Solving the Two-Dimensional Euler Equation using a Point-Iterative Linear Solve and Green's Theorem Gradients

Vincent Charles Betro

April 19, 2007

Abstract

One subset of the Navier-Stokes equations is the Euler Equations. Despite the fact that these are non-linear in nature, they can be solved using an LU-factorization by linearizing the fluxes. The use of the chain rule in creating analytic flux jacobians, Cuthill-McKee reordering in clustering non-zero values about the diagonal, and CFL ramping to allow for more correct gradient calculation are all key elements of the approach taken. The end product was a second order implicit solver run on a NACA0012 airfoil as a test case.

Nomenclature

Ω	A control volume
Γ	The boundary of a control volume
$\vec{F}$	The flux over a control volume
$\vec{n}$	The unit normal pointing out of the control volume at a given face
Q	A vector composed of density, x-y momentum, and total energy at a given node
ρ	Density at a given node
u	X velocity at a given node
v	Y velocity at a given node
e_t	Total energy at a given node
e_i	Internal energy at a given node
γ	Specific gravity (here, assumed to be 1.4)
c	Speed of sound in air
$\bar{u}$	Magnitude of the velocity vector
p	Pressure
R_i	Residual
α	Angle of attack

0.1 Introduction

One primary aim of the study of Computational Fluid Dynamics is to solve the Navier-Stokes equations using numerical techniques (Anderson07). To that end, one of the model equations that can be used as a building block is the two-dimensional inviscid Euler Equation:

$$\frac{\delta}{\delta t}\int_{\Omega} Q d\Omega + \int_{\Gamma} \vec{F}\cdot\vec{n} d\Gamma = 0 \tag{1}$$

where Ω is a control volume, Γ is the boundary of the control volume, and $\vec{n}$ is the unit normal pointing out of the control volume at a given face. In the two-dimensional version of Equation 1, Q is a vector of the mass, x-y momentum, and energy per unit volume as follows:

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e_t \end{bmatrix}$$

and $\vec{F}$ is the flux over the faces, both viscous and inviscid. Here, since we are using the Euler Equation, only the inviscid aspects will be discussed.

To discretize Equation 1, Backward Euler is used for the time derivative. It should be noted here that VanLeer flux vector splitting is used. As a consequence, the flux vector is split as such for the left and right states for subsonic flow (non-dimensionalized):

$$\vec{F}^{\pm} \cdot \vec{n} = \begin{bmatrix} \pm\frac{1}{4}\rho c \left(\frac{\bar{u}}{c} \pm 1\right)^2 \\ F_1^{\pm}\left(\frac{\vec{n}_x}{\gamma}(-\bar{u} \pm 2c) + u\right) \\ F_1^{\pm}\left(\frac{\vec{n}_y}{\gamma}(-\bar{u} \pm 2c) + v\right) \\ F_1^{\pm}\left(\frac{(-(\gamma-1)\bar{u}^2 \pm 2(\gamma-1)\bar{u}c + 2c^2)}{\gamma^2-1} + \frac{(u^2+v^2)}{2}\right) \end{bmatrix} \tag{2}$$

where c is the speed of sound and $\bar{u} = \vec{n}_x u + \vec{n}_y v$. If the magnitude of the velocity of the node to the left of the face is supersonic and flowing out of the control volume, then $\vec{F}^{+}$ becomes (non-dimensionalized):

$$\vec{F}^{+} \cdot \vec{n} = \begin{bmatrix} \rho\bar{u} \\ \rho u\bar{u} + \vec{n}_x p \\ \rho v\bar{u} + \vec{n}_y p \\ (e+p)\bar{u} \end{bmatrix} \tag{3}$$

This is identical to the change in $\vec{F}^{-}$ when the magnitude of the velocity of the node to the right of the face is supersonic and flowing into the right node's control volume. In both cases, the opposite signed $\vec{F}$ is $\vec{0}$ (VanLeer82).

Also, the fluxes on the far-field boundaries were taken to be free-stream (zero) and those at the inner, inviscid boundaries were taken to be of the form:

$$F = \begin{bmatrix} 0 \\ \vec{n}_x p \\ \vec{n}_y p \\ 0 \end{bmatrix}. \tag{4}$$

The integral of the fluxes through the faces of each control volume is computed by cycling through the faces and summing the appropriately signed component (positive normal always points *out* of the control volume) to the residual of each node. The only caveat is that in order to close off the boundary segments, a three-quarters share is taken from the closer node and a one-quarter share from the further node, using the flux from Equation 4. This sum for each node will be denoted from here on as R_i.

Since the code can be run both first and second order implicit, it should be noted that when running second order, the extrapolated values of Q (see Equation 5) are used to evaluate the fluxes, where $\vec{r}$ is the distance from the node to the midpoint of the face. Since these extrapolated values are used to reconstruct Q, Linear Least Squares would have been the best method for computing gradients due to its ability to give a more accurate reconstruction (Anderson07). Here, Green's Theorem was used to generate the gradients, yet the extrapolated values still provided good results.

$$\vec{Q}_{face} = \vec{Q}_{node} + \nabla\vec{Q}_{node} \cdot \vec{r} \tag{5}$$

Thus, our discretized, implicit equation takes the following shape:

$$A\left(\frac{Q_i^{n+1} - Q_i^n}{\Delta t}\right) + R_i^{n+1} = 0 \tag{6}$$

However, the values of the fluxes at the $n + 1^{th}$ time step are not known, so Taylor Series must be used to derive the following expression for R_i^{n+1}:

$$R_i^{n+1} = R_i^n + \sum \frac{dR_i}{dQ_j} \Delta Q_i^n \tag{7}$$

where j is a neighboring node (that shares an edge). Substituting Equation 7 into Equation 6 and noting that $\Delta Q = Q^{n+1} - Q^n$, the result is:

$$\left(\frac{A}{\Delta t} + \sum \frac{dR_i}{dQ_j} \right) \Delta Q_i^n = -R_i^n \tag{8}$$

The quantity $\frac{dR_i}{dQ_j}$ is obtained by taking the derivative of each primitive variable in Q with respect to each element in Q. For example, $\frac{d\rho}{dQ}$ would be the following vector:

$$\begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

.

Now, the derivative of each element in the flux (Vectors 2 and 3) is taken with respect to each of the primitive variables, and the chain rule is applied as such for the case of ρ (Anderson07):

$$\frac{dR}{dQ} = \frac{dR}{d\rho} \frac{d\rho}{dQ}$$

.

0.2 Implementation

In order to implement this scheme numerically, a valid mesh was needed and initial free-stream values from which to initialize the flow field. An angle of attack in radians, α, and velocity, $Mach$ number, were input from which the following non-dimensionalized values

were obtained to place in the previous equations:

$$\rho = 1$$

$$c = 1$$

$$p = \frac{\rho c^2}{\gamma}$$

$$u = Mach\, cos(\alpha)$$

$$v = Mach\, sin(\alpha)$$

$$e_i = \frac{p}{(\gamma - 1)\rho}$$

$$e_t = \rho(e_i + \frac{1}{2}(u^2 + v^2))$$

In order to non-dimensionalize, the following equivalences were used , where the hat denotes the non-dimensionalized value and the infinity denotes the reference value for the given problem:

$$\hat{\rho} = \frac{\rho}{\rho_\infty}$$

$$\hat{c} = \frac{c}{c_\infty}$$

$$\hat{p} = \frac{p}{\rho_\infty c_\infty^2}$$

$$\hat{u} = \frac{u}{u_\infty}$$

$$\hat{v} = \frac{v}{v_\infty}$$

$$\hat{e_i} = \frac{e_i}{c_\infty^2}$$

$$\hat{e_t} = \frac{e_t}{c_\infty^2}$$

Equation 8 will yield a large, sparse matrix of 4×4 submatrices that will look like Matrix 9 using C++ indexing:

$$\begin{bmatrix} D_0 & \frac{dR_0}{dQ_1} & \dots & \frac{R_0}{dQ_{nn-2}} & \frac{R_0}{dQ_{nn-1}} \\ \frac{dR_1}{dQ_0} & D_1 & \dots & \frac{R_1}{dQ_{nn-2}} & \frac{R_1}{dQ_{nn-1}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{dR_{nn-2}}{dQ_0} & \frac{dR_{nn-2}}{dQ_1} & \dots & D_{nn-2} & \frac{R_{nn-2}}{dQ_{nn-1}} \\ \frac{R_{nn-2}}{dQ_0} & \frac{R_{nn-2}}{dQ_1} & \dots & \frac{R_{nn-1}}{dQ_{nn-2}} & D_{nn-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{nn-2} \\ x_{nn-1} \end{bmatrix} = \begin{bmatrix} R_0 \\ R_1 \\ \dots \\ R_{nn-2} \\ R_{nn-1} \end{bmatrix} \tag{9}$$

where D_i is a 4×4 with the additional $\frac{A}{\Delta t}$ on the diagonal and $\frac{dR}{dQ}$ is a 4×4 analytic flux jacobian matrix.

However, in order to save memory, a compressed row storage routine was used (Anderson07). In order to generate this compressed row storage, the `ja` array was created, which is the node to node hash table compressed into one array. Thus, the number of non-zero entries in Matrix 9 are now known and a compressed version of Matrix 9 can be dimensioned which will hereafter be called `mat`. The three-dimensional matrix `mat` will have as its first dimension the number of entries in `ja` and the next two dimensions of 4 and 4, since each node in the hash table will have a corresponding 4×4 submatrix. Each partition of `mat` can be accessed by creating the `ia` array, which holds the beginning index of each incomplete "row" in Matrix 9. Finally, the `iau` array is generated, which holds the locations in the `mat` matrix of the diagonal entries in the Matrix 9.

When creating the control volumes, the median-dual style control volume was used (Anderson07). This creates a control volume (red lines in Figure 1) around a point (blue node in Figure 1) such that the edges are composed of segments emanating from each triangle's edge midpoint and connecting to its centroid.

In building the left hand side matrix, a value for Δt was required, except on the boundaries, where the entire diagonal was initialized to ones. Since Δt varies by node, it was

obtained from the following formula:

$$\Delta t = \frac{CFL * len}{|\bar{u}| + c}$$

where len is the perimeter of the control volume and $c = \sqrt{\frac{\gamma p}{\rho}}$.

In order to solve Equation 8, a Point-Iterative solver was used with symmetric Gauss-Seidel updating and LU-factorization with forward and back substitution. First, the off-diagonal submatrices of each row were transfered to the corresponding right-hand side (where R_i resides) by multiplying them by the values of the current solution (at time n) and subtracting the result from R_i, which will be called R_i^*. Then, the system composed of a diagonally dominant 4×4, due to the addition of the $\frac{A}{\Delta t}$ term, and R_i^* is solved using LU-factorization to obtain the value of the solution at a given node at time $n + 1$. The RMS error was computed at the end of each time step using the following formula:

$$RMS = \frac{\sum_{i=0}^{nn} (R_i^* - D_i * x_i)^2}{nn} \tag{10}$$

Additionally, in order to cluster the dependent nodes that each node has as nearest neighbors along the diagonal of Matrix 9, which yields a faster solve since more updated values are available as the matrix is cycled through, Cuthill-McKee Reordering was used (Cuthill80). This, along with using symmetric Gauss-Seidel updating in the Point-Iterative solve, allows the most updated values of Q to be used for each step of the linear solve, and speeds convergence.

0.3 Results

The following results are based on a 4156 node, Delauney mesh around a NACA0012 airfoil. Second order spatial accuracy was used and ran for 1000 external iterations (such that the values of Q are updated from the ΔQ obtained from solving Equation 8 in between each iteration) or until reaching an RMS error of $1.0e - 15$.

One of the tests that was run was to vary the number of iterations of the Point-Iterative solve (which solves for ΔQ in Equation 8) that could be done before the update of Q. Using

lower numbers of point-iterative iterations than the choices given in the application before updating Q could have caused the update directions to be wrong, and thus the system would not converge. Since symmetric Gauss-Seidel was used, these errors would propagate very quickly.

The less point-iterative iterations that are used, the quicker the system will converge in time but with more non-linear iterations (see Figure 2). As more iterations are added in the Point-Iterative loop, the convergence takes less non-linear iterations, but significantly more time, since the changes in the ΔQ obtained from the Point-Iterative solve after more than 5 internal iterations become drastically less significant.

The maximum CFL is given as an input value and ramped up from 1 over 100 iterations using Formula 11.

$$CFL = CFL_{init} + \frac{iteration - 1}{99}(CFL_{max} - CFL_{init}) \tag{11}$$

As one can see in Figure 3, the low CFL numbers took the longest to converge, and a CFL of 500 had the most rapid convergence. The cases with CFL numbers of 1500 and 2500 failed to converge, and thus were not given as options in this application. This is due to the fact that the gradients are changing so rapidly in the second-order solve that the initial time stepping must be slowed down for a while to catch all the changes accurately, without yielding negative densities and pressures or causing zeros to appear on the diagonals of the flux jacobians. Once this change in gradients slows down, the CFL can be quickly raised and remain there to get a more rapidly converged answer.

In cases where an initial second order solve was not possible, a first order solve was run for 100 iterations, allowing the gradients o change more slowly and by lower magnitudes. Thus, the negative pressures and densities were avoided that would have caused zeros on the diagonal and thus failure to converge due to the poorly constructed flux jacobians. Once

the gradients stopped changing as rapidly, the process continued on with a second order run and dissipation occurred in the results as expected.

Additionally, the resemblance to the physical realities of shocks on an airfoil garnered from the solver was quite good despite the absence of flux limiters (which would alleviate the extra spikes seen in Figure 4). The shocks are displayed for both the top and bottom of the airfoil in the appropriate spots in terms of both coefficient of pressure and flow density, in Figures 4 and 5, respectively.

Conclusions

Using a reasonable amount of computing power, solutions that emulate reality were found in regards to shocks and pressure coefficient data on a NACA0012 airfoil. By adding such elements as flux limiters and better approximations of viscous flow over the boundaries, even better results will be achieved. However, such novel ideas as compressed row storage and chain rule analytic flux jacobians proved invaluable in the solution of the Euler Equations.

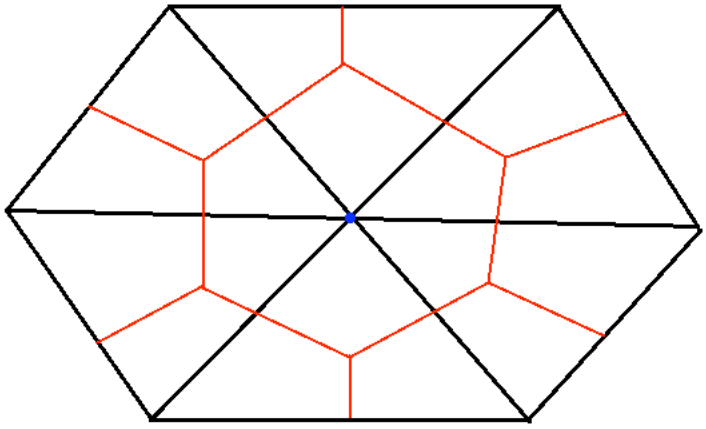

Figure 1: An example median-dual control volume (in red) for the blue node.

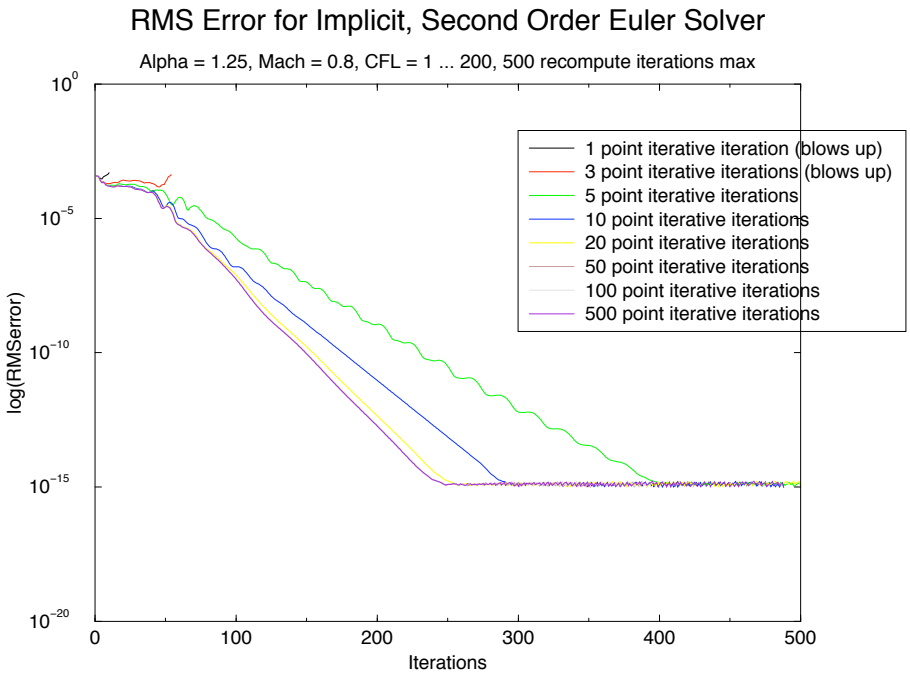

Figure 2: RMS error versus iterations for second order solve at varying numbers of Point-Iterative iterations

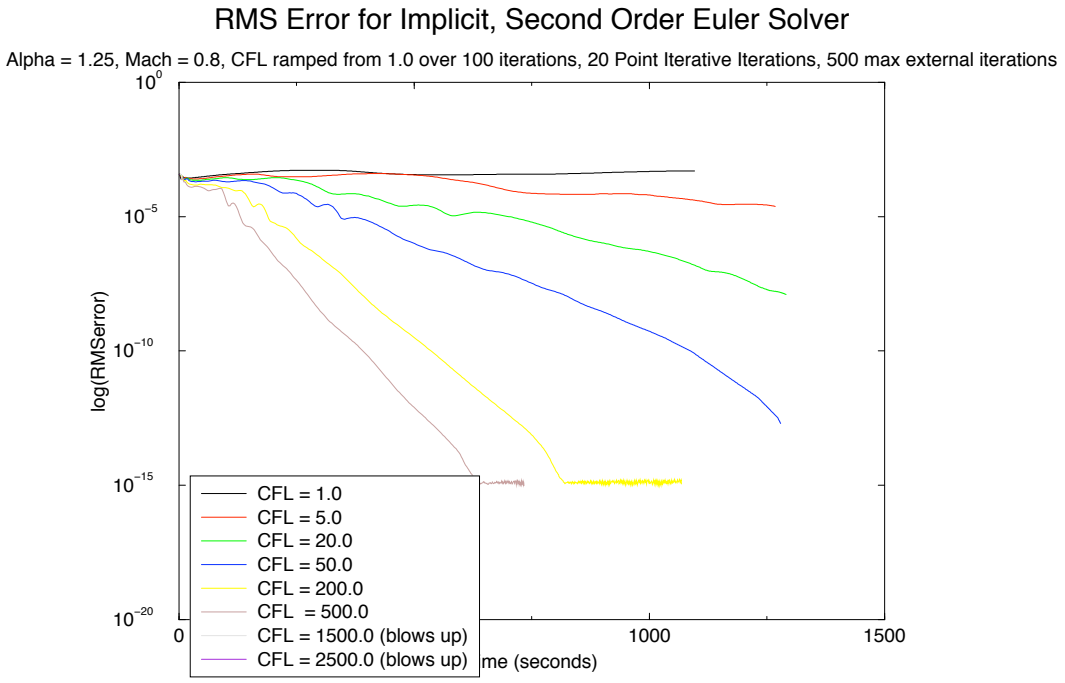

Figure 3: RMS error versus time for second order solve at varying maximum CFL numbers

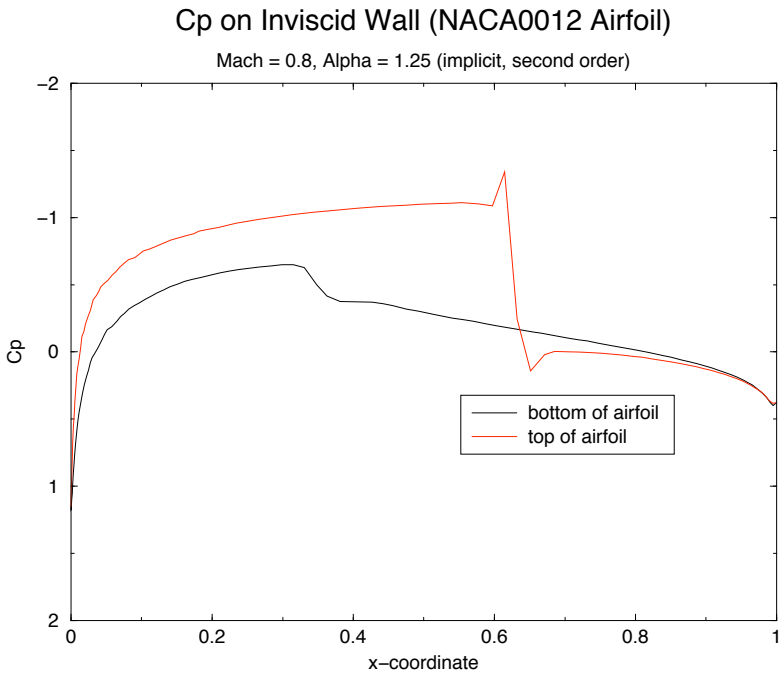

Figure 4: Coefficient of Pressure for NACA0012 airfoil, obtained from second order solve at Mach=0.8, Alpha = 1.25

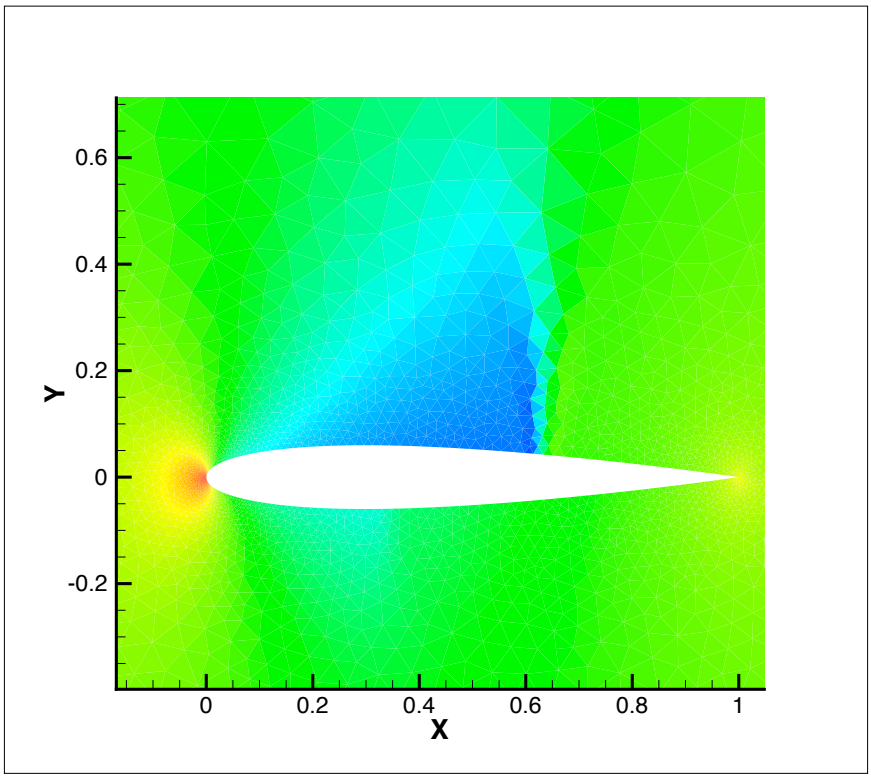

Figure 5: Density contours for NACA0012 airfoil, obtained from second order solve at Mach=0.8, Alpha = 1.25

Bibliography

[Anderson, 2007] Anderson, W. K. (2007). Introduction, Control Volumes, Higher Order Reconstruction, Point-Iterative Implicit Schemes, Inviscid Flux Evaluation/Inviscid and Farfield BC's. In *ENCM710 Course Notes (UT Sim Center '07)*.

[Cuthill-McKee, 1980] Cuthill, M., McKee, J. (1980). Node Reordering Scheme to Reduce Bandwidth of Sparse Block Matrices. *AMS Notices of the American Mathematical Society (June '80)*, pages 349–389.

[VanLeer, 1982] VanLeer, R. (1982). Flux Vector Splitting Techniques. *AIAA AIAA Bulletin (March '82)*, pages 149–199.