and cumbersome as you add more points to the mesh, causing slower computation and 2) it can cause "phantom" phenomena to show up in areas that are too highly resolved for the physics being modeled.

The way we circumvent this is to use a technique called adaptive mesh generation. In the interface, you will see a category called "Refinement Step". As we move from "Coarse"

to “Finest”, there is more going on than simply randomly adding points. Effectively, we look at the previous solution, determine areas of high gradient (in other words, areas where the change in the dependent variable from cell to cell is great), and refine just those areas. This allows us to determine a very fine-grained solution without extraneous information or excess computation.

The following series of figures (Figures 4 – 7), show the process of adaptive refinement in a nutshell. First, a mesh on a NACA0012 airfoil that was generated with no knowledge of the solution is presented, as well as the solution found on that mesh. Notice that the shock on top of the airfoil is beginning to form. Then, after using that solution data to generate a mesh where refinement is high in the area of the shock, you will notice that the final solution has a much more highly resolved shock area and more accurately depicts the performance of the airfoil at Mach 0.8 (the speed of sound is Mach 1.0, but this is a transonic case nonetheless) with an angle of attack of 1.25°.

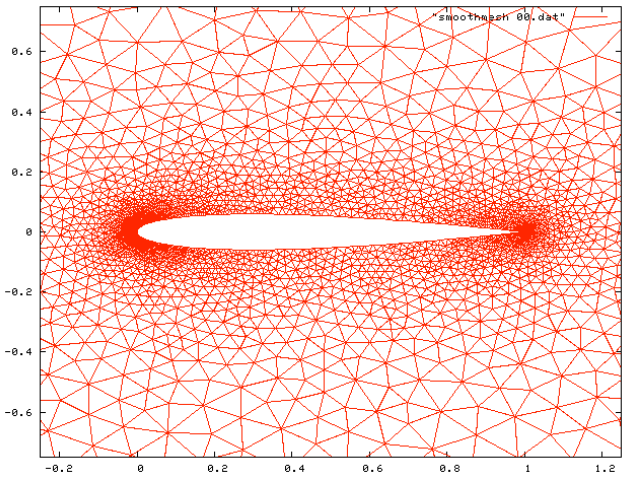

Figure 4: Original mesh on NACA0012 airfoil

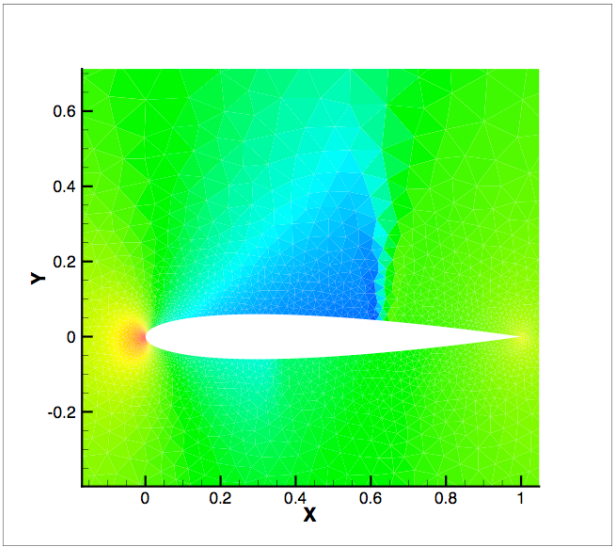

Figure 5: Original solution on NACA0012 airfoil (contours represent Cp)

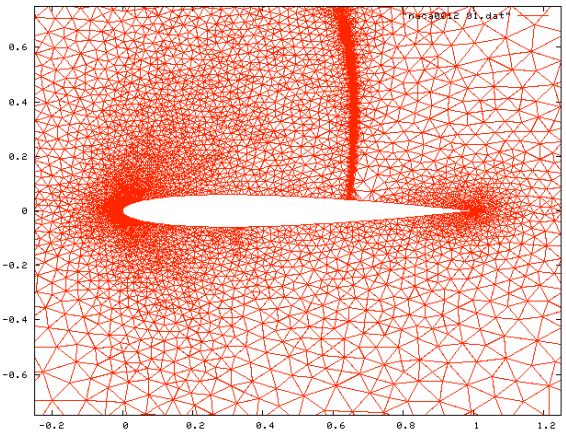

Figure 6: Adapted mesh on NACA0012 airfoil

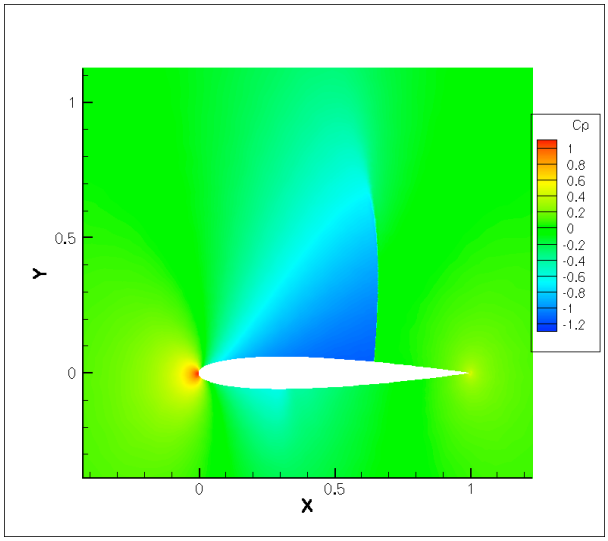

Figure 7: Final solution on NACA0012 airfoil (contours represent pressure coefficient)

Now, let us try a few cases ourselves to get comfortable with the changing solution as we go from one mesh refinement to the next.

First, let us look at the interface (Figure 8). You will notice six fields from which you can select values, three action buttons, and four tabbed image fields. The general order of business we will be following is to:

1) Choose an angle of attack, and generate a mesh.
2) Choose an order of accuracy, a mach number (velocity of the flow field), a number of point iterative iterations (degree of accuracy to which the system is solved in between updates of the dependent variables), a CFL number (the speed with which we step through pseudo-time to the steady state), and a mesh refinement level (that we will increase after each solution has been generated…we always begin with a coarse mesh).

3) Look at the solutions generated, possibly even comparing residual plots with different numbers of point iterative solves and CFL numbers.
4) Generate a new, adapted mesh on the solution and repeat steps 2 and 3.

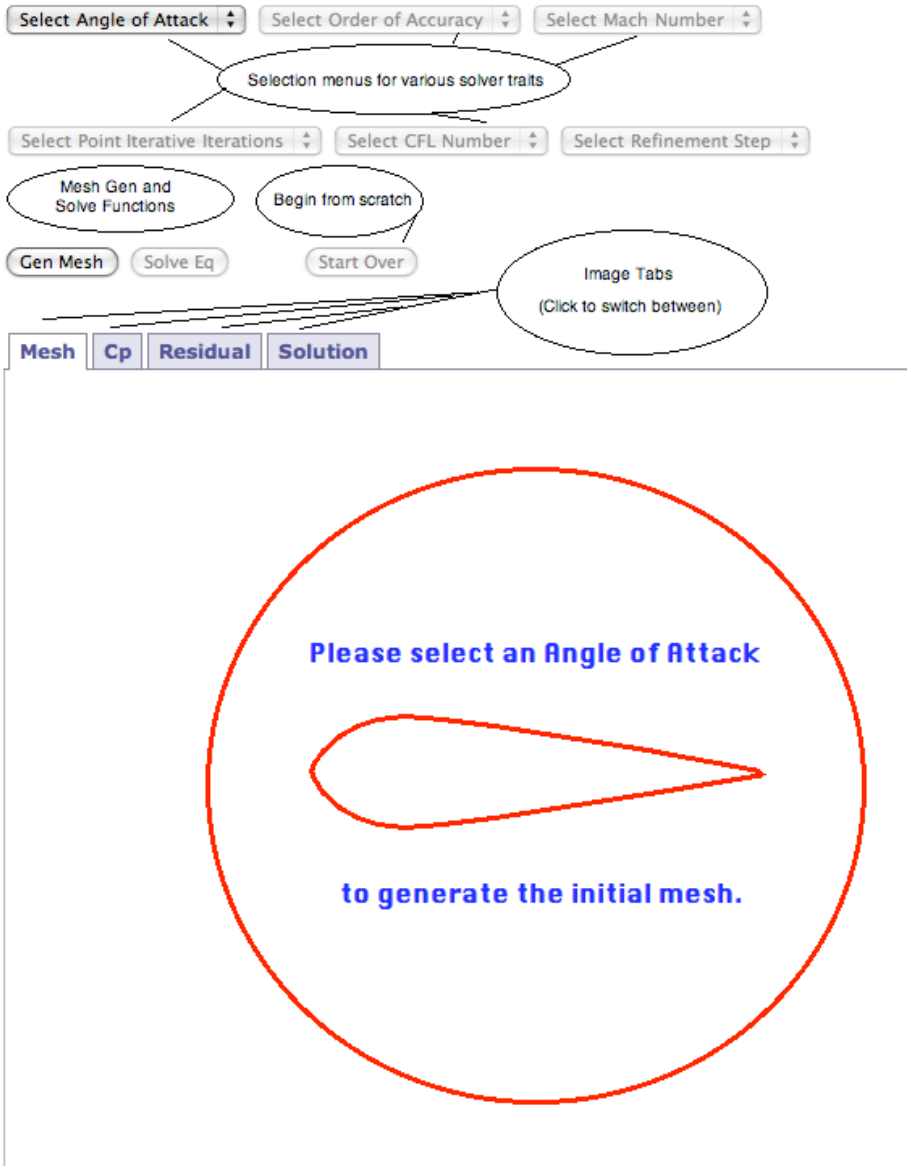

Figure 8: Interface for the Online Euler Solver Emulator.

So, let us begin running some cases and discussing the results as a group.

1) SUBSONIC: alpha = 0, mach = 0.55, order = 1, Point Iterative = 20, CFL = 500 Run this case as stated, then try different CFL numbers and Point Iterative counts. Look at the residual plots and discuss the convergence behavior at each CFL and Point Iterative count. Also, look at the Cp and Solution plots and get a baseline for the types of results seen in this symmetric case

2) TRANSONIC: alpha = 2, mach = 0.75, order = 1, Point Iterative = 20, CFL = 500
alpha = 2, mach = 0.75, order = 2, Point Iterative = 20, CFL = 500

Run these two cases and consider the solution plots as well as the Cp plots, spotting any differences. What does this tell us about order of accuracy? Also, what differences do we see in the solution plots from our previous case of alpha = 0?

3) SUPERSONIC: alpha = 1, mach = 1.05, order= 1, Point Iterative = 200, CFL = 50
alpha = -1, mach = 1.05, order = 1, Point Iterative = 200, CFL = 50

Since the airfoil is a symmetric one, you should see that these two cases are inverted versions of the other. Based on what we know about flight, does this make sense?

4) Now, let's try the above case running the same parameters at each level of refinement. What do you notice about the changes in the mesh each time? Are they in areas of interest? How does the new mesh affect the resolution of the solution?

Results

Now that we have looked at some simulated results, let us try to determine some patterns in the physics they emulated. It is instructive to know some basic aerodynamic principles before we delve in.

Effectively, lift is generated by the pressure below a wing being greater than the pressure above the wing, and drag is generated by the very opposite effect. There are many other contributing factors to drag such as skin friction drag, viscous effects, and turbulent flow, but let us focus on the results we can see from the information given in our simulated flow solution: the regions of high and low pressure seen on the contour plot.

Also, there is the idea of shock formation. Shocks are formed when the speed of the flow past an airfoil goes transonic, and they can be seen in the Cp plots as sharp fluctuations in the gradient of the Cp. These phenomena cause pressure changes that can seriously affect the subsonic lift and drag specifications of the airfoil.

So, let us look at some results, ask ourselves a couple of questions, and discuss our answers as a group:

1) Look at the solution plots of the following cases (on coarse refinement):
   a) mach = 0.65, alpha = 0°, order = 1, cfl = 50, point iterative = 20
   b) mach = 0.65, alpha = 2°, order = 1, cfl = 50, point iterative = 20
   c) mach = 0.65, alpha = 4°, order = 1, cfl = 50, point iterative = 20

What happens to lift and drag as we increase the angle of attack? Based on what we have learned about Bernoulli's Principle, does that make sense?

2) Look at the mesh, solution, and Cp plots for the following case (on coarse, finer, and finest refinement):
    a) mach = 0.75, alpha = 2°, order = 1, cfl = 50, point iterative = 20

What do you see occurring on the mesh as we move from coarse to finer refinements? Does it change the resolution of the fish-tail shock in the solution as well?

Finally, it is instructive to see that what we do in computational engineering is also being done by experimentalists, often for many more years due to the fact that computers large and fast enough to do real-life simulations are a fairly recent development. As a consequence, much of the work being done currently is being done in both environments as a way of validation. After all, we are all uses the same mathematical theory as the basis of both types of experiments, and it is incumbent on mathematicians, scientists, and engineers to determine why any differences in results and predictions exist. Is it experimental error? Is it computational inaccuracy? Is there something missing in the theory?

The following images were taken from "Investigating the Aerodynamic Response of a NACA 0012 Airfoil with Gurney Flap Configuration"[1]. While our airfoil did not have a Gurney Flap, it is still instructive to see a three-dimensional representation of the object we are simulating in two dimensions as well as experimental results using smoke injection to show similar phenomena to what we are emulating with our solver at slightly smaller angles of attack.

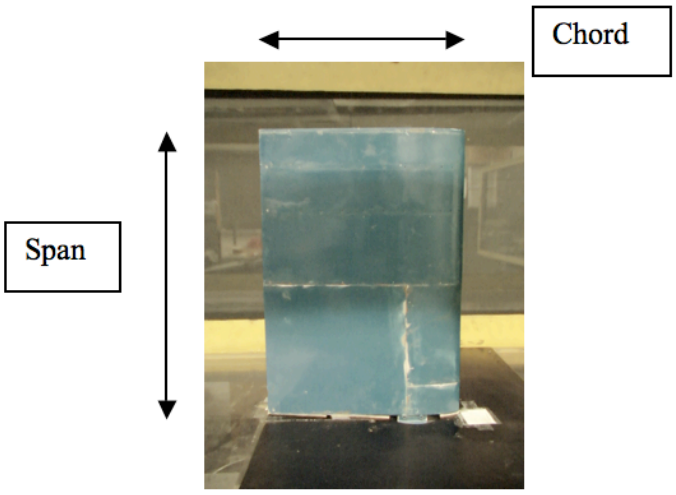

Figure 9: NACA 0012 Experimental Airfoil (top view): The terms chord and span are used to refer to the above denoted lengths on an airfoil.

[1] Author: Ty Mukherji, Senior Pratt Fellow. Advisor: Dr. Earl Dowell. Published: April 25, 2006

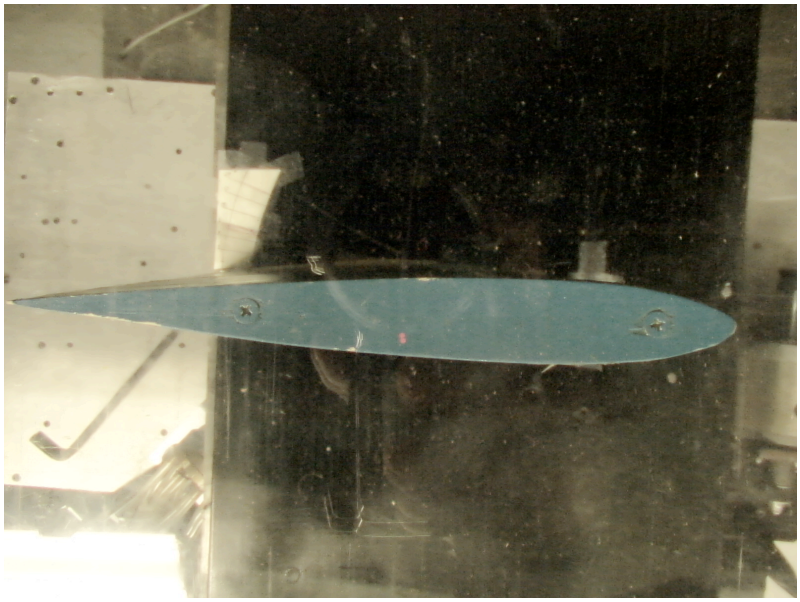

Figure 10: NACA 0012 Experimental Airfoil (side view): This is the angle at which we are viewing all of our solution data.

Figure 11: Smoke injected flow over a NACA 0012 airfoil at 11° angle of attack, showing flow separation.

Additionally, you can see videos of simulations of the flow separation at 12° angle of attack in avi videos on the following site from UTA:

http://www.uta.edu/faculty/hshan/research/gallery.shtml#NC0012

I hope that you all learned a great deal, and it has been a pleasure having you at UTC and the SimCenter: National Center for Computational Engineering!